Assignment 1

Terry Stewart

January 15, 2014

1 SYDE556/750 Assignment 1: Representation in Populations of Neurons

- Due Date: January 29th at dawn (7:42am)
- Total marks: 20 (20% of final grade)
- Late penalty: 1 mark per day
- You can use any programming language you like, but it is recommended that you use a language with a matrix library and graphing capabilities. Two main suggestions are Python and MATLAB.

1.1 1) Representation of Scalars

1.1) Basic encoding and decoding

Write a program that implements a neural representation of a scalar value x. For the neuron model, use a rectified linear neuron model ($a = max(J, 0)$). Choose the α and J^{bias} values randomly such that the neurons have maximum firing rates uniformly distributed between 100Hz and 200Hz and the x-intercepts are uniformly distributed between -1 and 1. The encoders e are randomly chosen and are either +1 or -1 for each neuron. Go through the following steps:

- a) [1 mark] Plot the neuron responses a_i for 16 randomly generated neurons. (See Figure 2.4 in the book for an example, but with a different neuron model and a different range of maximum firing rates).
  - Since you can't compute this for every possible x value between -1 and 1, sample the x-axis with $dx = 0.05$. Use this sampling throughout this question)
- b) [1 mark] Compute the decoders d_i for those 16 neurons ($d = \Gamma^{-1}\Upsilon$; $\Upsilon_i = \sum_x a_i x dx$; $\Gamma_{ij} = \sum_x a_i a_j dx$). Report their values.
  - The easiest way to compute d is to use the matrix notation mentioned in the course notes. If A is the matrix of neuron activities (the same thing used to generate the plot in 1.1a), then $\Gamma = A^T A dx$ and $\Upsilon = A^T x dx$
- c) [1 mark] Compute and plot $\hat{x} = \sum_i (d_i a_i)$. Overlay on the plot the line $y = x$. (See Figure 2.7 for an example). Make a separate plot of $x - \hat{x}$ to see what the error looks like. Report the Root Mean Squared Error value.
- d) [1 mark] Now try decoding under noise. Add random normally distributed noise to a and decode again ($\hat{x} = \sum_i [d_i(a_i + \eta)]$). η is a random variable with mean 0 and standard deviation of 0.2 times the maximum firing rate of all the neurons. Create all the same plots as in part c). Report the Root Mean Squared Error value.

- e) [2 marks] Recompute the decoders d_i taking noise into account ($\Gamma_{ij} = \sum_x a_i a_j dx + \sigma^2 \delta_{ij}$). Show how these decoders behave when decoding both with and without noise added to a by making the same plots as in c) and d). Report the RMSE for both cases.
  - As in the previous question, σ is 0.2 times the maximum firing rate of all the neurons.
- f) [1 mark] Show a 2x2 table of the four RMSE values reported in parts c), d), and e). This should show the effects of whether or not noise occurs (i.e. whether η is added to a) and whether or not the decoders d are computed taking noise into account. Write a few sentences commenting on what this means.

1.2) Exploring sources of error

Use the program you wrote in 1.1 to examine the sources of error in the representation.

- a) [2 marks] Plot the error due to distortion E_{dist} and the error due to noise E_{noise} as a function of N, the number of neurons. Generate two different loglog plots (one for each type of error) with N values of [4, 8, 16, 32, 64, 128] (and more, if you would like). For each N value, do at least 5 runs and average the results. For each run, different α, J^{bias}, and e values should be generated for each neuron. Compute d under noise, with σ equal to 0.1 times the maximum firing rate. Show visually that the errors are proportional to $1/N$ or $1/N^2$ (see figure 2.6 in the book).
- b) [1 mark] Repeat part a) with σ equal to 0.01 times the maximum firing rate.
- c) [1 mark] What does the difference between the graphs in a) and b) tell us about the sources of error in neural populations?

1.3) Leaky Integrate-and-Fire neurons

Change the code to use the LIF neuron model:

$$a_i = \begin{cases} \frac{1}{\tau_{ref} - \tau_{RC} ln(1 - \frac{1}{J})} & \text{if } J > 1 \\ 0 & \text{otherwise} \end{cases}$$

- a) [1 mark] Generate the same plot as 1.1a). Use $\tau_{ref} = 0.002$ and $\tau_{RC} = 0.02$.
  - Note that you will need to compute new α and J^{bias} values that will achieve the desired tuning curves (uniform distribution of x-intercepts between -1 and 1, and maximum firing rates between 100Hz and 200Hz). Since you know two points on the tuning curve (the x-intercept and the point where it hits maximum firing), this gives you 2 equations and 2 unknowns, so you can find α and J^{bias} by substituting and rearranging.
- b) [2 marks] Generate the same plots as 1.1e), and report the RMSE for both.

1.2 2) Representation of Vectors

2.1) Vector tuning curves

- a) [1 mark] Plot the tuning curve of an LIF neuron whose 2D preferred direction vector is at an angle of $\theta = -\pi/4$, has an x-intercept at the origin (0,0), and has a maximum firing rate of 100Hz.
  - Remember that $J = \alpha e \cdot x + J^{bias}$, and both x and e are 2D vectors.
  - This is a 3D plot similar to figure 2.8a in the book. The main differences are that the preferred direction vector e is different, there is a different x-intercept and maximum firing rate (so α and J^{bias} are different), and you don't have to cut the tuning curve such that any value of x outside the unit circle is set to zero (this is only done in the book to help show the shape of the curve).

  - In the scalar case (that you did in question 1.1a), the maximum firing rate occurred when $x = 1$ for neurons with $e = 1$ and at $x = -1$ for neurons with $e = -1$. Of course, if the graph in 1.1a was extended to $x > 1$ (or $x < -1$), neurons would start firing faster than their maximum firing rate. Similarly, here the "maximum firing rate" means the firing rate when $x = e$. This should allow you to reuse your code from 1.3a) to compute α and J^{bias} for a desired maximum firing rate and x-intercept.
  - To generate 3D plots in MATLAB, see here
  - To generate 3D plots in Python, see here
- b) [1 mark] Plot the tuning curve for the same neuron as in a), but only considering the points around the unit circle. This will be similar to Figure 2.8b in the book. Fit a curve of the form $Acos(B\theta + C) + D$ to the tuning curve and plot it as well. What makes a cosine a good choice for this? Why does it differ from the ideal curve?
  - To do curve fitting in MATLAB, see here
  - To do curve fitting in Python, see here

2.2 Vector representation

- a) [1 mark] Generate a set of 100 random unit vectors uniformly distributed around the unit circle. These will be the encoders e for 100 neurons. Plot these vectors.
- b) [1 mark] Compute the optimal decoders. Use LIF neurons with the same properties as in question 1.3. When computing the decoders, take into account noise with σ as 0.2 times the maximum firing rate. Plot the decoders. How do these vectors compare to the encoding vectors?
  - Note that the decoders will also be 2D vectors
  - In the scalar case, you used x values between -1 and 1, with $dx = 0.05$. In this case, you can regularly tile the 2D x values ([1, 1], [1, 0.95], ... [-1, -0.95], [-1, 1]). Alternatively, you can just randomly choose 1000 different x values to sample.
- c) [1 mark] Generate 20 random x values around the unit circle. For each x value, determine the neural activity a for each of the 100 neurons. Now decode these values (i.e. compute $\hat{x}$) using the decoders from part b). Plot the decoded values and compute the RMSE.
- d) [1 mark] Repeat part c) but use the *encoders* as decoders. This is what Georgopoulos used in his original approach to decoding information from populations of neurons. Plot the decoded values this way and compute the RMSE. In addition, recompute the RMSE in both cases, but ignoring the magnitude of the decoded vector. What are the relative merits of these two approaches to decoding?
  - To ignore the magnitude of the vectors, normalize the length of the decoded vectors before computing the RMSE