PROBLEM SET 2

Neural coding and adaptation in linear-nonlinear models

Contents

1 Neural coding in retinal ganglion cells

1.1 Background

Sensory systems do a remarkable job of extracting behaviorally relevant information from the environment and communicating that information to other brain regions. The field of neural coding explores this by asking: What information do neurons encode or represent? and How do biophysical mechanisms contribute to that encoding?

We can get a handle on these questions by formulating a quantitative model that characterizes the relationship between the external environment and neural activity. These *encoding models* allow us to distill our understanding of a sensory system, highlight key computational features, tackle normative questions (e.g. optimality), and uncover biophysical limits that have shaped the evolution of neural circuits.

For this problem, you will be analyzing data recorded from a salamander retinal ganglion cell (RGC) in response to a white noise visual stimulus. You will formulate a particular encoding model known as the linear-nonlinear (LN) model [2] that predicts the response of the cell to a given stimulus, and write code to learn the parameters of this model. You will also perform an analysis known as spike-triggered covariance (STC) analysis [6], to further characterize features that this particular ganglion cell is sensitive to.

For starters, download the required files (RGC data and the Matlab template script) from the course website. All of the code you need to write will be in the `rgc_analysis.m` script.

1.2 Experiment details

The data you will be exploring is stored in an `hdf5` file called `rgc_data.h5` (see the template script for how to load this data into Matlab). The data consists of a 16.67 minute recording of an OFF ganglion cell from the salamander retina. The stimulus was flickering white noise bars, sampled at a frame rate of 100Hz. The `stimulus` array has dimensions (30x100000) corresponding to the pixel value of the 30 bars over 100000 frames. The `time` array contains the time of the stimulus presentation for each stimulus frame. Finally, the `spike_times` array contains the spike times of an isolated retinal ganglion cell (RGC) recorded in response to the stimulus.

1.3 Spike-triggered analysis

To analyze this cell, you will first need to compute the *spike-triggered ensemble* (STE). This is a matrix containing the stimulus that directly preceeded a particular spike, for every recorded spike. Think of the STE as a cloud of points in the high-dimensional stimulus space. As we discussed in class, the mean of this set of points is known as the *spike-triggered average*. We will also characterize this point cloud by its *covariance*, this leads us to spike-triggered covariance (STC) analysis (see [6] for more information).

Part 1 Spike-triggered analysis The `rgc_analysis.m` script will loop over the set of spike times, and for each one, extract the stimulus that occurred right before that spike and store it in a matrix.

1. What is the dimensionality of the spatiotemporal filter? (This is the product of the number of spatial dimensions and the number of temporal samples in your filter).

2. First, you need to initialize the matrix that will store the spike-triggered ensemble (the variable `ste` in the script).

3. Then, fill out the code in the for loop that loops over the set of spike times, and for each spike, store the stimulus preceeding that spike in the STE.

4. Once you have the STE, you can compute its mean and covariance (after the for loop) to estimate the spike-triggered average and covariance, respectively. Remember to make sure that the dimensions of the covariance matrix are correct–it should be a square matrix where one side has length given by the dimensionality of the filter (not the number of spikes).

5. We will further break down the STC matrix by computing its eigendecomposition. The provided visualization code will generate an image of the STA, the eigenspectrum, and the STC eigenvectors. Remember, each eigenvector of the STC matrix is a spatiotemporal feature that has been unrolled as a vector. Add appropriate labels to each of these plots.

For this part, turn in plots of the spike-triggered average, the eigenvalues of the spike-triggered covariance matrix, and the top three eigenvectors (reshaped to look like a spatiotemporal filter). This plotting code has been written for you, but make sure to add appropriate axes labels and titles to the figures. In addition, answer the following questions:

1. Describe what the spike-triggered average looks like. What does this tell you about what this ganglion cell encodes?

2. How many eigenvalues in the eigenvalue spectrum are significant, i.e. above the noise floor? (you can estimate this by simple visual inspection of the eigenvalue spectrum, but a better approach would be to quantitatively estimate the noise distribution of eigenvalues by shuffling the data and repeating the procedure).

3. Estimate the dimensionality of the subspace of stimuli that this cell is sensitive to.

4. Describe what the eigenvectors look like. How do they compare to the STA?

5. Computing the STC requires a lot of data. How can we be sure that we have computed enough to accurately estimate the STC eigenspectrum? Can you come up with a simple way to test if we need more data (without recording more data)?

1.4 Linear-nonlinear (LN) models

In the same analysis script, you will also estimate a nonlinearity–a function that describes the threshold and amount of amplification necessary to best predict the ganglion cell response given the stimulus and the STA. To do this, you will need to do the following (again, all of this follows the template in `rgc_analysis.m`).

Part 2 LN modeling

1. First, we need to compute the linear projection (or dot product) of the stimulus onto the linear filter (STA) that we computed earlier. In the loop over time, compute the projection of each stimulus slice onto the STA and store it in the variable `u`.

2. Now, you need to bin the spike times into an array that stores the number of spikes observed at a particular time, which we will call `spike_counts`. Use Matlab's `hist` command to bin spike times using the bins given by the `time` variable.

3. Now we are ready to compute the nonlinearity of the LN model. Remember, the nonlinearity is the mean number of spikes (y-axis) vs. the projection of the stimulus onto the STA (x-axis). The code loops over discretized values of the projection (the variable `ub`, the discretization allows us to average out noise), and for each value of `ub`, you need to average the `spike_counts` array at times where the projection happens to lie within each particular bin.

For this part, turn in your plot of the estimated nonlinearity (again, the plotting code is provided–you need to add axis labels and a title). The code computes the nonlinearity using a different method, the ratio of histograms, your nonlinearity should match that one. In addition, answer the following:

1. What shape does the nonlinearity have? What does this imply about how the neuron responds to multiple inputs (e.g. the combination two flashes as opposed to an individual flash)?

2. Estimate (just by eye) a threshold of the nonlinearity.

3. For this stimulus, what fraction of the time is this neuron above threshold?

4. What is the dimensionality of the subspace of stimuli that a linear-nonlinear model is sensitive to?

5. To fit this linear-nonlinear model, we used a stimulus with zero mean and fixed contrast. Natural stimuli, on the other hand, have many changes in mean luminance and contrast. What does this mean for our LN model? How can the LN model deal with such stimuli?

2 Feature selectivity of single neuron models

Sensory neurons encode features of the stimulus. One basic aspect of this is the temporal pattern of the input. By virtue of their intrinsic filtering properties, different neurons may prefer different temporal patterns. Using the single compartmental models you created last week we will investigate the temporal features and the nonlinear amplification (nonlinearity) preferred by different types of neurons.

2.1 The spike-triggered average of the integrate-and-fire neuron

As a warm-up, we will first compute the spike-triggered average of the leaky-integrate-and-fire (LIF) neuron. That is, we will inject a random current into a simulated neuron, and compute the average current that preceeds a spike. This was computed analytically in [5], you can use the figures in that paper as a guide for what to expect. You can use the script `sta_iaf.m` as a template.

Part 3 The STA of the LIF neuron

1. You only need to make two modifications to the template–choose a value for the standard deviation of the injected current (σ), and create the current array that we will use to stimulate the integrate and fire neuron.

2. The rest of the code simulates the integrate-and-fire neuron (just like last week) and computes the STA by averaging the current preceeding every spike. Add labels to the generated plot of the STA and include it in your report.

3. Is the filter you observe monophasic or biphasic? What does this mean about how this cell responds to current input?

4. Vary the standard deviation (strength) of the current. How does this affect the filter? Plot multiple STAs on the same figure, one for each value of the standard deviation that you choose.

5. How does the STA change as a function of the current strength?

2.2 Adaptive gain scaling in the hodgkin-huxley model (from Lecture 4, on 4/12)

Next, we will investigate whether or not known single neuron mechanisms are sufficient to give rise to adaptive changes in gain that occur automatically when the input changes. To do this, we will use the Hodgkin-Huxley (HH) simulation. For treatments on the spike-triggered average of the HH neuron, see [3, 1]. The simulation you will be running was published by Mease et. al. [4] (you should read that paper so that you know what to expect).

Again, the basic goal is we would like to understand what single neuron mechanisms could give rise to gain scaling. Specifically, we will simulate a Hodgkin-Huxley neuron in response to different current fluctuations (by varying the standard deviation of the injected current, just as we did in the integrate-and-fire neuron). We will then compute linear filters and nonlinearities (so, LN models) under these different conditions to see how they change.

You can use the provided `sta_hh.m` template as a guide.

Part 4 Gain scaling in the Hodgkin-Huxley neuron

1. The template provided simulates a Hodgkin-Huxley neuron, and fits a linear-nonlinear model to the response of the simulated neuron. Choose a length of simulation (`T`) and injected noise strength (standard deviation, `I_sigma`) to use. The longer the simulation, the more time it will take to compute. Start small and increase the length depending on how long you want to wait–something in the range of 1000-100000 seconds should be good.

2. The code uses a loop to compute the spike-triggered average of the Hodgkin-Huxley neuron. For the integrate and fire neuron, spike times were well defined (by when the neuron crossed threshold). For the Hodgkin-Huxley neuron, you will need to identify spike times directly from the voltage trace

(by identifying peaks in the voltage). To do this, use the provided `peakdet` function. The `peakdet` function identifies peaks in a provided signal that are larger than some delta, which is a parameter you must provide to the function. Decide on a value for the delta parameter that will robustly identify spikes. Verify that the spike detection is working properly.

3. The provided code will compute and plot the STA and nonlinearity (parts of the LN model). What does the STA of the Hodgkin-Huxley model look like? Is it monophasic or biphasic?

4. Describe the qualitative shape of the nonlinearity.

5. Now, compute LN models again but this time in response to current injection with different standard deviation. Did the linear filter change? What about the nonlinearity?

6. Using the LN models across different contrasts as a guide, do you see evidence for gain scaling as a possible feature of the HH equations?

7. Turn in plots of any LN models that you fit (filters and nonlinearities).

8. Discuss the advantages and disadvantages of having individual neurons perform their own gain scaling (as opposed to it being a network phenomenon, which is how we implemented it last week).

References

[1] Blaise Agüera y Arcas and Adrienne L Fairhall. What causes a neuron to spike? *Neural Computation*, 15(8):1789–1807, 2003.

[2] EJ Chichilnisky. A simple white noise analysis of neuronal light responses. *Network: Computation in Neural Systems*, 12(2):199–213, 2001.

[3] Adrienne L Fairhall, William Bialek, et al. Computation in a single neuron: Hodgkin and huxley revisited. *Neural Computation*, 15(8):1715–1749, 2003.

[4] Rebecca A Mease, Michael Famulare, Julijana Gjorgjieva, William J Moody, and Adrienne L Fairhall. Emergence of adaptive computation by single neurons in the developing cortex. *The Journal of Neuroscience*, 33(30):12154–12170, 2013.

[5] Liam Paninski. The spike-triggered average of the integrate-and-fire cell driven by gaussian white noise. *Neural computation*, 18(11):2592–2616, 2006.

[6] Odelia Schwartz, Jonathan W Pillow, Nicole C Rust, and Eero P Simoncelli. Spike-triggered neural characterization. *Journal of Vision*, 6(4):13–13, 2006.