SYDE 556/750— Assignment 1

Due date: January 27, 2020

Notes

- This assignment is worth 20 marks (20% of the final grade). The number of marks for each question is indicated in brackets to the left of each question.
- Please use Python 3 along with the `numpy` and `matplotlib` libraries to solve these assignments. In particular, fill out this Jupyter Notebook template:

  `https://github.com/astoeckel/syde556-w20/raw/master/assignment_01_template.ipynb`

- Clearly label any plot you produce, including the axes.
- You won't be judged on the quality of your code, but writing reusable functions will greatly simplify this and future assignments.
- Make sure to execute the Jupyter command "Restart Kernel and Run All Cells" before submitting your solutions. You will loose marks if your code fails to run on my computer.
- Send the completed notebook to `astoecke@uwaterloo.ca` using your University of Waterloo email account, make sure to include "[SYDE 556/750]" in the subject line. The deadline is at `23:59 EST` on January 27, 2020.
- There is a late penalty of one mark per day late.
- **Do not use or refer to any code from Nengo!**

1 Representation of Scalars

1.1 Basic encoding and decoding

Implement a neural representation of a scalar value x. For the neuron model, use a rectified linear neuron model, i.e., $a = G[J] = \max(J, 0)$. Choose the maximum firing $a^{\max}$ rates randomly (uniformly distributed between 100 Hz and 200 Hz at $x = 1$), and choose the x-intercepts ξ randomly (uniformly distributed between -0.95 and 0.95). Use those values to compute the corresponding α and J^{bias} parameters for each neuron. The encoders e are randomly chosen and are either $+1$ or -1 for each neuron. Go through these steps:

[0.5] a) *Computing gain and bias.* In general, for a neuron model $a = G[J]$ (and assuming that the inverse $J = G^{-1}[a]$ exists), solve the following system of equations to compute the gain α, and the bias J^{bias} given a maximum rate $a^{\max}$ and an x-intercept ξ.

$$a^{\max} = G[\alpha + J^{\text{bias}}]\,, \qquad\qquad 0 = G[\alpha\xi + J^{\text{bias}}]\,.$$

Now, simplify these equations for the specific case $G[J] = \max(J, 0)$.

[0.5] b) *Neuron tuning curves.* Plot the neuron tuning curves $a_i(x)$ for 16 randomly generated neurons following the intercept and maximum rate distributions described above.

📖 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, monotonously sample the x-axis with $\Delta x = 0.05$, i.e. there should be 41 monotonously increasing x values. Use this sampling throughout this question.

[1] c) *Computing identity decoders.* Compute the optimal identity decoder $\mathbf{d}$ for those 16 neurons (as shown in class). Report the value of the individual decoder coefficients. Compute d using the matrix notation mentioned in the course notes. Do not apply any regularization. A is the matrix of activities (the same data used to generate the plot in 1.1b).

📌 When performing the matrix inversion required to compute $\mathbf{d}$, the lack of regularization may result in numerical issues and corresponding warning messages. Try setting a fixed random seed (e.g. by calling `np.random.seed`) to reliably generate tuning curves that do not have this problem.

[1] d) *Evaluating decoding errors.* Compute and plot $\hat{x} = \sum_i d_i a_i(x)$. Overlay on the plot the line $y = x$. Make a separate plot of $x - \hat{x}$ to see what the error looks like. Report the Root Mean Squared Error (RMSE) value.

📖 See Figure 2.7 for an example.

[1] e) *Decoding under noise.* Now try decoding under noise. Add random normally distributed noise to a and decode again. The noise is a random variable with mean $\mu = 0$ and standard deviation of $\sigma = 0.2 \max(A)$ (where $\max(A)$ is the maximum firing rate of all the neurons). Resample this variable for every different x value for every different neuron. Create all the same plots as in part d). Report the RMSE.

[1] f) *Accounting for decoder noise.* Recompute the decoder $\mathbf{d}$ taking noise into account (i.e., apply the appropriate regularization, as shown in class). 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 all cases.

📌 As in the previous question, $\sigma = 0.2 \max(A)$.

[1] g) *Interpretation.* Show a 2x2 table of the four RMSE values reported in parts d), e), and f). This should show the effects of adding noise and whether the decoders d are computed taking noise into account. Write a few sentences commenting on what the table shows, i.e., what the effect of adding noise to the activities is with respect to the measured error and why accounting for noise when computing the decoders increases/decreases/does not change the measured RMSE.

1.2 Exploring sources of error

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

[2] a) *Exploring error due to distortion and noise.* 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 at least $[4, 8, 16, 32, 64, 128, 256, 512]$. 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 $\sigma = 0.1 \max(A)$. Show visually that the errors are proportional to $1/N$ or $1/N^2$.

- 📌 Do *not* add noise to A when computing the decoders in this question. Instead, appropriately account for noise when computing the decoders.
- 📖 See Equation 2.9 and Figure 2.6 in the book.

[1] b) *Adapting the noise level.* Repeat part a) with $\sigma = 0.01 \max(A)$.

[1] c) *Interpretation.* 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 rate approximation of the LIF neuron model

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

[0.5] a) *Computing gain and bias.* As in the second part of 1.1a), given a maximum firing rate $a^{\max}$ and a bias J^{bias}, write down the equations for computing α and the J^{bias} for this specific neuron model.

[0.5] b) *Neuron tuning curves.* Generate the same plot as in 1.1b). Use $\tau_{\text{ref}} = 2\,\text{ms}$ and $\tau_{RC} = 20\,\text{ms}$. Use the same distribution of x-intercepts and maximum firing rates as in 1.1, i.e., choose the maximum firing rates $a^{\max}$ as uniformly distributed between 100 Hz and 200 Hz at $x = 1$, choose the x-intercepts ξ as uniformly distributed between -0.95 and 0.95, and choose the encoders e as either -1 or $+1$.

[1.5] c) *Impact of noise.* Generate the same four plots as in 1.1f) (adding/not adding noise to A, accounting/not accounting for noise when computing $\mathbf{d}$), and report the RMSE both with and without noise.

2 Representation of Vectors

2.1 Vector tuning curves

[1] a) *Plotting 2D tuning curves.* 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 100 Hz.

- 📌 Remember that $J = \alpha\langle \mathbf{e}, \mathbf{x} \rangle + J^{\text{bias}}$, and both $\mathbf{x}$ and $\mathbf{e}$ are 2D vectors.
- 📌 In general, the maximum firing rate of a neuron is defined to occur when the input is of unit length along its preferred direction, i.e., $\langle \mathbf{e}, \mathbf{x} \rangle = 1$, which, for unit-length $\mathbf{e}$ is equivalent to $\mathbf{e} = \mathbf{x}$. Of course, if we increase the magnitude of the input vector beyond unit length, neurons would start firing faster than their maximum firing rate. Similarly, here the "maximum firing rate" means the firing rate when $\mathbf{x} = \mathbf{e}$. This should allow you to reuse your equations from 1.3a) to compute α and J^{bias} for a desired maximum firing rate and x-intercept.
- 🐍 To generate 3D plots in Python, see here.

📖 This is a 3D plot similar to Figure 2.8a in the book.

[1] b) *Plotting the 2D tuning curve along the unit circle.* Plot the tuning curve for the same neuron as in a), but only considering the points around the unit circle, i.e., sample the activation for different angles θ. Fit a curve of the form $c_1 \cos(c_2\theta + c_3) + c_4$ to the tuning curve and plot it as well.

📖 This will be similar to Figure 2.8b in the book.

🐍 To do curve fitting in Python, see here.

[0.5] c) *Discussion.* What makes a cosine a good choice for the curve fit in 2.1b? Why does it differ from the ideal curve?

2.2 Vector representation

[1] a) *Choosing encoding vectors.* Generate a set of 100 random unit vectors uniformly distributed around the unit circle. These will be the encoders $\mathbf{e}$ for 100 neurons. Plot these vectors with a quiver or line plot (i.e., not just points, but lines/arrows to the points).

[0.5] b) *Computing the identity decoder.* Use LIF neurons with the same properties as in question 1.3. When computing the decoders, take into account noise with $\sigma = 0.2 \max(A)$. Plot the decoders in the same way you plotted the encoders.

📌 The decoder will be a matrix $D \in \mathbb{R}^{2\times 100}$. Each column in D corresponds to a 2D vector.

📌 In the scalar case, to sample the input space, you used x values between -1 and 1, with $\Delta x = 0.05$. In this case, you can regularly tile the 2D $\mathbf{x}$ values $([1,1],[1,0.95],...[-1,-0.95],[-1,1])$. Alternatively, you can just randomly choose 1600 different $\mathbf{x}$ values to sample.

[0.5] c) *Discussion.* How do these decoding vectors compare to the encoding vectors?

[1] d) *Testing the decoder.* Generate 20 random $\mathbf{x}$ values throughout the unit circle (i.e., with different directions and radiuses). For each $\mathbf{x}$ value, determine the neural activity a_i for each of the 100 neurons. Now decode these values (i.e. compute $\hat{x} = D\mathbf{a}$) using the decoders from part b). Plot the original and decoded values on the same graph in different colours, and compute the RMSE.

📌 Technically, the RMSE is not well-defined for non-scalar data. The usual convention is to "flatten" the series, i.e., treat the extra dimensions as scalar datapoints. For n d-dimensional datapoints

$$RMSE(\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_n; \mathbf{x}_1, \ldots, \mathbf{x}_n) = \sqrt{\frac{1}{nd}\sum_{i=1}^{n}\sum_{j=1}^{d}\left(\hat{x}_{ij} - x_{ij}\right)^2}.$$

🐍 Per default, the `np.mean` function computes the mean on a "flattened" array.

[1] e) *Using encoders as decoders.* Repeat part d) 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 and compute the RMSE. In addition, recompute the RMSE in both cases, but ignore the magnitude of the decoded vectors by normalizing before computing the RMSE.

[1] f) *Discussion.* When computing the RMSE on the normalized vectors, using the encoders as decoders should result in a larger, yet still surprisingly small error. Thinking about random unit vectors in high dimensional spaces, why is this the case? What are the relative merits of these two approaches to decoding?