Fall 2017 - Mini project on population decoding

Due noon, Thursday, November 16.

Suppose you have 100 neurons to cover a range of "stimulus" values (it could be a motor variable like reaching direction or running speed, or a sensory variable like sound frequency or visual distance). For the moment we will consider a 1-dimensional stimulus variable "stim", plotted on the x-axis from 1 to 100. Each neuron has an idealized tuning curve that gives its average firing rate over the range of stimulus values, given by a Gaussian function

$$f(stim) = maxspikerate * exp(\frac{-(stim-\mu)^2}{2\sigma^2})$$

where μ is the tuning curve center (an integer in the range of 1 to 100), and σ is the tuning curve width (standard deviation). But you don't have access to $f(stim)$ directly. Rather, the neuron generates spikes randomly at the rate f according to a Poisson process, and you just get to count spikes over some time interval.

Digression about a Poisson process. In this type of random process, which governs things like radioactive decay, the parameter λ tells you the number of events you expect to measure in a given time interval T, which in our case would be $\lambda = f(stim) * T$. The Poisson distribution is the probability distribution over the number of events k (i.e. spikes) you will actually measure over that time interval in repeated trials given that λ. So if $\lambda = 5$, you will get 5 spikes on average, but you can possibly get $k = 0, 1, 2, \ldots$ to an arbitrarily large number of spikes. The probability of getting k events when you expect λ is given by

$$\frac{\lambda^k}{k!} e^{-\lambda}$$

Check out out the wikipedia page to see what Poisson distributions look like for different values of λ.

Suppose the maximum firing rate the neuron generates is 100 Hz at the peak of its tuning curve, but you are only counting spikes over a 100 ms period, then the maximum expected number of spikes you will ever expect (which is at the peak of the tuning curve) is $\lambda = 10$ spikes. Note that it's fine to *expect* a non-integer number of spikes in a time interval, so λ could be 2.379, but the Poisson distribution gives you probabilities of counting *integer* numbers of events k given that (possibly non-integer) value of λ, since integers are all that can actually happen (i.e. you can't count 5.73 spikes).

So here is your goal. You will generate a uniform random stimulus between 0 and 100. In matlab you can do this like so:

```
stim = 100 * rand;
```

You will model an array of gaussian tuning curves with centers μ at 0, 1, 2, … 100 (these can go in an array), and tuning curve widths σ; these can go in a different array. You can assume for this project that all the neurons have the same value of σ.

Your goal: determine whether narrow or wide tuning curves are better for accurately representing (i.e. allowing you to accurately decode) the stimulus from the noisy firings of the neuron in the array.

Here are the steps:

1. Generate a random stimulus (see above).

2. Model the “clean” responses of the 100 neurons to the stimulus, according to the gaussian tuning curves. Given the numbers above, this would be an array of λ values between 0 and 10. Plot it: it will look like a bell-shaped bump on the x axis centered wherever the stimulus was.

3. Model the “dirty” response, i.e. the number of spikes actually generated by each neuron given its expected number of spikes λ in the clean response array. These noisy spike counts go into another array. In matlab, you can draw a number k from a Poisson distribution with mean lambda as follows:

   k = poissrnd(lambda)

   Try it and see what k’s come out. Plot the dirty response of the whole neuron array. To make a scatter plot of this in matlab:

   scatter(0:100, k)

4. Now it’s time to decode. The values in the dirty array is what the brain (or you, who is recording from the brain) gets to know about the stimulus. Your job is to decode this as well as possible. The optimal decoding scheme under these circumstances is to simply calculate the weighted average of the tuning curve centers, weighted by their (dirty) response levels. Note this is a dot product, divided by the sum of the dirty responses. Careful that in matlab, you may need to use the transpose operation when you take the dot product (which is a row * a column). And when you divide an array by a number, remember you need the elementwise operation ./

5. Now see how well you did! Generate 100 random stimuli in a loop, for each stimulus generate the dirty neuron responses, decode the dirty responses, and compare the actual stimulus to the decoded stimulus by (1) making a scatter plot of the two values, and (2) computing the mean squared error (MSE) of your prediction. Hint: when you calculate MSE, restrict yourself to the middle of the range (e.g. between 40 and 60), in other words, avoid the ends of the array where other systematic biases occur that have nothing to do with noise. Hopefully the data will lie pretty close to the diagonal meaning the decoded values match the stimuli closely. Zoom in to see the spread above and below the diagonal, which is your decoding error.

6. Now comes the crux of your investigation: determine experimentally whether it’s better to have narrow or wide tuning curves to maximize decoding accuracy. Try tuning curve widths from 1 (or less) to at least 10. You do not need to try every value or generate plots for every value; choose 3 plots that represent the range of cases. You will determine the best tuning curve width by examining by eye the decoding errors around the diagonal in your decoded-vs-actual scatter plot, and backing up your conclusion with your MSE measure. State your conclusion. Also, what is going on near the borders of the array? How should this be handled?

To Turn In: Please turn in a PDF document with the graphs and explanations you use to "determine experimentally whether it’s better to have narrow or wide tuning curves to maximize decoding accuracy." Be detailed in your analysis (with both graphs and words) and answer any questions posed in the homework prompt. A good place to start is to go step by step, and turn in your results from each step. Then go on with your analysis from part 6. Explain things in words as you go. Copy and paste your matlab code to the end of your report.

Extra credit: Repeat #6 for a 2-D array of receptive fields.