Neuro 120 Homework 2: Data Analysis
Due: Thursday 1 March 2018 at 5 pm

In this homework you will analyze spiking responses from an electrophysiology dataset, investigate overfitting dynamics in linear regression, and solve the cocktail party problem using independent component analysis. Starter code and data for the assignment is provided on Github. 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. **Auditory neuroplasticity.** Here you will investigate data collected by one of your TFs in the lab of Christoph Schreiner at UCSF for an experiment that probed the impact of early experience on responses in auditory cortex. Seemingly simple manipulations of a young animal's environment can dramatically reshape its neural responses well into adulthood. For example, playing a repetitive pure tone at a single frequency to rat pups can cause their primary auditory cortex to become $\sim 50\%$ larger, with a huge fraction of neurons selective to the exposure frequency (see e.g., de Villers-Sidani et al. *Critical period window for spectral tuning defined in the primary auditory cortex (A1) in the rat.* Journal of Neuroscience, 2007.). This neuroplasticity caused a stir when it was first discovered both because it speaks to ideas about nature and nurture, and because it has therapeutic applications in, for example, restoring hearing through cochlear inplants (restoring signals from the peripheral auditory nerve may not be enough if cortical processing has already adapted to the lack of input).

   The data you will examine is from the following experimental setup: litters of rat pups were reared in a home cage with a speaker above it. The speaker continuously played a specific sound sequence as the young rats were growing up. You can hear the exposure stimulus by playing `exposure_stimulus.wav`. You will hear that the stimulus consists of a repetitive chord-like stimulus. In particular, the chord is pulsed at 6Hz. After this sound exposure period, spiking responses were recorded in primary auditory cortex using silicon electrode probes. Here you will analyze responses to two different types of stimuli: a 1 minute segment of the sound sequence they were reared with; and a 15 minute 'dynamic moving ripple' (DMR) stimulus which can be used to generate spike triggered averages (STAs) and linear receptive field estimates. You can hear this DMR stimulus in `DMR_stimulus.wav`. It contains a range of different spectral and temporal modulations. (Several other stimuli were also played, but you won't investigate those here–they could be the target of a final project, however).

   First, we will analyze basic spike trains. In `exposure_stimulus_investigation.m` you will find starter code that loads a set of variables into the workspace from the file `exposure_stimulus_experiment.mat`. The vector `spikes_single_unit` contains the spike times (in seconds) for a specific neuron from an animal in the experimental group. The vectors `spikes_control` and `spikes_exp` contain all spike times recorded from a control and an experimental animal respectively. The total number of neurons from which we recorded is in the variable `N_control` and `N_exp` for the control and experimental animal respectively.

   (a) Make a raster plot of the response of the single unit to the exposure stimulus. In particular, treat each 1/6 of a second chord pulse as a separate stimulus presentation. Your raster plot will have an x-axis corresponding to time (0 to 1/6s), and a y-axis corresponding to stimulus number (1 to 360). Does this neuron appear to respond to the stimulus?

   (b) To further investigate this, plot the average firing rate for this neuron in response to the stimuli by smoothing with a Gaussian kernel. That is, for each of the 360 stimulus presentations, estimate the firing rate by summing up Gaussian distributions centered on each spike. Then compute the average of these curves over stimuli. Use a Gaussian of standard deviation $\sigma = 5ms$.

   (c) Generate plots for $\sigma = 50ms$ and $\sigma = 0.5ms$. What is the impact of very large or very small standard deviations?

   (d) So far we have examined a single neuron. Now let's see if these trends hold more generally, and investigate how neural responses might have changed between the control and experimental groups. Compute and plot a grand average poststimulus time histogram for all spikes and all recorded neurons, for both the control and experimental groups. That is, count the average

number of spikes falling into a set of time bins (here the average is over the stimuli and the number of neurons from which we have responses), divided by the bin width to yield a firing rate in Hz. Use a bin width of 5ms.

(e) What differences do you observe between the experimental and control groups? Did neurons become more or less selective to the exposure stimulus? Did the precision of responses change? Are there any differences in responses after the initial peak response?

Now we will generate STA receptive field estimates for a specific neuron, using responses to the 15 minute DMR stimulus. The file `sta_estimation.m` contains simple starter code. It loads the data file `dmr_experiment.mat`, which contains the following variables: `spikes`, a list of spiking times in seconds for this neuron; `stim_spectrogram`, a 2D spectrogram of the DMR stimulus (here the rows index different frequencies and the columns index different time points); `stim_freq`, a vector with the frequencies of each row in the spectrogram in Hz; and `stim_time`, a vector with the time of each column in the spectrogram in seconds. When you run the first section of `sta_estimation.m`, it will plot a spectrogram of the DMR stimulus (you will have to zoom in, particularly in the time dimension, to see much structure).

(f) Generate a spike-triggered average for this neuron that extends 125 ms into the past and 125ms into the future. That is, for each spike, cut out a segment of the stimulus corresponding to the 250ms window centered on the spike, and average these together. The portion of the STA that is 125ms into the future is a useful sanity check: the stimulus in the future cannot impact the spiking behavior of the neuron, so we expect to see an STA that is approximately zero for these future times.

(g) What frequency is the neuron most selective to?

(h) According to the STA, would the neuron's peak response be higher to a constant tone at its preferred frequency, or to a brief tone pip? Approximately how long should a tone pip be to evoke the largest response?

(i) The STA is sensitive to stimulus correlations, and is only the optimal linear filter if the stimulus correlations are white (the identity matrix). Plot the stimulus correlation matrix (`stim_spectrogram*stim_spectrogram'`). Is it close to an identity matrix?

2. **Random neural networks and overfitting in regression.** Here we will investigate how the complexity of a model impacts its generalization performance, using linear regression. The starter code in `nn_regression.m` instantiates a setting in which a simple neural network is attempting to learn a sinusoidal function of its input. The initial part of the code contains parameters controlling the problem setting. The code generates N training inputs in the vector x that are equally spaced between $-\pi$ and π (this could represent the orientation of a visual stimulus, say). The target output is a sinusoidal function plus noise, $y = \sin(x) + \epsilon\eta$, where $\eta \sim \mathcal{N}(0, I)$ is unit Gaussian noise and the parameter ϵ controls the strength of this noise (it is initially set to zero).

   In lecture, we saw how regression from different order polynomials of the input could lead to under- or overfitting behavior. Instead of polynomials, here we will use a different model class: a nonlinear neural network with randomly drawn synaptic connections. This network has N_h neurons in it, with Gaussian weights J and rectified linear (ReLU) nonlinearities. In response to an input $[x; \mathbf{1}] \in \mathbb{R}^{2\times N}$, the network generates the activity $H = \max(0, J[x; \mathbf{1}])$ where $J \in \mathbb{R}^{N_h \times 2}$ is a matrix of synaptic connections drawn randomly from a Gaussian distribution. In essence, the network receives a low dimensional input x, and produces a potentially much higher dimensional representation in the neural activity H. This type of random nonlinear expansion is thought to occur in several brain structures, including the rodent olfactory bulb and the fly mushroom body among others (see Babadi & Sompolinsky. *Sparseness and Expansion in Sensory Representations.* Neuron, 2014). Finally, we will use linear regression to predict y using the activity in the network H.

(a) Implement linear regression, with regularization. We want $y \approx wH$ for some weight vector w. From the normal equations you've seen in class, the optimal weight vector is

$$w^* = yH^T(HH^T + \lambda I)^\dagger.$$

Here we have included the regularization parameter λ, and also have used the pseudoinverse $\dagger$ (`pinv()` in matlab), rather than the inverse, to allow exploration of the setting where we have fewer training examples than neurons in the network. Plot several example predictions for $N_h = 2, 10, 26$.

(b) Train networks of different sizes in the range $N_h = [1\ 40]$. This is the *overdetermined* regime where you have more training samples than parameters in your model. Plot the average mean squared test error as a function of N_h. You'll want to average over many repetitions at each size (≥ 500 repetitions). What would you say is the approximate best size model? How do things go wrong when the model is too small or too big?

(c) Now throw caution to the wind and train some massive networks in the range $n_h = [40\ 500]$. This is the *underdetermined* regime, where you have fewer training samples than parameters in your model. In underdetermined models, there is an infinite space of solutions that will fit the training points exactly. By using the pseudoinverse in the linear regression equation, we are picking the solution with smallest norm. Plot the average mean squared test error as a function of N_h. What would you say is the approximate best model size? How variable is the performance of the really big networks compared to smaller networks?

(d) Investigate a model with $N_h = 500$, but set the label noise $\epsilon = 0.2$. What is the typical MSE with and without label noise? Do the predictions with label noise look good? Now adjust the regularization parameter $\lambda \approx 1e^{-4}$. How does this change the MSE and the predictions with and without noise?

(e) Based on your results, does it make sense for neural networks to massively expand the dimension of their inputs? What regime (in terms of label noise and number of neurons relative to amount of training data) would you argue is the relevant regime for the learning problems faced by brain?

3 Image demixing: visual cocktail party problem.

In this problem, you will use Independent Components Analysis to separate a set of mixed images into individual components of the mixture. A first step toward this, to familiarize yourself with the format of the data, you will write functions that remove the mean, and calculate the covariance of a given data set. You will then diagonalize the covariance to determine principle components of the data, and use these to whiten it. ICA can then be performed on this whitened version of the data.

Throughout the problem, we will use the convention that individual data points will be placed in the rows of matrics, with their length running along the row. So, a dataset of N points in D dimensions will be an $N \times D$ matrix.

a) **Covariance matrix and PCA** First, we want to determine the principle components of the data. For this problem, work with the toy data toWhiten.mat to check your work.

(i) Write an inline function that removes the mean from a data matrix. Check by checking the mean of the output.

(ii) Without using MATLAB's built-in covariance function, write a function that calculates the covariance matrix of the data. Check against the built-in function. *Tips:* 1) This can be done with an inline function as well. 2) Remember that you need to divide by $n - 1$ to bias correct the covariance estimates.

(iii) Use MATLAB's built-in *eig* function to determine the eigenvalues and eigenvectors of the covariance matrix. Overlay the principle components of the data on a scatter plot of the data.

(iv) Look at the dimension of the covariance matrix. Comment on conditions (values of D and N) for which the diagonalization will be time consuming. Compare the time required for

singular value decomposition (MATLAB's *svd* function) with the time required for calculation and diagonalization of the covariance matrix by running both on random 1000×5000 and 5000×1000 matrices.

b) **Whitening the data.** Whitening of data is transformation of the data by linear functions in order to set the mean of the data to zero, and the covariance matrix of the data to the identity matrix. Use the toy toWhiten.mat data again to help write a function that whitens input data. It should: 1) Remove the mean of the data. 2) Project the data into the basis of principle components. 3) Scale each of the principle components by $1/\sqrt{\lambda}$ where λ is the eigenvalue corresponding to that principle component. Check by plotting a scatter plot of the toWhiten data after whitening. Given how you constructed your whitened data, how can you extract the first principle components from the whitened data? (don't over think it!)

c) **ICA:** Given 3 samples of mixtures of 3 images, we want to find the component images. ICA does this by applying a particular linear transformation to the whitened data. Import the mixed images from the file *mixedImg.mat*. Using the provided *plotImgs* function to plot the images at each stage,

(i) Plot the mixed images.

(ii) Whiten the images using you function from part (a). Plot the whitened images.

(iii) The function *learnWeights* will learn a transformation (applied to the right of the data matrix in this case) that maximize the kurtosis of the extracted components. Natural images have non-Gaussian pixel statistics, so this transformation will find images that are as far from the mixture as possible. Run this function on the whitened images, apply the transformation to the whitened images, and plot the resulting output. Whitening prior to learning makes learning easier; try learning without whitening first to see what happens.