PROBLEM SET 2

METHODS IN COMPUTATIONAL NEUROSCIENCE 2013

1. Predicting where a monkey will reach

Data from a monkey performing a center-out task were kindly provided by Nicho Hatsopolous. The problem was adapted from an assignment for his Computational Neuroscience course at University of Chicago. You will construct a decoder to predict which direction the monkey reaches.

1.1. **The Data.** Nicho's description of the monkey's task and the data follows:

These behavioral data were collected using a manipulandum, which is an exoskeleton that fits over the arm and constrains movement to a 2-D plane. Think of the manipulandum as a joystick controlled with the whole arm. The behavioral task was the center-out paradigm pioneered by Georgopoulos and colleagues (1982). The subject first holds the cursor over the center target for 500 ms. Then a peripheral target appears at one of eight locations arranged in a circle around the center target. In our task there is an instructed delay, which means that after the peripheral target appears the subject must wait 1000-1500 ms for a go cue. After the go cue, the subject moves to and holds on the peripheral target for 500 ms, and the trial is completed.

`Lab5_CenterOutTrain.mat` is a file that contains neural spike times and events associated with a center-out task (to one of 8 directions). Load this. You will analyze the data in the struct `unit`. This contains the spike times for 143 neurons and also indicates whether the recorded neurons are from M1 or PMd. You also have information about 158 trials, including the instruction times, the go cue times, and the per trial direction of the target. The targets are labeled 1 through 8, starting from 0°, then 45°, on to 315°. `go(k)` and `instruction(k)` are the times when the go and instruction cues occur on the k^{th} trial in units of seconds. `unit(n).times` lists all the times when the n^{th} neuron spiked.

1.2. **Maximum Likelihood Decoder.** Define the response vector $\mathbf{r}$ to be the spike counts of every recorded neuron on a particular trial (between the instruction time and the go time). We pose a neural decoding problem: identify the most likely movement direction d^*. By Bayes' Rule,

$$\Pr[d|\mathbf{r}] = \Pr[\mathbf{r}|d]\Pr[d]/\Pr[\mathbf{r}].$$

Let's make several simplifying assumptions. The prior probability of each movement direction is uniform $\Pr[d = i] = 1/8$ for each of 8 directions i, so it is irrelevant. Additionally, the marginal probability $\Pr[\mathbf{r}] = \sum_{i=1}^{8} \Pr[\mathbf{r}, d = i]$ is independent of d, so we can leave it out from our optimization. Finally, we make the drastic but highly convenient assumption that neurons are independent, given the intended movement direction. (If you can eschew these assumptions in part 4 with an improved algorithm, please feel free!) That is,

$$\begin{aligned}\Pr[\mathbf{r}|d] &= \Pr[r_1|d]\Pr[r_2|d]\dots\Pr[r_N|d]\\ &= \prod_{i=1}^{N}\Pr[r_i|d].\end{aligned}$$

So we have

$$d^* = \arg\max_d \Pr[d|\mathbf{r}] = \arg\max_d \prod_{i=1}^{N}\Pr[r_i|d].$$

Since log is a monotonic function, it preserves the locations of maxima. It also turns products into sums, reducing the difficulty of many optimizations, and keeps products of small numbers from being rounded down to 0 by the floating point representation of the computer. We should therefore instead find

$$\begin{aligned}d^* = \arg\max_d \log\Pr[d|\mathbf{r}] &= \arg\max_d \log\left(\prod_{i=1}^{N}\Pr[r_i|d]\right)\\ &= \arg\max_d \sum_{i=1}^{N}\log\Pr[r_i|d].\end{aligned}$$

1.3. **What You Need To Do.**

(1) Visualize the data by making raster plots for the different conditions and histograms of firing rates. You can be creative about how exactly you visualize the data – we just want you to be familiar with what the raw data look like.

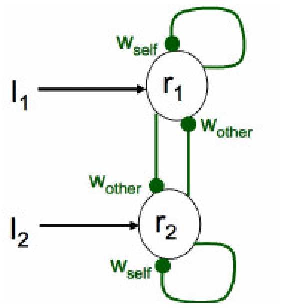
(2) Decide on some simple parametric probability distribution $\Pr[r_i|d, \Theta_{id}]$ (with parameters Θ_{id}) to represent the probability of a spike count or firing rate of the i-th neuron given intended movement direction d. Use the training data to estimate the parameters of these distributions.

(3) Calculate d^* for each trial, and compare this to the actual direction. Validate by holding out 10% of the data as a validation set that you don't use to train your decoder, and calculate percent correct on the validation set.

(4) Develop an improved algorithm, inspired by the lectures and your own creativity. Let one of the TAs know when your algorithm is ready to test on a separate batch of test data.

2. Model circuit for selective amplification, neural integration, perceptual rivalry, or many other phenomena

Many neural networks, including the oculomotor neural integrator, consist of 2 populations of neurons whose effective connectivity is such that they provide self-excitation to neurons in their own population and inhibit neurons in the opposite population. A simple case of these circuits occurs when the connections are symmetric as in the following diagram:



Here, w_{self} gives the connection between either neuron population and itself and w_{other} gives the connection between the different neuron populations. In this problem, we will quite generally find the eigenvalues and eigenvectors for a circuit with this connectivity, and then apply our findings to several relevant cases.

a. Write down the equations describing this network, assuming the 2 neurons each decay to zero with time constant τ in the absence of input. Write them both as separate equations for dr_1/dt and dr_2/dt and as a single equation written compactly in terms of the firing rate vector $\mathbf{r} = [r_1, r_2]$ and a recurrent connectivity matrix $\mathbf{W}$.

b. Recalling that the eigenvectors of $\mathbf{W}$ are the patterns of inputs with the property that the outputs of the network will be a scaled version of this same pattern, and noting the symmetry of the network, try to guess the eigenvectors $\xi^{(1)}$ and $\xi^{(2)}$ of $\mathbf{W}$. (Use the convention that the first element of each eigenvector equals +1.) Confirm that you have found the correct direction by showing that $\mathbf{W}\xi = \lambda\xi$ for each eigenvector and give a formula for the corresponding eigenvalues λ_1 and λ_2.

c. Decompose the input vector $\mathbf{I}$ into a combination of the two eigenvectors $\mathbf{I} = b_1\xi^{(1)} + b_2\xi^{(2)}$ you found in part (b). Writing out this equation for each component, you should find that the coefficients b_1 and b_2, respectively, correspond to a part of the input that is common to both cells, $\mathbf{I}_{\text{common}}$, and a part $\mathbf{I}_{\text{diff}}$ that gives the difference of each cells' input from this common value, i.e. that:

$$\mathbf{I}_1 = \mathbf{I}_{\text{common}} + \mathbf{I}_{\text{diff}}$$
$$\mathbf{I}_2 = \mathbf{I}_{\text{common}} - \mathbf{I}_{\text{diff}}$$

This implies that the common and difference portions of the input behave independently.

d. *Simple model of selective amplification of differences between inputs.* Suppose that each neuron excites itself ($w_{\text{self}} > 0$) and inhibits the other

($w_{\text{other}} < 0$) and that the inhibitory connections are stronger than the self-excitatory connections (i.e. $|w_{\text{other}}| > w_{\text{self}}$).

(1) What will happen to inputs that are common to the two cells? (Will they be amplified or attenuated?) Show this by looking at the eigenvalue for the appropriate eigenvector.
(2) What will happen to inputs that are opposite for the two cells?
(3) Calculate the eigenvalues for $w_{\text{self}} = 0.2$ and $w_{\text{other}} = -0.7$.
(4) Calculate the amplification factors $1/(1-\lambda)$ for each eigenvector, using the values in part (3) above.
(5) If each cell in the network has an intrinsic time constant $\tau = 18$ ms, what will be the corresponding time constants τ_{eff} for each component? From this answer, do you expect the amplified component to change more or less quickly than the attenuated component?

e. Consider a network with $w_{\text{self}} = 0.2$ and $w_{\text{other}} = -0.7$, as in part (d). Suppose $\mathbf{I}_1 = 63$ Hz and $\mathbf{I}_2 = 57$ Hz.
(1) Write these inputs in the form $\mathbf{I} = b_1\xi^{(1)} + b_2\xi^{(2)}$. What are b_1 and b_2?
(2) Recall that the steady-state firing rate is obtained by simply scaling each component (b_1 or b_2) of $\mathbf{I}$ by the corresponding amplification factor ($(1-\lambda_1)^{-1}$ or $(1-\lambda_2)^{-1}$, respectively), and then adding together these components, i.e.

$$\mathbf{r} = \frac{b_1}{1-\lambda_1}\xi^{(1)} + \frac{b_2}{1-\lambda_2}\xi^{(2)}$$

What will the steady-state firing rate vector $\mathbf{r}_\infty = [r_{\infty,1}, r_{\infty,2}]$ equal? For various initial conditions, sketch the expected trajectory of the firing rates as they approach this steady-state value.

f. Simulate the network by numerically solving the equations; the point is to "blindly" simulate these equations and confirm that you get the same answer you found analytically. For initial conditions, set $\mathbf{r}(t = 0) = \mathbf{I}$, i.e. start out the network at the firing rate values that would have been obtained in steady-state if there were no recurrent connections.

Plot $\mathbf{r}_1$ and $\mathbf{r}_2$ as a function of time for long enough to see them reach steady state. On the same set of axes, plot the eigenvectors $\xi^{(1)}$ and $\xi^{(2)}$. Check that the steady state values that you get are the same as those you calculated analytically.

g. Modify the above network so that it integrates a quantity proportional to the difference between $\mathbf{I}_1$ and $\mathbf{I}_2$. What is the condition on w_{self} and w_{other} for this to occur? Also observe what happens when the inputs $\mathbf{I}_1$ and $\mathbf{I}_2$ equal zero. Do you observe persistent neural activity? (For comparison, set $\mathbf{I}_1$ and $\mathbf{I}_2$ equal to zero in part (f)).For this question, dont worry that firing rates can become negative. You can consider $\mathbf{r}$ to be the firing rate relative to a background level.

h. *Winner-take-all networks.* For a more extreme example of the rivalry between inputs observed in part (f), see what happens if w_{other} is a large negative value. For this network, add constraints on your neurons such that firing rates that become negative get thresholded to zero. Experiment with various initial conditions and input values.

(i) Such a network is called a *winner-take-all network* because the loser becomes zero and the winner achieves a large value determined by its external input $\mathbf{I}$ and by w_{self}. What is the condition for one of the neurons to be driven to zero firing rate? What additional condition (on the synaptic weights) will lead the winner to exhibit runaway growth? (If you wish to explore this part of the model's parameter space, you can prevent runaway growth from occurring by adding saturation to the model. This is most simply done by imposing a maximum firing rate constraint such that neurons do not increase their firing rates beyond a fixed level $r_{\max}$).

(ii) This model is commonly used as a simple description of phenomena such as perceptual rivalry. A fundamental property of perceptual rivalry is that the percept remains stable for awhile and then stochastically switches to the other possibility. Can you find a way to modify the previous model to reproduce this behavior?

3. Spike train analysis

Here we will explore some methods for spike train analysis. To study the effectiveness of the methods we will generate the stimulus and spikes ourselves and compare the results of the analysis to the ground truth. For a review see reference 3 below.

(1) Generate a spatiotemporal white noise stimulus $s(x,t)$, with x ranging between -0.5 and 0.5 and t ranging between 0 and a T of your choosing. Discretize x into $p = 25$ bins so that your full stimulus array is $p \times T$ in size. The p dimensional stimulus vector presented at time t will be denoted as $\vec{s}(t)$.

(2) Build a one dimensional LNP model. The neuron's spatial linear filter will be defined by the "edge" kernel

$$f(x) = ax^3 e^{-x^2/b}$$

with $a = 20$ and $b = 0.02$. f can also be thought of as a vector in the p dimensional vector space, so it will be denoted as $\vec{f}$. The firing rate at time t given the stimulus is then

$$r\left(\vec{s}(t)\right) = r_0 \left(1 + \exp\left[\left(\theta - \vec{s}\,(t) \cdot \vec{f}\right)/\Delta\right]\right)^{-1}$$

with $r_0 = 1, \theta = 0.025$ and $\Delta = 0.05$.

(3) STA
   (a) Compute the Spike Triggered Average (STA) on a simulated spike train. How close is it to $\vec{f}$? Recall that the spike triggered average is the average over stimuli that preceded a spike. Here the spiking is instantaneous so there is no lag between stimulus presentation and spiking.
   (b) From the distribution of all stimuli and spike triggered stimuli projected on $\vec{f}$, compute the nonlinear part of the LNP model. Hint: use Bayes' theorem to fit $p(\text{spike}|\vec{s}(t) \cdot \hat{f})$ in terms of a ratio of two distributions you have access to ($\hat{f} = \text{STA}$).
   (c) Generate a test stimulus and simulate a number of responses using the STA, the nonlinearity you found and a poisson random number generator. Compare these responses to spike trains that were generated using the model described above.

(4) STC
   (a) First we will consider a one dimensional model with a quadratic nonlinearity:

$$r\left(\vec{s}(t)\right) = r_0 \left(1 + \exp\left[\left(\theta - \left(\vec{s}\,(t) \cdot \vec{f}\right)^2\right)/\Delta\right]\right)^{-1}$$

   with $\theta = \Delta = 0.01, r_0 = 1, a = 20$.
      (i) Simulate a spike train and compute the STA. Are you able to recover $\vec{f}$? If not explain why using the distribution of all and spike triggered stimuli projected on $\vec{f}$.
      (ii) Compute the STC and discuss its success in recovering the model used to generate the spikes. Recall the spike triggered covariance

method:

$$\begin{aligned}
C^{ij}_{\text{stc}} &= \frac{1}{N_{\text{spike}}-1}\sum_{k=1}^{N_{\text{spike}}}\left(s_i^{t^k_{\text{spike}}}-s_i^{\text{sta}}\right)\left(s_j^{t^k_{\text{spike}}}-s_j^{\text{sta}}\right)\\
C^{ij}_{\text{prior}} &= \frac{1}{N-1}\sum_{t=1}^{N}\left(s_i^t-\bar{s}_i\right)\left(s_j^t-\bar{s}_j\right)\\
\Delta C &= C_{\text{stc}}-C_{\text{prior}}
\end{aligned}$$

(b) Now we turn to work with a two dimensional model neuron. The firing rate is

$$\begin{aligned}
r\left(\vec{s}(t)\right) &= r_0\left(1+\exp[(\vec{s}(t)\cdot\vec{f}-\theta_f)/\Delta_f]\right)^{-1}\times\\
&\quad\left(1+\exp[(\vec{s}(t)\cdot\vec{g}-\theta_g)/\Delta_g]\right)^{-1}
\end{aligned}$$

Where $\vec{f}$ is as before with $a=2$, $g(x)=ce^{-x^2/d}$ with $c=0.005$ and $d=0.02$ and $\Delta_f=0.005, \Delta_g=0.004, \theta_f=\theta_g=0$.

(i) Discuss the meaning of the parameters $\Delta_f, \Delta_g, \theta_f, \theta_g$ and sketch $r(\vec{s})$ in the space $\left(\vec{s}\cdot\vec{f}, \vec{s}\cdot\vec{g}\right)$.

(ii) Compute the STC. How many spikes are needed for a good reconstruction? How did you determine the significance of the eigenvalues?

(iii) Bonus - Compute the nonlinearity from the two dimensional distributions and make predictions for a test set.

(5) GLM

Recall the Generalized Linear Model with an exponential nonlinearity (for a single neuron and a 1 dimensional stimulus). The firing rate in response to a stimulus $\vec{s}$ obeys the equation:

$$\lambda(t)=\exp\left(\vec{k}\cdot\vec{s}(t)+\vec{h}\cdot\vec{y}(t)+\log\lambda_0\right)$$

where $\vec{y}(t)$ is the spike history vector and $\vec{h}$ corresponds to the post-spike history filter. $\vec{k}$ is the stimulus filters, and λ_0 is a baseline firing rate. Here we will work with $\vec{k}=\vec{f}$.

(a) Generate a white noise stimulus file.

(b) Define two response history filters. One should lead to a refractory period, and one to bursty firing.

(c) Simulate spike trains in response to the noise file you have created and save the data. Do the cells behave as intended? To study the short time dynamics of the cell you can plot the auto correlation function (`xcorr` in MATLAB).

(d) Fit a GLM to the simulated spike trains and compare your predefined filters to the ones you found. Below are some hints you may find useful to fit a GLM.

(i) n_s, n_h will be the dimensionality of the stimulus and response history filters, respectively.

(ii) Expand the dimensionality of the stimulus to be $n_s\times T$ (call that matrix `S`) and the dimensionality of the response to be $n_h\times T$ (call that matrix `R`).

(iii) Initialize a vector `a` of length $1+n_s+n_h$. Positions 1 through n_s will hold $\vec{k}$, the stimulus history filter. Positions n_s+1 through n_s+n_h will hold $\vec{h}$, the spike history filter, and position n_s+n_h+1 will hold the baseline firing rate.

(iv) Recall that $\log\lambda(t) = \log\lambda_0 + \vec{k}\cdot\vec{s}(t) + \vec{h}\cdot\vec{y}(t)$. Define the function handle `A = @(a,S,R)` which will hold $\lambda(t)$ for the duration of the simulation.

(v) Using `A` define the function handle `L` which will be the log likelihood function to be extremized.

(vi) Use the functions `fminunc` and `optimset` to calculate the GLM.

(6) Bonus - If a cell has a spatial *and* temporal filters (of dimensions n_t, n_x respectively), assuming you had repeated stimulus presentations, how could you find an approximation for the spatiotemporal receptive field without working in the $n_t \times n_x$ dimensional space but rather in a $n_t + n_x$ dimensional space?

4. Compressed Sensing

a) L_2 **minimization in one dimension**. Here we will explore why L_1, but not L_2, regularization sets the estimated signal exactly to zero sometimes. Consider a single scalar signal s. Suppose you get to take a noisy measurement of this signal $x = s + \epsilon$ where ϵ is a zero mean Gaussian noise variable with variance σ^2. Suppose you use a zero mean unit variance Gaussian prior on the unknown signal. Then your maximum a-posteriori (MAP) estimate $\hat{s}$ of the unknown signal s is $\hat{s} = \text{argmax}_s p(s|x)$. Show that $\hat{s}$ arises as the solution to the following optimization problem:

$$\hat{s}(x,\sigma^2) = \text{argmin}_s \left(\frac{1}{2}\frac{(s-x)^2}{\sigma^2} + \frac{1}{2}s^2 \right). \tag{1}$$

b) Solve the above optimization problem and find write down the function $\hat{s}(x,\sigma^2)$.

c) L_1 **minimization in one dimension.** Now lets suppose we think for some reason the true signal s has a finite probability of being 0 (i.e. the single scalar is sparse). So lets replace the L_2 regularization above with the L_1 regularization below to get a different estimate:

$$\hat{s}(x\,,\sigma^2) = \text{argmin}_s \left(\frac{1}{2}\frac{(s-x)^2}{\sigma^2} + |s| \right). \tag{2}$$

Prove that

$$\hat{s}(x\,,\sigma^2) = \text{sgn}(x)(|x| - \sigma^2)_+, \tag{3}$$

where $(y)_+ = y$ if $y > 0$ and 0 if $y \le 0$. Also plot $\hat{s}$ as a function of x to see what it looks like. You should see that L_1 regularization yields a *nonlinear* estimate (as a function of x) that is exactly 0 as long as your measurement x is below a noise dependent threshold. On the otherhand, L_2 regularization yields an estimate that is a *linear* function of the measurement x.

d) **Playing around with compressed sensing**. To get a sense of the performance of compressed sensing, reproduce the experimental results (blue points) in Fig. 8B of reference 2 below (don't worry about the theory - i.e. the red curves). As a starting point, build on the provided code in `CompSenseDemo.m`, which will allow you to compute the compressed sensing based reconstruction error for one value of the measurement density α and one value of the fraction of nonzeros f and one trial. You will see that as you reduce α for a given f you will find a phase transition from perfect reconstruction to error prone reconstruction. Comment intuitively on what types of errors you see as you reduce α.

e) **Fun in high dimensions**. Part of the reason compressed sensing and random projections work is that in high dimensions (whether in the space of natural signals or space of neural activity patterns), funny things can happen that defy our low dimensional intuition. To exercise your high dimensional thinking, consider the following puzzle: the volume of a solid sphere of radius r in dimension N is proportional to $\frac{r^N}{N!}$ (for even N). This implies that the volume of a sphere of any fixed radius goes to 0 as the dimension goes to infinity. Give an intuitive reason for this. (Hint: consider the volume of an N dimensional hypercube of side length 1).

References

[1] Ganguli S, Sompolinsky H (2012), Compressed sensing sparsity and dimensionality in neuronal information processing and data analysis *Ann Rev. Neurosci.*.

[2] Advani M, Lahiri S, and Ganguli S (2013), Statistical mechanics of complex neural systems and high dimensional data *Journal of Statistical Mechanics: Theory and Experiment*.

[3] Sharpee TO (2013) Computational identification of receptive fields *Ann Rev. Neurosci.*.