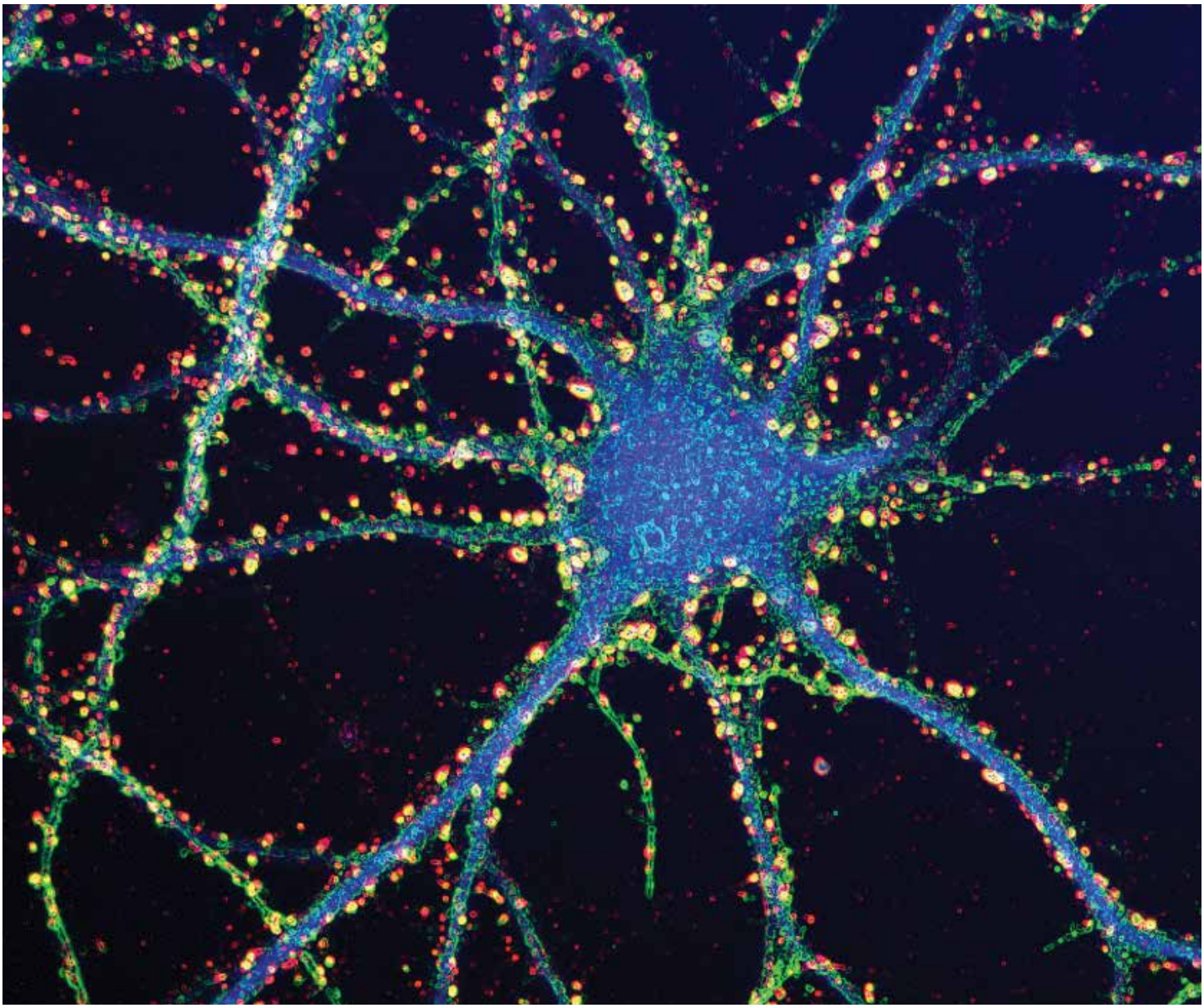

Neuroscience 2016

SHORT COURSE 2

Data Science and Data Skills for Neuroscientists

Organizers: Konrad P. Kording, PhD, and Alyson Fletcher, PhD

SfN Society for Neuroscience

Short Course 2

Data Science and Data Skills for Neuroscientists

Organized by Konrad P. Kording, PhD, and Alyson Fletcher, PhD

Society for Neuroscience

Please cite articles using the model:
[AUTHOR'S LAST NAME, AUTHOR'S FIRST & MIDDLE INITIALS] (2016)
[CHAPTER TITLE] In: Data Science and Data Skills for Neuroscientists.
(Kording K, Fletcher A, eds) pp. [xx-xx]. San Diego, CA: Society for Neuroscience.

Neuroscience 2016

SHORT COURSE 2

Data Science and Data Skills for Neuroscientists

Organized by: Konrad P. Kording, PhD, and Alyson Fletcher, PhD
Friday, November 11, 2016
8 a.m.–6 p.m..
Location: San Diego Convention Center • Room: 6C • San Diego, CA

TIME	TALK TITLE	SPEAKER
7:30–8 a.m.	**Check-in**	
8–8:10 a.m.	Opening Remarks	Konrad P. Kording, PhD • Northwestern University Alyson Fletcher, PhD • UCLA
8:10–9:10 a.m.	• Canonical data analysis cascade • MATLAB for neuroscientists • Peristimulus time histograms (PTSHs)	Pascal Wallisch, PhD • New York University
9:10–10:10 a.m.	• Fundamentals of statistics: densities, mean-squared error, and regression • Bootstrapping and confidence intervals • Time of maximal firing rates	Robert Kass, PhD • Carnegie Mellon University
10:10–10:30 a.m.	**Morning Break**	
10:30–11:10 a.m.	• Fitting tuning curves • Error bars and model selection with bootstrap	Konrad P. Kording, PhD • Northwestern University
11:20 a.m.–12:10 p.m.	• Classification in neural systems • Poisson point process models for spiking • Model selection: cross-validation and overfitting	Alyson Fletcher, PhD • UCLA
12:10–1:10 p.m.	**Lunch**	
1:10–2:10 p.m.	• Generalized linear models (GLMs) with spike history • GLMs with neuron–neuron interactions • Regularization, ridge regression, and smoothing	Jonathan Pillow, PhD • Princeton University
2:20–2:40 p.m.	Sparsity in neural decoding	Alyson Fletcher, PhD • UCLA
2:50–3:20 p.m.	Multilayer models and deep learning	Konrad P. Kording, PhD • Northwestern University
3:20–3:40 p.m.	**Afternoon Break**	
3:40–4:40 p.m.	• Neural population models principal component analysis (PCA) • Time series models and Gaussian process factor analysis (GPFA) • State space dynamical systems	Maneesh Sahani, PhD • University College London
4:50–5:50 p.m.	• Local network structure of neural data • Clustering and modularity in networks • How networks change: the dynamics of graphs	Danielle Bassett, PhD • University of Pennsylvania
5:50–6:00 p.m.	**Closing Remarks**	Konrad P. Kording, PhD • Northwestern University Alyson Fletcher, PhD • UCLA

Table of Contents

Introduction

Data skills and data science are moving away from being specialties that a small minority of computational scientists get excited about to becoming central tools used by the bulk of neuroscientists. The objectives for this short course are twofold. First, we will teach basic, useful data skills that should be in the toolkit of virtually all neuroscientists. Second, we will survey the field of more advanced data science methods to give participants an overview of which techniques to use under which circumstances.

The course is structured around hands-on programming exercises. Lectures will go hand in hand with tutorials. During the day, the focus will be on participants solving many frequently encountered data analysis problems themselves, aided by lectures given by leading experts.

The course will cover a broad range of topics. It will start with basic topics, including data cleanup, data visualization, and fundamental statistical ideas. It will then progress to everyday problems such as fitting functions to tuning curves, adding error bars, and decoding. More advanced topics will include generalized linear models, dimensionality reduction, time-series data, and networks. In this way, the course should impart a solid understanding of the basic techniques used for neural data analysis.

The Basics of Neural Coding

Konrad P. Kording, PhD

Rehabilitation Institute of Chicago
Northwestern University
Chicago, Illinois

NOTES

The Basics of Neural Coding

How do neurons represent the world?

At first glance, there seems to be a world of difference between fundamental physiological features of neurons, such as firing rates and tuning curves, and the quantitative measurements of perception and behavior. Yet we know that somehow, neuronal processes must underlie all of perception and behavior. The goal in this chapter is to indicate how this gap can be bridged. We will start by summarizing how neurons encode variables in the world.

Historically, we have treated the brain as a "black box" that performs probabilistic inference. This suffices if one is interested primarily in modeling behavior. However, in systems neuroscience, elucidating the link between biological and psychological states is a central objective. In this chapter, we explore the connection between behavior and neural activity from the perspective of our normative framework. Because we have seen that abundant evidence exists for Bayesian optimality at the behavioral level, at least in simple tasks, we will ask the following questions:

- How do neurons represent states of the world?
- How do neurons represent likelihood functions?
- How do neurons use these representations to calculate posterior distributions?

The Bayesian normative framework offers a means of addressing these questions that runs counter to the bulk of neural modeling work. Modelers often construct neural networks out of simulated, more or less biophysically plausible elements, and examine their emergent dynamics. In many cases, this "bottom-up" approach has the disadvantage that the link of these networks to behavior is tenuous or only qualitative. In contrast, it is possible to take a top-down approach to neural computation, in which the construction of a neural model is guided by a normative behavioral model. In this approach, the search for a neural implementation of a perceptual phenomenon is guided simultaneously by behavioral and physiological constraints.

In this chapter, we will first computationally describe neurons, formalizing them as a generative model that produces outputs in response to either direct inputs or to a stimulus given to the brain. We will introduce the concepts of a neural population and neural variability. Using these concepts, we can understand how neurons can encode a probability distribution and therefore can carry implicit knowledge of uncertainty. As an introduction to neural modeling and population coding, the present chapter is limited in scope. Our main goal here is to provide the necessary background information for an understanding of the representation of probability at the neural level.

A focus on generative models of neurons

When we define the generative model with respect to world states and sensory observations, we conveniently represent the sensory observation as a measurement that lives in the same space as the stimulus. For instance, we conceive a sound stimulus at a particular location as producing an observation drawn from a Gaussian distribution centered at that location. At the neurobiological level, however, the sensory observation is not such a measurement, but rather neuronal activity evoked by the sensory stimulus: neurons encode the physical stimulus as a pattern of spiking activity, and the brain must somehow decode this activity in order to infer the world state. Here we take a first look at the neuronal level, and we consider the mapping from sensory stimuli to the spiking activity produced in sensory neurons. Once we fully specify how sensory stimuli give rise, in a probabilistic way, to neural activities, we will be in a position to formulate how neurons may encode uncertain stimuli and how the brain can infer the state of the world from neuronal activity.

The brain does not have direct access to the sensory input, I. Rather, the sensory input activates receptor cells such as auditory hair cells or photoreceptors, which in turn activate nerve fibers (axons), causing electrical impulses (action potentials or spikes) to travel into the CNS. These impulses are the data upon which the brain makes inferences about the world. The activity of neurons in a relevant brain area in response to a stimulus, denoted $\boldsymbol{r}$, constitutes the internal representation or observation of that stimulus. Neural activity is variable: when the same sensory input I is presented repeatedly, $\boldsymbol{r}$ will be different every time. This is to the result of stochastic processes that inject variability: photon noise, stochastic neurotransmitter release, stochastic opening and closing of ion channels, etc. Thus, a probability distribution $p(\boldsymbol{r} \mid I)$ is needed to describe the neural activity.

Since activity $\boldsymbol{r}$ is variable even when the input I is kept fixed, and I is variable even when the stimulus s is kept fixed, it follows that r is variable when s is kept fixed, even if the nuisance parameters are not variable. This neural variability is captured in a probability

NOTES

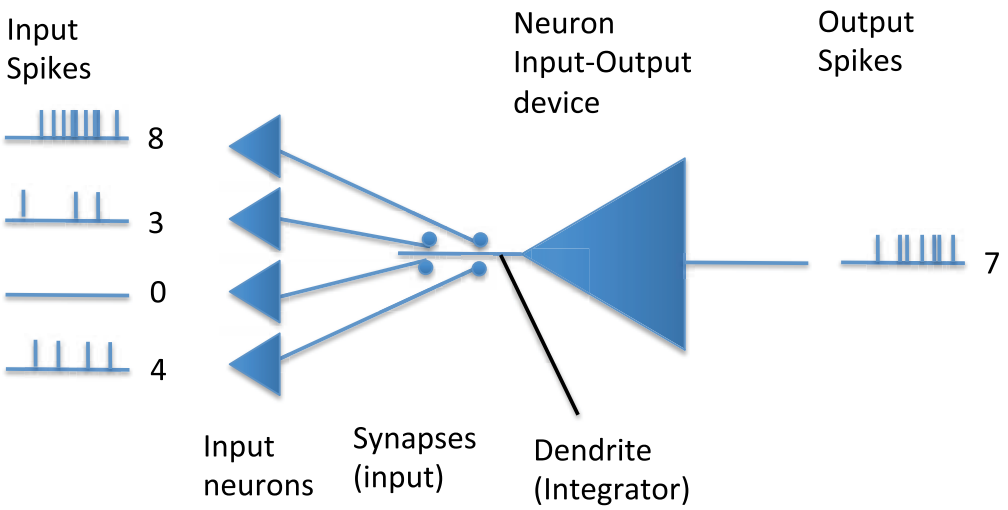

Figure 1. Biology of a neuron: neurons as input–output devices. Neurons transmit electrical impulses (action potentials, aka spikes) away from their cell bodies, along output structures (axons). In a majority of cases in the mammalian nervous system (the so-called chemical synapse), when the spike reaches the end of an axon (axon terminal), it induces the release of neurotransmitter molecules that diffuse across a narrow synaptic cleft and bind to receptors on the input structure (dendrites) of the postsynaptic neuron. The effect of the transmitter released by a given input neuron may be inhibitory (reducing the probability that the postsynaptic neuron will fire spikes of its own) or excitatory (increasing the probability that the postsynaptic neuron will fire), depending on the transmitter released and on the receptor that binds it.

Each neuron receives input from a (typically large) number of other neurons. For simplicity, this figure shows a single postsynaptic neuron receiving just four inputs. Each of the input neurons fires a number of spikes over a relevant time interval. These result in neurotransmitter release onto our neuron of interest. The postsynaptic neuron integrates these inputs and produces an output spike train of its own. For our purposes, we will simplify the modeling of the neuron to modeling the number of output spikes, either in response to a stimulus or in response to the numbers of input spikes it receives from other neurons.

distribution $p(\boldsymbol{r} \mid s)$. The main goal of this chapter is to define and motivate mathematical descriptions of $p(\boldsymbol{r} \mid s)$. We would like you to think of $\boldsymbol{r}$ as the spiking activity in early sensory cortex, such as primary visual cortex area V1. The stimulus is a basic feature in the outside world, such as position, orientation, etc.

Neurons as Mappings from an Input to an Output

There are many ways of modeling neurons, ranging from detailed biophysical models of the structure and function of individual ion channels, to highly abstract models of neurons as information processing units. For our purposes, we treat neurons as simple input–output systems (Fig. 1). A neuron receives inputs from a group of other neurons. Over the relevant time scale, it receives a number of spikes from each of the input neurons and produces (or emits) a number of output spikes. A neuron is thus characterized by its transfer function, spikes = f(input spikes).

We will work toward asking the question, "How could a given computation be implemented?" and not so much, "Which specific neural circuits actually implement that computation?" That being said, for making testable physiological predictions, it is clearly important to focus on a particular species and brain area. However, even so, the localization of the computation is of secondary interest to the mechanisms of the computation. Our goal is to understand potentially ubiquitous neural processing mechanisms rather than to model a specific circuit.

Tuning Curves

The concept of "tuning curves" became popular with the pioneering experiments of Hubel and Wiesel in the late 1950s (Hubel and Wiesel, 1959). They recorded from the V1 region in cat while stimulating with illuminated oriented bars (Fig. 2*a*). They found that the response of a cortical neuron was systematically related to the orientation of the stimulus. There exists one orientation of the stimulus where the neuron fires most rapidly: the neuron's preferred orientation. For other orientations, the activity decreases with increasing angle relative to the preferred orientation. A plot of the mean firing

NOTES

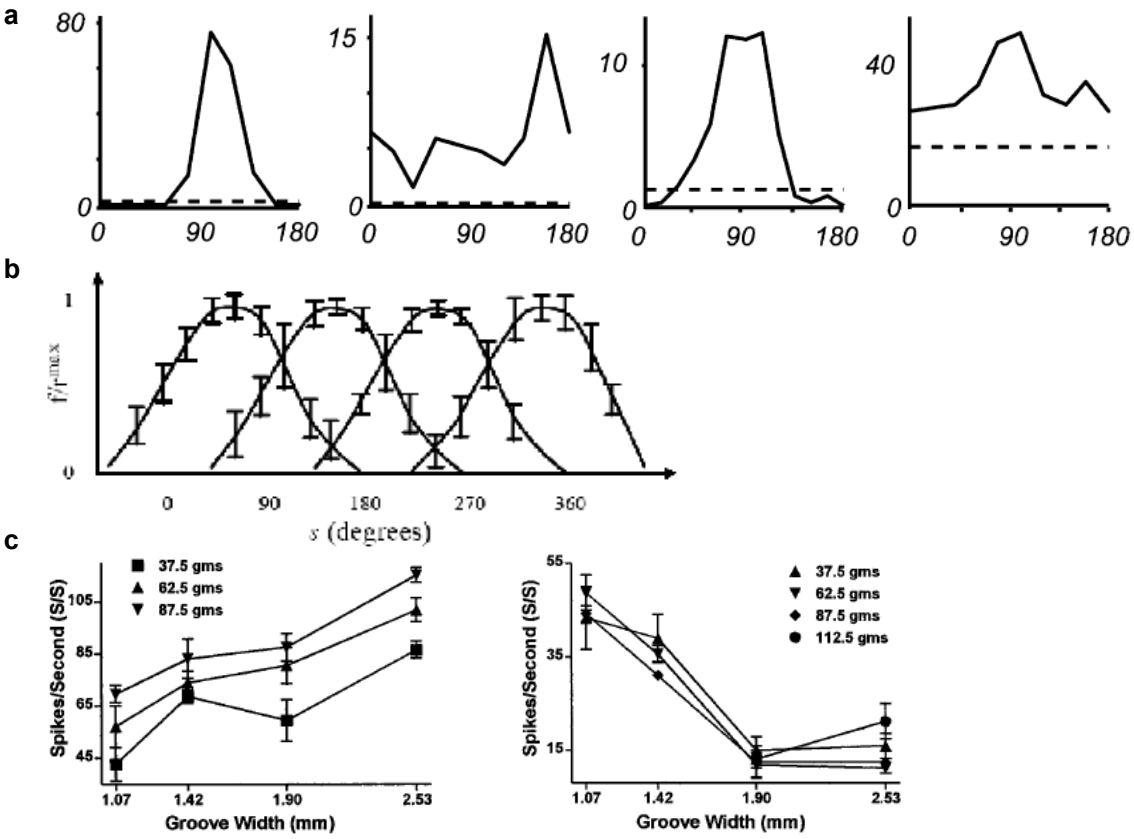

Figure 2. Tuning curves. ***a***, Tuning curves for orientation in macaque primary visual cortex (V1). The dashed line represents the spontaneous firing rate. Reproduced from Shapley et al. (2003), their Fig. 10. Copyright 2003, Cell Press. ***b***, Normalized tuning curves for the direction of air current in four interneurons in the cercal system of the cricket. Reproduced from Theunissen and Miller (1991). Copyright 1991, the American Physiological Society. ***c***, Tuning curves for the width of the groove in a tactile grating in macaque second somatosensory cortex (S2). Different curves are for different magnitudes of the contact force (expressed as mass). Reproduced from Pruett et al. (2000), their Figs. 3*A* and 4*A*. Copyright 2000, the American Physiological Society.

rate (e.g., spikes per s) as a function of angle describes the neuron's tuning curve. In the case of many visual neurons, this is a unimodal function (Fig. 2*b*).

Tuning curves can have a wide variety of shapes, depending on the species, the brain area, and the stimulus features. For example, in motor cortex, we find that neural responses influence the direction of movement of the hand of a monkey. Instead of narrow unimodal functions, we usually find broad tuning curves. In auditory cortex, the frequency of the sounds stimulus affects the firing rate of the neuron in a complex tuning curve. And in the hippocampus, a region of the mammalian brain involved in memory acquisition and navigation, there is a two-dimensional representation of positions. In experiments with rats, firing rates of hippocampal neurons depend on both x and y positions. The important thing in all these cases is that reasonably simple tuning curves characterize the mapping from sensory stimuli to the activity of neurons.

Bell-shaped tuning curves

When modeling tuning curves, scientists usually use simple functions. When we model tuning curves like those found for neurons in V1, we typically use bell-shaped tuning curves. Because rotating a bar by 180 degrees leads to the same response, we usually use circular Gaussian functions (Von Mises function, Fig. 3*a*). By contrast, when scientists deal with auditory stimuli of varying amplitude, tuning curves typically show increasing activities. Piecewise linear functions can be used for such scenarios (Fig. 3*b*). We will now consider some tuning curve functions in more detail.

NOTES

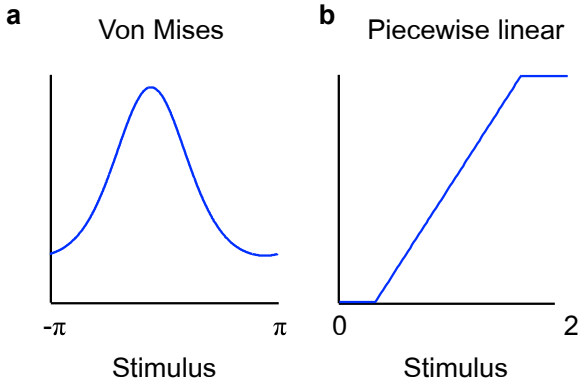

Figure 3. Model tuning curves. Two types of functions are commonly used to model tuning curves: bell-shaped (***a***) or monotonic (***b***). The stimulus ranges are merely illustrative.

It should be noted that, even though the tuning curve might look bell-shaped or even be described by a Gaussian function, it is certainly not a probability distribution. That is, it is not normalized and generally has no direct relation to probability distributions.

In V1, neurons are tuned to the orientation of a visual stimulus, such as a bar. The tuning curve is typically unimodal and symmetric around the preferred orientation. Furthermore, the mean spike rate is the same at any angle and at that angle plus or minus 180 degrees (as the bar stimulus is identical when rotated by 180 degrees). A common way to describe such a curve is to use a Von Mises function (also called a circular Gaussian) (Fig. 3*a*):

$$f_i(s) = ge^{\kappa(\cos(s - s_i) - 1)} + b. \qquad (1)$$

Here, κ is called the concentration parameter. The higher κ, the more narrowly the neuron is tuned. This function has been used to fit tuning curves over orientation, such as those in Fig. 2*a*.

Monotonic tuning curves

The tuning curve describes a neuron's mean activity as a function of the presented stimulus. Tuning curves are usually bell-shaped or monotonic. Various mathematical functions have been used to model them. A non–bell-shaped tuning curve occurs in some neurons. An example is a monotonic tuning curve such as those shown in Fig. 2*b*. Several possibilities can be considered. The simplest form is a rectified linear function:

$$f_i(s) = [gs + b]_+, \qquad (2)$$

where g is positive for monotonically increasing, and negative for monotonically decreasing tuning curves.

Note that these neurons do not truly have a preferred stimulus (for the monotonically decreasing tuning curves, you could say it is 0, but that does not help much). A clear problem of a rectified linear function is that it is unbounded: as s increases, $f(s)$ does not stay below any maximum value. This is unrealistic, since neurons have a limited dynamic range and cannot fire more than a certain number of spikes per second, no matter how large s becomes.

It may come as a surprise that the exact shape of the tuning curve is not critical to most of the theory we will discuss here. The theory will be general and work for tuning curves of any shape. However, the shape is of great practical relevance, since it is the starting point of any implementation of neural population activity. Moreover, we will occasionally use a specific functional form to allow for analytical calculations.

More detailed models for the tuning curve

It is possible to describe the mapping from stimulus s to the output of a V1 neuron in much more detail. Instead of describing the world state as a scalar orientation, we can describe the entire two-dimensional image as a vector $\boldsymbol{I} = (I_1, I_2, \ldots I_m)$, where m is the number of pixels. Figure 3 shows examples of images of an oriented bar similar to the ones used by Hubel and Wiesel (1959). As the orientation of the bar changes, the entire image changes, so we can consider $\boldsymbol{I}$ to be a function of s and write it as $\boldsymbol{I}(s)$.

Each neuron has a spatial filter, which means that it will respond positively to light in certain locations in the image and negatively to light in other locations. In other words, the neuron associates a weight, positive or negative, with every pixel in the image. We call this filter or weight vector w, and it can itself be visualized as an image. A typical V1 filter is shown in Figure 3; this is built so that if the image contains an orientation at the location of the filter, then the neuron will respond strongly. The neuron's average spike count in response to the image is then a sum of the pixel intensities multiplied by the corresponding weights:

$$f(s) = w_1 I_1(s) + w_2 I_2(s) + \ldots + w_m I_m(s) = \boldsymbol{w} \cdot \boldsymbol{I}(s). \quad (3)$$

Because orientation is varied, this produces a tuning curve similar to the one in Figure 3. Spike counts would still be generated from a Poisson variability with mean $f(s)$, just like in the main text. The model of tuning curves in Equation 3 is called a "linear model," because $f(s)$ is a linear combination of image intensities; it is not a linear function of orientation s.

NOTES

There are many ways to extend this model. For instance, one can postulate that $f(s)$ is a nonlinear (but monotonically increasing) function of $\boldsymbol{w} \times \boldsymbol{I}(s)$. The resulting family of models is called LNP models, where L stands for linear (Eq. 3), N for nonlinear, and P for Poisson.

Variability

So far, we have discussed a neuron's selectivity: which stimuli it spikes to or "likes," as described by its tuning curve. However, if we view neuronal activity as resulting from a statistical generative model, we need to specify both neurons' stimulus-dependent activity as well as their (also potentially stimulus-dependent) variability. For an identical, repeated stimulus, how much variation exists in the response from trial to trial? This variability will be critical for performing inference at the neural level. Here we will focus on Poisson variability.

Poisson variability

Poisson variability (Fig. 4*a*) is defined for a spike count, e.g., the number of spikes elicited by a flash of light that is presented for 10 ms. Spike count is a nonnegative integer; it can be 0. Suppose a stimulus s is presented, and the mean spike count of a neuron in response to this stimulus is $\lambda = f(s)$, which does not need to be an integer. λ_i is also called the "rate" of the Poisson process. Then the actual spike count will vary from trial to trial, around λ. For every possible count r, we seek its probability. A "Poisson process" (or in our context, a Poisson spike train) is defined as follows. Imagine a fixed time interval, and divide it into small bins (e.g., 1 ms each). We assume that each bin can contain 0 spikes or 1 spike, and that the occurrence of a spike is independent of whether and when spikes occurred earlier (it is sometimes said that a Poisson process "has no memory"). It can be proved in such a case (see Sample Problems) that for a Poisson process with mean λ, the probability of observing a total of r_i spikes on a single trial is given by the Poisson distribution:

$$p(r_i \mid \lambda_i) = \frac{1}{r_i!}\ e^{-\lambda_i} \lambda_i^{r_i}\,. \qquad (4)$$

Here, $r!$ (read "r factorial") is defined as $1 \times 2 \times 3 \times \ldots \times r$.

The Poisson distribution is shown for $\lambda = 3.2$ and $\lambda = 9.5$ in Figure 4*b*. Keep in mind that, while r is an integer, λ can be any positive number. For low λ, the distribution is less symmetrical than for high means. In fact, at high mean firing rates, the distribution looks roughly Gaussian. However, note that the Poisson distribution is discrete, so drawing it as a continuous curve would be a mistake.

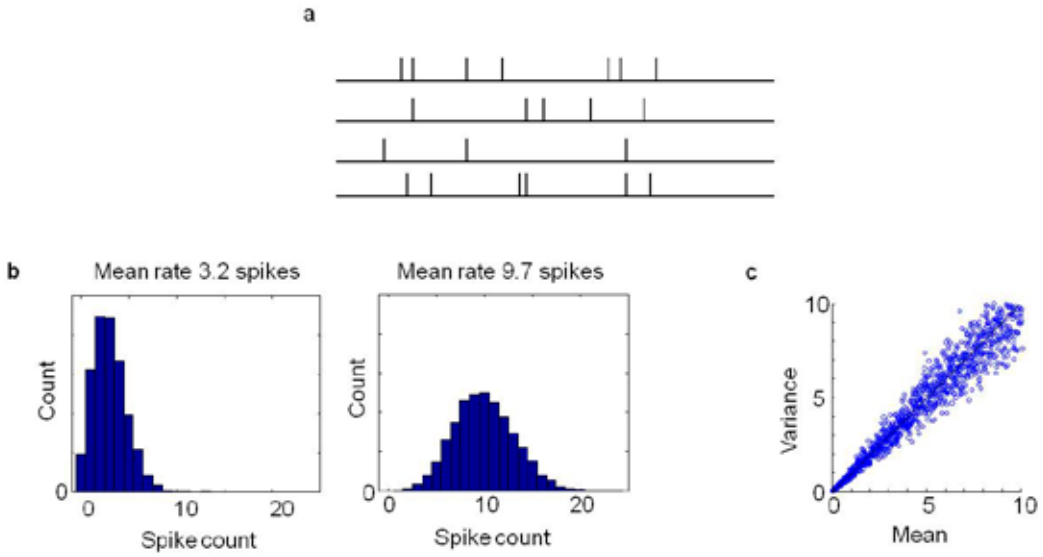

Figure 4. Poisson variability. ***a***, Hypothetical spike trains evoked in the same neuron by the same stimulus, repeated four times (trials). Not only do the spike times differ between trials, but the spike counts also differ. ***b***, Histograms of the spike count of a single Poisson neuron, with different mean rates. ***c***, In a Poisson process, the variance is equal to the mean. This is illustrated by simulating 100 spike counts for each value of the mean and calculating the variance. Variance is plotted against the mean. The diagonal is the unity line.

NOTES

An important property of the Poisson distribution is that the variance of a Poisson-distributed variable is equal to its mean: if the mean firing rate of a Poisson neuron is λ, then the variance of this neuron's spike count is also λ (Problem 2 and Fig. 4*c*). The ratio of the variance to mean of a neuron's spike count is called the "Fano factor"; for a Poisson process, the Fano factor is 1.

For our generative model of neural firing, we need to specify the probability of a firing rate, r, as a function of the stimulus, s. To do this, we note that λ is a function of the stimulus: it is the height of the tuning curve (the neuron's average firing rate) at stimulus level s. Therefore, in terms of the stimulus, Equation 4 can be written as

$$p(r_i \mid s) = \frac{1}{r_i!}\ e^{-f_i(s)} f_i(s)^{r_i}. \qquad (5)$$

This is the form we will use frequently. Note that the neuron's tuning curve, $f_i(s)$, plays a role in the neuron's variability, but that it was not necessary to commit to any specific tuning curve shape to derive the form of the variability. In other words, Poisson variability can go together with any type of tuning curve, whether bell-shaped or monotonic. It is a common mistake to confuse the tuning curve with variability. This is understandable when one compares plots like those in Figures 3*a* and 4*b*, but the meaning of the axes in these plots is completely different. The relationship between the tuning curve and the variability in firing is illustrated in Figure 5.

It may at first appear that stimuli that evoke higher firing rates will be less informative, because higher firing rates are associated with more variability (e.g., spike rate variance = mean spike rate for a Poisson process). However, higher firing rates in fact convey more information. To see this, consider the mean as the signal and the standard deviation (SD) of the variability as the noise. Then the noise (square root of the variance) equals the square root of the mean. The signal-to-noise ratio therefore increases as the square root of the mean. We will later make this statement more precisely for the case of a population of neurons.

Numerical example

The rate of a Poisson neuron is $\lambda_1 = 3.2$. What is the probability that this neuron is silent? That it fires 1 spike? That it fires 10 spikes?

Solution: From Equation 4, the probability that the neuron fires 0 spikes is $1/0! \times \exp(-3.2) \times 3.2^0 = \exp(-3.2) = 0.04$, or 4%. The probability that the neuron fires 1 spike is $1/1! \times \exp(-3.2) \times 3.2^1 = \exp(-3.2) \times 3.2 = 0.04 = 0.13$, or 13%. The probability that the neuron fires 10 spikes is $1/10! \times \exp(-3.2) \times 3.2^{10} = 0.001$, or 0.1%.

More realistic models

Poisson variability is reasonably physiologically realistic, but with a number of caveats. Real Fano factors of cortical neurons are often close to 1, but can take values as low as 0.3 and as high as 1.8. Another unrealistic aspect of Poisson variability is that it assumes that spikes are independent of previous spikes. This is clearly not true: after a neuron fires a spike, it cannot fire again for a short duration, called the "refractory period" (typically several milliseconds). Thus, during that period, the probability of firing a spike is 0, contradicting the way we defined the Poisson process. There exists a literature that extends the models we discuss here to more realistic models, but it is beyond the scope of this chapter.

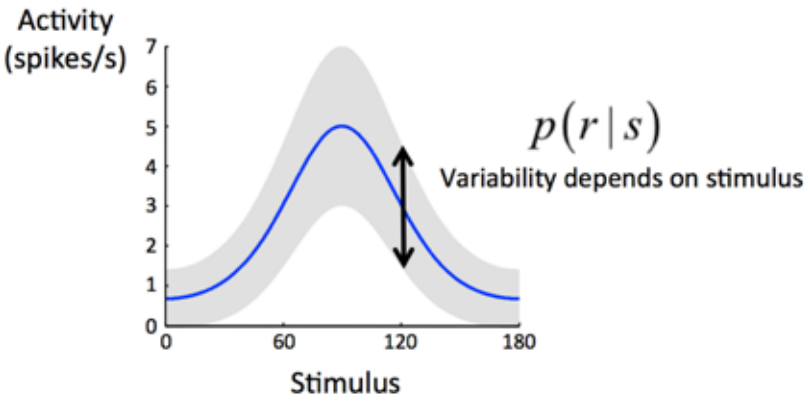

Figure 5. Poisson firing rate variability. The tuning curve (blue) shows mean spikes per second evoked by the stimulus. The shaded gray region illustrates the variability in firing rate upon the administration of repeated, identical trials.

The origin of variability

The origin of neural variability is unknown and likely a combination of factors. Part of it has an external origin: when the same value of a visual stimulus variable is presented, this might not mean that the retinal image is identical. In many experiments, stimulus reliability is controlled by manipulating the amount of external noise. In those cases, the retinal image will be different even though

NOTES

the stimulus variable of interest is the same. Ideally, variability is measured under repeated presentations of the exact same physical image. This has been done in area MT (middle temporal) in the macaque, and response variability was still found. This variability can be attributed to internal factors. Internal sources of variability include neurotransmitter release and synaptic failure, both of which are stochastic processes.

Key points

A neuron's response to a particular stimulus varies from trial to trial. Such variability or "noise" can be described by a probability distribution: Poisson or Gaussian are common choices. In real neurons, variance is approximately proportional to the mean. The origin of variability is not yet understood.

Population Codes

Neurons are often part of populations that respond to the same state-of-the-world variable. Population activity is described through the tuning curves of individual neurons as well as their joint variability. "Neural populations" are groups of neurons that are all selective for a particular stimulus feature. The neurons in a population are often but not necessarily located close to each other in the brain, and they often have similarly shaped tuning curves but with different preferred stimulus values; a population consisting of neurons with the identical tuning curve would not be particularly useful, since all neurons would "cover" the same restricted region of stimulus space.

A "population code" refers to the stimulus-evoked firing rates of a population of neurons with different tuning curves. Population codes are believed to be widespread in the nervous system. For instance, in areas V1 and V4 of the macaque, population codes exist for orientation, color, and spatial frequency. In the hippocampus in rats, a population code exists for the animal's body location. The cercal system of the cricket has a population code for wind direction. Secondary somatosensory area (S2) in the macaque has population codes for surface roughness, speed, and force. The post-subiculum in rat contains a population code for head direction. Primary motor cortex (M1) in macaque uses populations coding for direction of reach. Even abstract concepts such as number appear to be encoded by population codes in the prefrontal cortex.

The firing rates of the set of neurons depicted in Figure 2*a* form a population code; in fact, the cricket cercal system population consists of exactly the four neurons shown there. An idealized example with Gaussian tuning curves is drawn in Figure 6*a*. In this example, we show 10 neurons with preferred stimuli equally spaced on an interval. Such equal spacing is an idealization just as an exact Gaussian shape is. Yet, this is usually how a population code is simulated. We call the preferred stimuli $s_1, \ldots, s_N$, where N is the number of neurons in the population. The tuning curves differ only in their preferred stimuli, so Equation 1 is valid; $f_i(s)$ is the tuning curve of the neuron with preferred stimulus s_i. The set of tuning curves of all neurons in the population, $\{f_1(s), \ldots, f_N(s)\}$, is denoted as a vector-valued function $\boldsymbol{f}(s)$.

Just as we modeled the variability of a single neuron's spike count, we can model the variability of the entire population. We denote by $\boldsymbol{r}$ the vector of spike counts of the neurons in the population: $\boldsymbol{r} = (r_1, \ldots, rN)$. This is also called a "population pattern of activity." The probability of observing a pattern of activity $\boldsymbol{r}$ in response to a stimulus s is denoted as $p(\boldsymbol{r} \mid s)$. The mean population pattern of activity over many trials is a smooth curve that resembles the tuning curve (Fig. 6*b*). Whereas the tuning curve shows the mean activity of one neuron in response to different stimuli, the population pattern shows the mean activity of every neuron in response to a single stimulus.

The vector notation for population activity is simply for convenience; it does not have a deeper meaning. One could just as well write $r_1, \ldots, r_N$ wherever $\boldsymbol{r}$ appears (and similarly for $\boldsymbol{f}$), but this would make equations unnecessarily cluttered. Within the vector $\boldsymbol{r}$ (or $\boldsymbol{f}$), the ordering of the neurons has no meaning at all. We will typically order them by their preferred stimulus, only to make visualizations of population patterns like the one in Figure 6*c* look sensible.

In analogy to the single-neuron case, we now discuss Poisson and Gaussian variability in the population.

Independent Poisson variability in a population

The simplest assumption we can make about the population is that for a given stimulus, the responses of the neurons are drawn independently from each other, and that each response follows a Poisson distribution (but with its own mean). If random variables are independent from each other (in this case, for a given stimulus), their joint probability distribution is the product of the individual distributions (again for a given stimulus). This means that we can write the population variability as

$$p(\mathbf{r} \mid \boldsymbol{s}) = p(r_1 \mid s) \ldots p(r_N \mid s) = \prod_{i=1}^{N} p(r_i \mid s). \quad (6)$$

NOTES

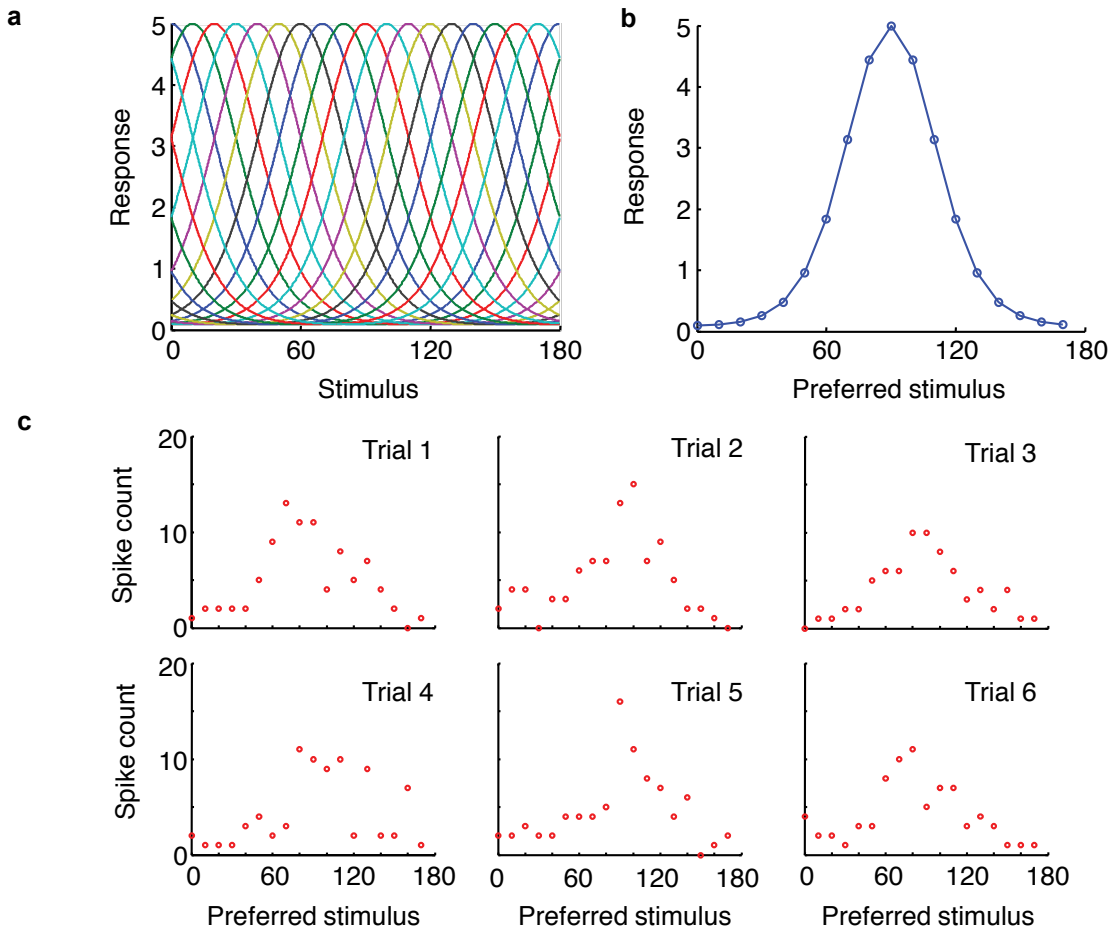

Figure 6. Neural population. ***a***, Idealized bell-shaped tuning curves of a population of 18 neurons responding to an arbitrary periodic stimulus. Preferred stimuli are equally spaced. ***b***, Mean activity of all neurons in the population in (***a***), ordered by their preferred stimulus, in response to a stimulus value of 90. ***c***, Six single-trial patterns of activity drawn from an independent Poisson population with Gaussian tuning curves and gain $g = 10$.

The last equality is just a notation for the product. Now we substitute Equation 5 for $p(r_i \mid s)$:

$$p(\mathbf{r} \mid \boldsymbol{s}) = \prod_{i=1}^{N} \frac{1}{r_i!} e^{-f_i(s)} f_i(s)^{r_i} \qquad (7)$$

This is the probability distribution of independent Poisson variability. Figure 6*c* shows patterns of activity drawn from this distribution, if tuning curves are Gaussian. In Problem 3, you will simulate such patterns yourself. Spike count is plotted as a function of the preferred stimulus of the neuron. Each dot corresponds to the activity of one neuron. We could have plotted them in any order, but visually, it is most insightful to order the neurons by their preferred stimulus. Each pattern in Figure 6*c* is the population analog of one spike count in the histograms of Figure 4*b*. For the population, it is impossible to draw the histogram, since $\boldsymbol{r}$ is now an N-dimensional vector, and we cannot draw histograms in N dimensions. Nevertheless, we can still calculate the probability of each pattern of activity like those in Figure 6*c*.

Numerical example

We assume a population of 9 independent Poisson neurons with Gaussian tuning curves and preferred orientations from –40 to 40 in steps of 10. The tuning curve parameters have values $g = 10$, $b = 0$, and $\sigma_{tc} = 20$. A stimulus $s = 0$ is presented to this population. What is the probability of observing a pattern of activity $\boldsymbol{r} = (3, 1, 7, 5, 8, 8, 7, 0, 2)$?

NOTES

Solution: Under our assumptions about the tuning curves, the mean activity of the ith neuron is $f_i(s = 0) = 10 \times \exp(-s_i^2/800)$. Across the population, this gives mean activities (1.3, 3.2, 6.1, 8.8, 10, 8.8, 6.1, 3.2, 1.3). Then from Equation 7:

$p(\mathbf{r} \mid s = 0) = e^{-1.3} \times 1.3^3/3! \times e^{-3.2} \times 3.2^1/1! \times \ldots \times e^{-1.3} \times 1.3^2/2! = 2.4 \times 10^{-9}$.

This number is striking because it is so small. How can it be that a pattern of activity that is not so different from the mean activities is so improbable? The reason is that this is one out of a huge number of possible patterns of activity. To get an idea of this number, let's do a rough estimation. Let's suppose that it is nearly impossible that any individual neuron will be firing 20 or more spikes, given the mean rates. Then, each neuron's activity can take 20 values (including 0). There are 9 neurons, and they are independent of each other, so the total number of patterns is $9^{20} = 1.2 \times 10^{19}$. If each of these patterns had been equally likely, each would have had a probability of $1/(1.2 \times 10^{19}) = 8.3 \times 10^{-20}$. Compared with this, the probability of the pattern we calculated above is actually very large! We conclude that it is expected that in an independent Poisson population, each pattern has a low probability, and the more neurons, the lower this probability. If the neurons were Poisson but not independent, fewer patterns would be possible, and the probability of a given pattern would tend to be higher.

The patterns in Figure 4 make clear that an individual pattern of activity is roughly shaped like the Gaussian tuning curve, but with a different x-axis: preferred stimulus as opposed to stimulus. In fact, if one were to average over many patterns of activity elicited by the same stimulus s, one would get a mean activity described by the set of numbers $f_i(s)$ for $i = 1,\ldots, N$. Looking back at Equation 1, we see that we can plot $f_i(s)$ as a function of the preferred stimulus s_i, with s being fixed. This is a Gaussian shape, just like $f_i(s)$ was a Gaussian shape as a function of s, with s_i being fixed. In other words, the mean population response to one repeated stimulus has the same shape as the mean response of a single neuron as a function of the stimulus. This is true for any tuning curve in which s_i and s can be swapped without affecting the functional form, such as Von Mises curves (Eq. 1).

Sample Problems

Problem 1

Why are monotonic tuning curves always over magnitude-type variables such as width, and not over circular variables such as orientation?

Problem 2

Are the following statements true or false? Explain.

a) The closer a stimulus is to the preferred stimulus of a Poisson neuron, the lower is the response variance of this neuron when the stimulus is presented repeatedly.

b) When neurons have similar and equally spaced tuning curves, then the mean population pattern of activity in response to a stimulus has the same width as the tuning curve.

c) When neurons have similar and equally spaced tuning curves, then the neural posterior has the same width as the tuning curve.

d) The variance of a single neuron responding to a stimulus can be determined from the value of its tuning curve at that stimulus value.

e) In any population, the variability of population activity is known if one knows the variability of each single neuron.

Problem 3

We assume a population of 9 independent Poisson neurons with Gaussian tuning curves and preferred orientations from –40 to 40 in steps of 10. The tuning curve parameters have values $g = 10$, $b = 0$, and $\sigma_{tc} = 20$. A stimulus $s = 0$ is presented to this population. What is the probability that all neurons stay silent?

Problem 4: Properties of the Poisson distribution

a) Prove that Equation 4 implies that the mean value of r_i is indeed $f_i(s)$. Recall that the mean of r_i is defined as

$$\langle r_i \rangle = \sum_{r_i=0}^{\infty} r_i \, p(\mathrm{r}_i \mid \lambda_i).$$

You will also need to use a variant of Equation 4.

b) Prove that Equation 4 implies that the variance of a Poisson process is equal to its mean. Recall that the variance of r_i can be written as

$$\mathrm{Var}\,(r_i) = \langle r_i^2 \rangle - \langle r_i \rangle^2 = \sum_{r_i=0}^{\infty} r_i^2 \, p(\mathrm{r}_i \mid \lambda_i) - \langle r_i \rangle^2$$

Problem 5

In a population of independent Poisson neurons with Gaussian tuning curves, examine the claim that $\Sigma_i f_i(s)$ is more or less independent of s.

NOTES

Problem 6

Prove that independent Poisson variability, Equation 7, is a special case of Poisson-like variability:

a) Rewrite Equation 7 in the form of Equation 4

b) Verify that Equation 4 holds for $\boldsymbol{h}(s)$ found in part (a).

Advanced Problem 7

In a sequence of R independent events with two possible outcomes, the probability of having one of both outcomes appear r times is described by the binomial distribution,

$$p(\boldsymbol{r}) = \binom{R}{r} \left(\frac{\lambda}{R}\right)^r \left(1 - \frac{\lambda}{R}\right)^{R-r}.$$

Here, $\binom{R}{r}$ is the binomial coefficient, $\frac{R!}{r!\,(R-r)!}$. Prove that the Poisson distribution is a good approximation of the binomial distribution if the sequence is long (R large) and the probability of the outcome of interest (λ/R) is small.

Advanced Problem 8

Prove that for large means, a Poisson distribution resembles a Gaussian distribution with variance equal to the mean.

Advanced Problem 9

Show that in the limit of large κ, the Von Mises function (Eq. 1) becomes the Gaussian function. Hint: in this limit, the Von Mises function becomes very strongly peaked around s_i, and we can use the Taylor series expansion $\cos(x) \approx 1 - x^2/2$.

Problem 10

Consider a population of neurons with known tuning curves, subject to independent noise.

a) If the noise is drawn from a normal distribution with fixed variance, prove that the maximum-likelihood decoder is equivalent to the template-matching decoder.

b) If the noise follows a Poisson distribution, tuning curves are Gaussian with zero baseline, and $\sum_i f_i(s)$ is independent of s, to which decoder is the maximum-likelihood decoder equivalent? Prove your answer.

Problem 11

Show that when neural variability is independent, and Poisson tuning curves are Von Mises function with zero baseline, and $\sum_i f_i(s)$ is independent of s, the maximum-likelihood decoder is equivalent to the population vector.

Problem 12

A Bayesian observer decodes a stimulus s from a neural population under a cost function, $C(\hat{s},s)$.

a) Prove that if the cost function is the squared error, the Bayesian estimate is the mean of the posterior distribution.

b) Derive the Bayesian estimate if the cost function is the absolute error, $C(\hat{s},s) = |\hat{s} - s|$.

c) What is the cost function corresponding to the maximum–a posteriori decoder?

Problem 13

In a discrimination task, an observer decides on each trial whether a stimulus has value s_1 or s_2. The stimulus elicits activity $\boldsymbol{r}$ in a neural population with tuning curves $f_i(s)$. Assume that $\boldsymbol{r}$ is drawn from an independent Poisson distribution and that $\sum_i f_i(s)$ is independent of s.

a) Calculate the log likelihood ratio and prove that the maximum-likelihood decision is based on the sign (positive or negative) of the inner product of $\boldsymbol{r}$ with a vector $\boldsymbol{w}$. Find an expression for the ith component w_i in terms of the numbers $f_i(s_1)$ and $f_i(s_2)$.

b) What does the absolute value of $\boldsymbol{w} \times \boldsymbol{r}$ mean to the observer? Explain.

c) Compute the mean and variance of $\boldsymbol{w} \times \boldsymbol{r}$ from part (a) when $\boldsymbol{r}$ is generated by s_1, and when $\boldsymbol{r}$ is generated by s_2. "Sensitivity" or discriminability is defined as the difference between both means divided by the square root of the mean of both variances. Find an expression for discriminability in terms of the sets of numbers $f_i(s_1)$ and $f_i(s_2)$ (for all i).

NOTES

Lab Problems

Problem 14: Simulating a Poisson process

a) Define 1000 time points. At each time point, determine whether a spike is fired by generating a random number that leads to a "yes" probability of 0.0032 (this corresponds to a mean of 3.2 spikes over all 1000 time points). Count the total number of spikes generated. Repeat for 10,000 trials. Plot a histogram of spike count and compare with Figure 4*a*. Compute the Fano factor.

b) Repeat for a mean of 9.5 spikes. Compare the resulting histograms with Figure 4*b*.

c) If your simulation software has a built-in command to randomly generate numbers according to a Poisson distribution (e.g., poissrnd in MATLAB), repeat steps (a) and (b) using this command.

d) A property of a Poisson process is that the time between two subsequent spikes (interspike interval, denoted here Δt) follows an exponential distribution: $p(\Delta t) = \exp(-\Delta t/\lambda)/\lambda$, where λ is the mean of the Poisson process. Verify this by plotting the histogram of interspike intervals across all Poisson spike trains you generated in (a) and comparing it with the exponential distribution.

Problem 15

DatasetReal (downloadable at http://klab.smpp.northwestern.edu/wiki/images/1/17/DatasetReal.mat) contains recordings from 35 neurons in the primary motor cortex and ~200 trials, roughly half of which were recorded while the monkey was moving left, while the others were recorded while the monkey was moving right (courtesy of the Miller Lab). Let's assume a Gaussian distribution of spike counts, given the direction of movement.

a) Calculate for each neuron the average firing rates for left and right movement and the associated SDs.

b) Do all neurons have similar average firing rates for left and right movements? Which of the neurons exhibit a significant difference between left and right movement?

c) What would be a good measure for strength of tuning of a neuron? Which neuron has the strongest tuning to direction?

d) If you decoded movement direction based on just this neuron, how well would you do on average?

e) If you combined data from all neurons using a naive Bayesian approach, how good could you be at solving the problem?

f) Is this a difficult problem? Could it have real-world relevance? Can you think of an application of naive Bayesian decoding that is more exciting?

$$p(right \mid spikes) \propto \frac{1}{Z} p(right) \prod \frac{1}{\sigma_{right,i}} e^{(spikes_i - \mu_{right,i})^2/(2\sigma^2_{right,i})}$$

This same approach of assuming that all cues are independent, even when they are not, is used in many domains of machine learning. A naive Bayesian approach is often used to solve real classification problems and is, for certain problems, a competitive machine learning technique. It is particularly strong when there are very little available data.

Acknowledgments

We refer readers interested in detailed neuron models to the text *Spiking Neuron Models* by Wulfram Gerstner and Werner Kistler. Historical video recordings of Hubel and Wiesel presenting stimuli to cat LGN and V1 neurons can be found online. This chapter was excerpted with permission from Konrad Kording, Bayesian Modeling of Perception and Action, Chapter 9, Oxford University Press.

References

Averbeck BB, Latham PE, Pouget A (2006) Neural correlations, population coding, and computation. Nat Rev Neurosci 7:358–366.

Deneve S, Latham P, Pouget A (1999) Reading population codes: a neural implementation of ideal observers. Nat Neurosci 2:740–745.

Foldiak P (1993) The 'ideal homunculus': statistical inference from neural population responses. In: Computation and neural systems (Eeckman F, Bower J, eds), pp 55–60. Norwell, MA: Kluwer Academic Publishers.

Georgopoulos A, Kalaska J, Caminiti R, Massey JT (1982) On the relations between the direction of two-dimensional arm movements and cell discharge in primate motor cortex. J Neurosci 2:1527–1537.

NOTES

Gerstner W, Kistler WM (2002) Spiking neuron models: single neurons, populations, plasticity. Cambridge, UK: Cambridge University Press.

Gur M, Snodderly DM (2005) High response reliability of neurons in primary visual cortex (V1) of alert, trained monkeys. Cereb Cortex 16:888–895.

Hoyer PO, Hyvärinen A (2003) Interpreting neural response variability as Monte Carlo sampling of the posterior. In: Proceedings of the 15th International Conference on Neural Information Processing Systems, pp 293–300. Cambridge, MA: MIT Press.

Hubel DH, Wiesel TN (1959) Receptive fields of single neurones in the cat's striate cortex. J Physiol (Lond) 148:574–591.

Ma WJ, Beck JM, Latham PE, Pouget A (2006) Bayesian inference with probabilistic population codes. Nat Neurosci 9:1432–1438.

Pouget A, Dayan P, Zemel RS (2003) Inference and computation with population codes. Ann Rev Neurosci 26:381–410.

Pruett JR Jr, Sinclair RJ, Burton H (2000) Response patterns in second somatosensory cortex (SII) of awake monkeys to passively applied tactile gratings. J Neurophysiol 84:780–797.

Sanger T (1996) Probability density estimation for the interpretation of neural population codes. J Neurophysiol 76:2790–2793.

Shapley R, Hawken M, Ringach DL (2003) Dynamics of orientation selectivity in the primary visual cortex and the importance of cortical inhibition. Neuron 38:689–699.

Theunissen FE, Miller JP (1991) Representation of sensory information in the cricket cercal sensory system. II. Information theoretic calculation of system accuracy and optimal tuning-curve widths of four primary interneurons. J Neurophysiol 66:1690–1703.

Tolhurst D, Movshon J, Dean A (1982) The statistical reliability of signals in single neurons in cat and monkey visual cortex. Vision Res 23:775–785.

Zhang K, Ginzburg I, McNaughton B, Sejnowski T (1998) Interpreting neuronal population activity by reconstruction: unified framework with application to hippocampal place cells. J Neurophysiol 79:1017–1044.

Characterizing and Correlating Spike Trains

Pascal Wallisch, PhD, and Erik Lee Nylen, PhD

Center for Neural Science
New York University
New York, New York

NOTES

Introduction

In this chapter, we introduce the standard spike-wrangling techniques that any neuroscientist should be familiar with. Specifically, we illustrate how to build programs that can analyze and display the information contained in spike trains (Rieke 1999). We start by showing how to represent and graph the spiking activity of a single neuron in a single train and then build toward creating a raster plot that depicts the spiking information of a single neuron over multiple trials. From there, we develop another useful representation of the information in a spike train: the peristimulus time histogram (PSTH). We also cover how to compute latency to first spike and how to represent this joint information as a heat map. Throughout, we use this simple but highly relevant example from neuroscience to illustrate key concepts in programming, such as the perils of hard coding or the use of for loops.

We will show how to characterize spike trains in several relevant languages, such as MATLAB and Python (Peters 2004), and introduce the canonical data analysis cascade, which is—to the best of our knowledge—the most efficient way to organize the processing of large-scale data analysis projects.

Neurons

Neurons are peculiarly social. Some neurons are veritable chatterboxes, some are rather quiet, and some are even trying to silence others. Some neurons prefer to talk only to close neighbors, whereas others send messages to comrades in far distant regions. In this sense, all neuroscience is necessarily social neuroscience. Of course, neurons don't communicate with each other via spoken words. Rather, they use action potentials—also known as voltage "spikes"—as their universal means of long-distance communication. Every neuron sends spikes in its own idiosyncratic fashion, and every neuron uses its dendritic arbor to receive signals from other neurons in turn. The interface points between neurons are known as "synapses." A neuron may have many points (anywhere from 1 for neurons in the retina to >100,000 for neurons in the cerebellum) of communication—synapses—with other neurons. Although spikes are not exchanged directly (the signal crossing the synapse is chemical in nature in almost all synapses), it is the voltage spikes that drive the neural communication machinery. Specifically, spikes traveling down the axon of the presynaptic neuron trigger the chemical action in the synapse that enacts further voltage changes, and perhaps more spikes in the postsynaptic neuron.

Neurons use spikes as their preferred medium of communication. It is fair to say that a major challenge faced by contemporary neuroscientists is to elucidate the meaning of these spikes. Put differently, the neuroscience community is trying to "crack the neural code." Our starting point in this pursuit is the signal itself—the spikes. Because of their nature as all-or-none events, we will represent the occurrence of spikes over time as numbers, specifically zeroes for "no spike" or ones for "spike." So consider the following list of numbers:

[0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0]

By itself, this list of numbers is meaningless. However, let's assume that we have a neuron in a dish, that the neuron is alive, that it is capable of sending spikes, that this neuron will generally not send any spikes in the dark, and that this neuron will send some amount of spikes if you shine a green light (at a wavelength of 550 nm) on it. Let's also assume that we have a recording electrode near the point where this neuron sends spikes, and that our electrode has the fancy ability of telling our computer whether or not the neuron is spiking over time, as captured in the vector with 0s and 1s above. This scenario is schematized in Figure 1.

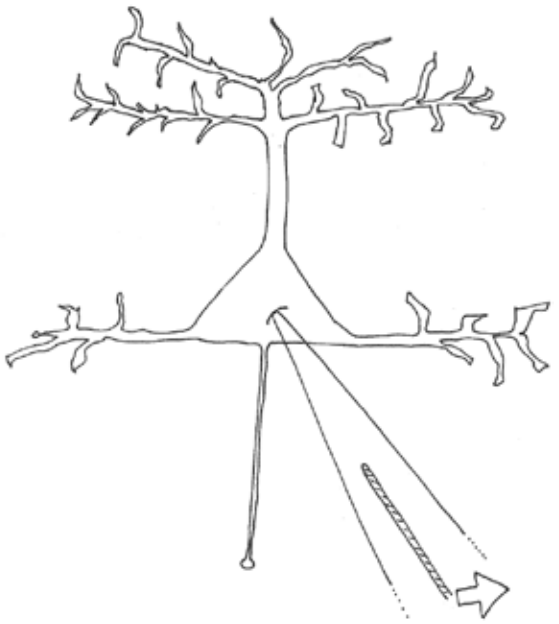

Figure 1. Extracellular electrode recording voltage spikes from an individual neuron in a dish.

NOTES

Without knowing yet why the spikes occurred, we can make a couple remarks about this list of numbers representing spikes. First, we know how many spikes are in the list:

Pseudocode	Sum up the numbers in the vector
Python	`>>> sum([0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0])` `4`
MATLAB	`>> sum([0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0])` `ans = 4`

But you didn't need Python or MATLAB to tell you that there are four 1s in this list, because you can see that immediately. It is, however, convenient for us to measure how long the list is using the function *len* in Python and *length* in MATLAB:

Pseudocode	Count the number of elements in the longest dimension of the vector
Python	`>>> len([0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0])` `21`
MATLAB	`>> length([0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0])` `ans = 21`

This means that the list of numbers has 21 entries or elements. Recall that each number in the list represents whether or not the neuron sent a spike at that time, that the 0 on the far left of the list represents *time* = 0, and that each successive number on the list represents whether or not the neuron sends a spike at that time. We could say:

We interpret this to mean that at times 0, 1, 2, 3, 4, 5, 6, 7, and 8, the neuron is not spiking. At time 9 it spikes, at time 10 it is quiet, at time 11 it spikes, at time 12 it is quiet, at time 13 it spikes, at times 14 through 16 is it quiet, and then at time 17 it spikes one last time before being quiet again.

We said earlier that this neuron tends to spike if it is illuminated with green light, but not in darkness. What we are simulating here is a tool known as "optogenetics," in which neurons will actually increase their activity in response to light (Boyden et al., 2005; Tye and Deisseroth, 2012).

So let's indicate the time points during which such a green light was on in green italics, leaving the rest of the time points black (representing times during which the light was off):

[0,1,2,3,4,*5,6,7,8,9,10,11,12,13*,14,15,16,17,
18,19,20]

Let us assume now that each time point above is in units of milliseconds. What this means is that we started recording from the neuron at time 0 when the light was off. After 4 ms of recording, on the fifth millisecond, the green light was turned on. The light then stays on for 9 ms—through the 13th millisecond—before shutting off.

Pseudocode	Create a list of 21 successive integers representing time and align it with the 21 neuron states

Python	`range(21), [0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0]`
	`([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20],`
	`[0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0])`

MATLAB	`>> [linspace(0,20,21); [0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0]]`
	`ans =`
	`0 1 2 3 4 5 6 7 8 9 10`
	`0 0 0 0 0 0 0 0 0 1 0`
	`11 12 13 14 15 16 17 18 19 20`
	`1 0 1 0 0 0 1 0 0 0`

NOTES

With this knowledge of the stimulus conditions, we can determine a characteristic feature of this neuron: its "first spike latency to stimulus." This parameter is generally used by neuroscientists to establish how "fast" or "ready to spike" any given neuron is.

We now know just enough about the stimulus and the neuron to be good neuroscientists and form a hypothesis. Let's hypothesize that the neuron always fires a spike 4 ms after a light is turned on.

Let's put the string of 0s and 1s into a variable now:

Pseudocode	Assign data to variable spikeTrain
Python	`>>> spikeTrain = [0,0,0,0,0,0,0, 0,0,1,0,1,0,1,0,0,0,1,0,0,0]`
MATLAB	`>> spikeTrain = [0,0,0,0,0,0,0, 0,0,1,0,1,0,1,0,0,0,1,0,0,0];`

With the spikes now stored in spikeTrain, we can pull out spikes at different time points. That is, we can ask, say, at time $t = 5$, when the light is first turned on, is the neuron spiking?

Pseudocode	Output the contents of variable spikeTrain at the position corresponding to $t = 5$
Python	`>>> spikeTrain[5]` `0`
MATLAB	`>> spikeTrain(6)` `ans = 0`

The answer is a resounding no. But what about if we want to know if the cell spikes at any time after $t = 5$?

Pseudocode	Output all elements of spikeTrain after the position corresponding to $t = 5$
Python	`>>> spikeTrain[5:]` `[0,0,0,0,1,0,1,0,1,0,0,0, 1,0,0,0]`
MATLAB	`>> spikeTrain(6:end)` `ans = 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0`

Note that here we see some subtle but critical differences in Python versus MATLAB. First, in MATLAB, the first element of a vector is element "1," whereas the corresponding first element in Python is "0." So to access the same element (here the one at time 5), we have to add 1 to the MATLAB index. Second, the colon operator : returns all elements from a starting point until an endpoint. Python assumes you want all elements until the end of the vector if no endpoint is specified, whereas the corresponding MATLAB command to specify "all elements until the end of the vector" is "end." Finally, note that the MATLAB command uses parentheses, whereas the Python command uses square brackets.

This output—in Python or MATLAB—represents the neuron's spiking activity after the green light turned on. If the first output value in the list were a 1, then the neuron's latency to first spike would be 0 ms, i.e., the neuron's response would be coincident with the light turning on. But things rarely happen instantly in biology, let alone neuroscience. Rather, whatever makes our neuron spike in response to light takes some time to flip some internal switches before the spike occurs. The

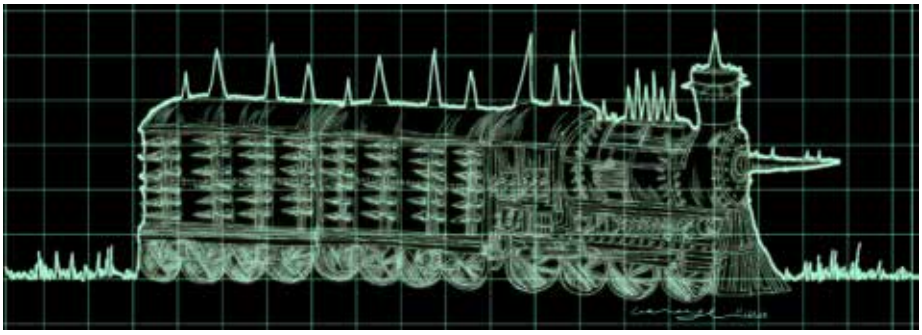

Figure 2. A spike train, as seen on an oscilloscope. Voltage spikes recorded from an individual neuron are plotted over time.

NOTES

record of the successive spiking of a neuron over time is called the "spike train" (Fig. 2), and various measures to characterize the spike train have been proposed.

The time it takes for a neuron to be responsive (that is, for a neuron to spike in response to a stimulus—in our case, to the light) is known as the "response latency." Different ways exist to measure response latency, but for a single light pulse and a single recording from a neuron that spikes relatively infrequently (or sparsely), the time it takes for the first spike to occur is a good measure. So let's calculate the latency to first spike by typing:

Pseudocode	Find the first value in the elements of spikeTrain that matches 1
Python	`>>> spikeTrain[5:].index(1)` `4`

MATLAB	`>> find(spikeTrain(6:end)==1);`
	`ans(1)-1`
	`ans = 4`

Note that in the MATLAB version of this code, we concatenate two commands in the same line by the use of the semicolon operator. We have to bring the result into the same time base as the Python code by adding 1 to the start point and then subtracting it again from the result. If we had started with calling the first element time "1," we would have had to subtract 1 from Python to get the right index and then add 1 to the answer.

Here we took our list of values in the variable spikeTrain that occurred after a time of 5 ms and used the Python function *index* (*find* in MATLAB) to locate the first value in this variable that represents the spike train that matches *1* (representing a spike). We can make this more flexible by giving the light onset time *5* a variable name and the spike indicator value *1* a variable name. Generally speaking, it is a bad idea to hard-code values that are given to functions explicitly. It is almost always better to use variables because code that uses variables is much easier to maintain.

Pseudocode	Assign values to variables, then pass those to the function to avoid hard-coding

Python	`>>> lightOnsetTime = 5`
	`>>> spikeValue = 1`
	`>>> spikeTrain[lightOnset Time:].index(spikeValue)`
	`4`

MATLAB	`>> lightOnsetTime = 5;`
	`>> spikeValue = 1;`
	`>> mShift = 1;`
	`>> find(spikeTrain(lightOnset Time+mShift:end)==spikeValue);`
	`ans(1)-mShift`
	`ans = 4`

This version, without the hard-coding, also makes it clearer what is going on in the MATLAB case. Because of MATLAB's indexing conventions, we know that MATLAB indices are always shifted by 1 relative to Python. So we allocate this to a constant ("mShift"), add it to the lightOnsetTime, and then subtract it again from the result. Note that the output is the same, which is reassuring, as our results should depend on the properties of the neuron we study, not the software we use to analyze the data we record from it.

Again, in order to find the latency to first spike, we take the variable *spikeTrain*, which contains all of the 1s and 0s that represent the presence or absence of spiking at a given time bin, and look only at the times after the light onset. We then look for the first time that the vector after light onset contains a 1. Technically speaking, this command returns the number of bins between light onset and the first spike, but because we know that the bin width is 1 ms, we can interpret the result as a time: the latency is 4 ms. In order to be able to reuse a result that we just calculated in this fashion, we should assign the results of a command to a variable. In this case, we will call it "latencyToFirstSpike." Generally speaking, it is advisable to use variable names that are meaningful, as that makes code much more readable.

Pseudocode	Calculate latency to first spike and assign it to a variable with meaningful name

Python	`>>> latencyToFirstSpike = spikeTrain[lightOnsetTime:].index(spikeValue)`
	`>>> print latencyToFirstSpike`
	`4`

MATLAB	`>> temp = find(spikeTrain(light OnsetTime+mShift:end)== spikeValue);`
	`>> latencyToFirstSpike = temp(1)-mShift`
	`latencyToFirstSpike = 4`

Note that in the example above, the MATLAB computations are done in two steps. We first declare a temporary variable temp that finds *all* instances in which spikes occur after light onset. Then in a second step, we find the index of the first element and correct for the MATLAB indexing shift by subtracting 1 and assigning that to the variable latencyToFirstSpike.

Python has its own set of idiosyncrasies—these brief calculations in MATLAB involve only regular parentheses. In Python, square brackets [ ] are used when specifying the range of values within a list of numbers. In contrast, regular parentheses ( ) are used to invoke functions with the particular parameters within the parentheses as inputs. In this case, we pass the variable spikeValue to the function *index*,

NOTES

which is a built-in function—*index* is not the only such function. Python has many functions that we'll be using, and when we do, we'll use parentheses to give them values to operate on.

Now we have our estimate of the neuron's latency (4 ms). As the readers of your research papers are likely to be primates, and primates are predominantly visually guided animals, we should make a plot to illustrate the spiking activity of the neuron (Fig. 3).

Figure 3. Cartoons can efficiently illustrate important principles.

To plot the spiking activity, we need to know the time of every spike in the list. People sometimes call these the "spike timestamps," but we'll just call them "spikeTimes":

`Pseudocode`	Find and then output the times at which the neuron spikes

`Python`	`>>> spikeTimes = [i for i,x in enumerate(spikeTrain) if x==1]`
	`>>> print spikeTimes`
	`[9, 11, 13, 17]`

`MATLAB`	>> spikeTimes = find(spikeTrain==1)-mShift spikeTimes = 9 11 13 17

The Python part of this is a whopping nest of code! Let's untangle it a bit. First, see how we put the list of numbers **`spikeTrain`** (a bunch of 1s and 0s) into the function **`enumerate`**. Don't bother typing **`enumerate(spikeTrain)`** into your command line or trying to print it yet. The function **`enumerate(spikeTrain)`** cycles through the list **`spikeTrain`** and keeps track of the index of each element in **`spikeTrain`**.

The middle part of the code **`i,x`** in **`enumerate(spikeTrain)`** means that we will be going through each element in **`spikeTrain`** and naming each element along the way "**`x`**," and wherever "**`x`**" is in the list spikeTrain, we'll call that location "**`i`**."

A diagram might help:

i will successively be each element in
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
x will successively be each element in
[0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0]

Let's look at that line of code again:

```
>>> spikeTimes = [i for i,x in
enumerate(spikeTrain) if x==1]
```

Note that Python or MATLAB makes use of double equals signs to test for the equality of two values. It is worth noting that a single equals sign is an assignment operator, assigning whatever is on the right (usually the result of a computation) to the left (usually a variable).

We now understand that the line means to give us the indices of *i* where *x* = 1.

i is
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
and *x* is [0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0]

As you can see, this is the case for the indices *i* of 9, 11, 13, and 17. And now, given this line, Python can see it too and return it to you (and put it into the variable "spikeTimes" as well).

We can conceive of this problem in reverse, too:

If *x* = 1, where *x* is each element in
[0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0]
gives us the corresponding value in *i*:
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

Where we underlined the *i*s and *x*s, where *x* = 1.

That's a lot of work—the equivalent MATLAB code seems much simpler—and would be simpler yet if we didn't have to bring the result into a format that conforms to Python's zero-indexing conventions—but that is a small price to pay to use a real programming language, like hackers do. After all, social life is all about tribal signaling.

But let's not lose sight of the fact that the point of doing this was so we can graph it. To graph (or "plot," as we will call it from now on) something in Python, we need to import a library of functions designed for this purpose. This library is called "*matplotlib*," a

NOTES

library that gives us all the plotting tools we need at this point.

Pseudocode	Import library of plotting functions
Python	`>>> import matplotlib.pyplot as plt`
MATLAB	No equivalent. These plotting functions already come with MATLAB

This means we've loaded the function *pyplot*, a function of the library *matplotlib*, but when we loaded it we gave it a shorthand (*plt*) so we can refer to it more easily—and with a nickname of our choice. You'll find that the strategy of importing functions as two- or three-letter shorthands will save you a lot of typing. We recommend it.

We will now create a figure to work on:

Pseudocode	Open figure
Python	`>>> fig = plt.figure()`
MATLAB	`>> figure`

This establishes a frame to work within. Within this frame, we can have from one to many smaller windows, or "subplots." For now, we're plotting a small set of spikes, so we need only one subplot.

Pseudocode	Place a subplot (or axes) into the figure
Python	`>>> ax=plt.subplot(111)`
MATLAB	No equivalent. If there is only one pair of axes, it will take up the entire figure by default. You could still type subplot(1,1,1) and place axes, but this step is not necessary.

What the *111* means will be a little clearer once we have multiple subplots in a figure. For now, just know that if you want only one plot in your figure, you are saying that you want one large subplot, and therefore use *subplot(111)*. We call this subplot "ax" in honor of the axes of a Cartesian plot, which is the kind we want here.

To plot the spike times, we'll use the common visualization method of a spike raster plot. In such a plot, each spike is represented as a vertical line (at the time when it occurred, with time on the *x*-axis).

Pseudocode	Plot vertical lines at the times when a spike occurred, then show the figure
Python	`>>> plt.vlines(spikeTimes, 0, 1)` `>>> plt.show()`
MATLAB	`>> line(repmat(spikeTimes,2,1), repmat([0; ],1,4),'color','k')` `shg`

Let's look at the plot this code produces in order to discuss what it means:

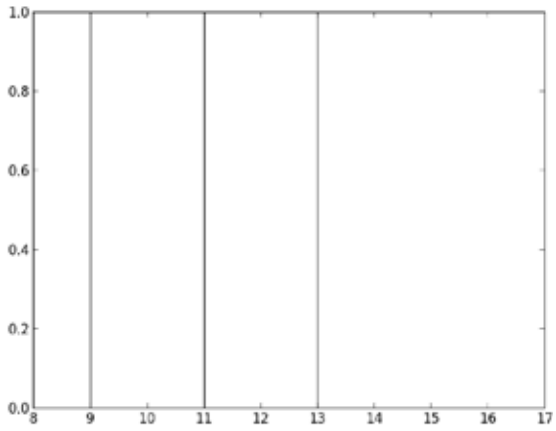

Figure 4. A bare-bones raster plot of a single trial.

Voilà! We have our first plot of neuroscience data. Shockingly, we have neither axis labels nor units—yet. Before we rectify this problem, let's discuss how the figure was brought about by the commands directly above.

In Python, we invoke the function *vlines* (a subfunction of the plotting package we called *plt*) by putting a period between them. This is generally the way to invoke subfunctions of imported packages. We then use parentheses to give values to this function. The three arguments we pass to the function *vlines* are: first a list of spike times, second the minimum value of the vertical lines—0, and third the maximum value of the vertical lines—1.

In MATLAB, there is no function for vertical lines specifically, so we use the general purpose function *line*. It takes matrices to specify *x* and *y* coordinates and plots one line for each column where row values indicate start and end positions of the lines. We have to create these matrices first, so we create two 2 × 4 matrices with the *repmat* function.

NOTES

We also have to set the color of the lines to black (the default MATLAB color is blue) in order to match the plot produced by Python. The function *shg* shows the current graph.

This plot illustrates when the neuron spikes, but doesn't contain any information about the light stimulus yet. Let's make the time during which the light is on a shaded green:

Pseudocode	Add a shaded green rectangle from times 5 to 14
Python	`In : plt.axvspan(5,14,alpha=0.1, color='g')`
MATLAB	`rectangle ('Position',[5, 0,9,1],'FaceColor',[0.7 1 0.7],'linestyle','none')`

This creates a shaded green box that spans the figure vertically and is bounded horizontally at 5 and 14.

Python: The *alpha* value makes the box transparent (smaller *alpha* values make the color appear lighter).

MATLAB: Specifies the color of the box by giving it an RGB triplet, in this case, a light green.

Let's now specify the range of times our *x*-axis should span by invoking the *xlim* function so that we can see times during which the neuron did not fire, i.e., before visual stimulation:

Pseudocode	Setting the range of the *x*-axis to include the entire time interval of interest
Python	`In : plt.xlim([0,20])`
MATLAB	`xlim([0 20])`

Before showing this figure to anyone, we strongly recommend adding a label to the *x*-axis and a title to the figure.

Pseudocode	Add meaningful axis and figure labels, which is critical in science
Python	`>>> plt.title('this neuron spikes in response to a single light stimulus')` `>>> plt.xlabel('time (in milliseconds)')`
MATLAB	`>> title('this neuron spikes in response to a single light stimulus')` `>> xlabel('time (in milliseconds)')`

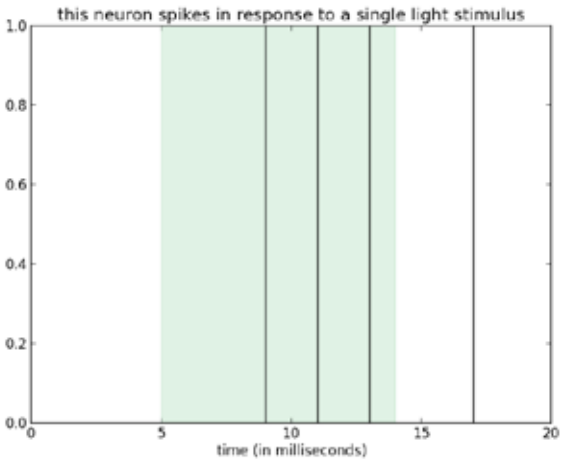

Figure 5. A raster plot of a single trial with axis labels, title, and stimulation condition.

Because of the noise inherent in the system as well as the measurements, data from a single trial are rarely sufficient to reach reliable conclusions about the connection between visual stimulation and the neural response.

What if there is some mechanism inside the neuron that causes it to spike highly unreliably? What if we are recording the signals of this neuron in a noisy environment? In theory, we would need to stimulate the neuron an infinite number of times and collect an infinite number of responses in order to be really sure. But most people are not theorists, living in a non-platonic world. As such, we neuroscientists have to make do with less than infinite amounts of data. Say we had just enough funding to allow us to collect data from 10 spike trains and 10 identical (green) light stimuli. These data are contained in the *tenSpikeTrains* variable.

NOTES

Pseudocode	Representing the 10 spike trains in the respective formats
Python	>>> tenSpikeTrains = [[0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0],[0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,1,0,0,0],[0,1,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0],[0,0,0,0,0,0,0,0,0,1,1,0,1,0,0,0,0,0,1,0,0],[0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,1,0],[0,0,0,0,0,0,0,0,1,1,1,0,0,1,1,0,0,1,1,0,0],[1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,0,0,0,0,0],[0,0,0,0,0,0,0,0,1,1,0,0,1,1,0,0,1,1,0,0,0],[0,0,0,0,0,0,1,0,0,1,0,1,0,0,0,0,0,1,1,0,0]
MATLAB analogous	>> tenSpikeTrains = {[0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0],[0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,1,0,0,0],[0,1,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0],[0,0,0,0,0,0,0,0,0,1,1,0,1,0,0,0,0,0,1,0,0],[0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,1,0],[0,0,0,0,0,0,0,0,1,1,1,0,0,1,1,0,0,1,1,0,0],[1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,0,0,0,0,0],[0,0,0,0,0,0,0,0,1,1,0,0,1,1,0,0,1,1,0,0,0],[0,0,0,0,0,0,1,0,0,1,0,1,0,0,0,0,0,1,1,0,0]
MATLAB suitable	>> tenSpikeTrains = [0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0; 0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0;0,1,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0;0,0,0,0,0,0,0,0,0,1,1,0,1,0,0,0,0,0,1,0,0;0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0;0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,1,0;0,0,0,0,0,0,0,0,1,1,1,0,0,1,1,0,0,1,1,0,0;1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,0,0,0,0,0;0,0,0,0,0,0,0,0,1,1,0,0,1,1,0,0,1,1,0,0,0;0,0,0,0,0,0,1,0,0,1,0,1,0,0,0,0,0,1,1,0,0]

Note that Python uses nested square brackets here. It appears, from the placement of the square brackets, that there are 10 lists (each inside square brackets) nested inside one all-encompassing list (note the double square brackets at the very beginning and end of *tenSpikeTrains*). In fact, this is the case, and *tenSpikeTrains* is technically a list of lists. We can represent these spike trains like that in MATLAB too, by using cells (the "MATLAB analogous" code), but the most suitable way to represent these data in MATLAB are as a 10 × 21 matrix ("MATLAB suitable"). In other words, we represent each spike train as entries in 21 consecutive columns, and each individual spike train as a separate row. So at every point in the MATLAB suitable code above, where there is a],[in Python, there is a semicolon in MATLAB. This works because the time base (21 bins with a width of 1 ms) is the same for each spike train. If this were not case, e.g., if there were missing data, using cells would be more apt. Matrices are ordered arrangements of numbers, and cells are ordered arrangements of matrices. So Cell:Matrix as Matrix:Number in MATLAB. Cells can accommodate matrices with different dimensionalities in each entry, so they are very helpful but make it harder to do some computations on them. In addition, they complicate the notation—note the curly braces { } that indicate we are dealing with cells.

Returning to the data, if we look at the first entry of *tenSpikeTrains*, we find:

Pseudocode	Return the contents of the first spike train
Python	>>> tenSpikeTrains[0] [0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0]
MATLAB analogous	>> tenSpikeTrains{1} ans = 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0
MATLAB suitable	>> tenSpikeTrains(1,:) ans = 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0

The first spike train is contained in the first row (MATLAB suitable) or first cell (MATLAB analogous) or the 0th list (Python). This is the flipside of Python zero indexing. It made sense that the first time of a time series would be time zero, but the first element of a list is still the first element, not the zeroth element. So each indexing convention (0 for Python, 1 for MATLAB) has use cases where it is more "natural," which is why both of them are still around.

NOTES

Regardless of implementation, the command doesn't return a single value, but a list. How many elements are in this master list?

Pseudocode	How many elements does the variable tenSpikeTrains have?
Python	`>>> len(tenSpikeTrains)` `10`
MATLAB analogous	`>> length(tenSpikeTrains)` `ans = 10`
MATLAB suitable	`>> size(tenSpikeTrains,1)` `ans = 10`

In the "MATLAB suitable" case, we have to count the number of rows, which is achieved by telling the *size* function to evaluate the first dimension (rows).

You might have expected much more than that (~200), but the *len* function in Python looks at how many things are inside the list much in the same way that a bundle of bananas counts as only one item at the grocery store when you are in the express lane. It simply returns the number of elements (in this case, lists) it contains. The same is true for the MATLAB analogous case—the cell has 10 entries, each of which is a matrix (and each of which represents an individual spike train). Accessing cell contents and doing computations on them is beyond the scope of this chapter, so we'll continue solely with the "suitable" case below.

In order to make a raster plot for the data from *all* the trials, we use a similar approach we used for plotting the data from a single trial, except we cycle through each list in the list. To do this, we will use a *for loop* (see below).[1]

As for Python, there are a few noteworthy things about this code, starting with the *for loop*. We know that the length of *tenSpikeTrains* is 10, so we can think of the *for loop* line of code as:

```
for trial in range(10)[2]:
```

We can also see that range(10) is:

```
>>> range(10)
[0,1,2,3,4,5,6,7,8,9]
```

That's right, range(10) is equal to the list [0,1,2,3,4,5,6,7,8,9]. In MATLAB, the command 1:size(tenSpikeTrains,1) achieves exactly the same result.

We can thus think of the *for loop* line of code in Python:

```
for trial in range(len(tenSpikeTrains)):
```

as equivalent to

```
for trial in [0,1,2,3,4,5,6,7,8,9]:
```

This means we are going to go to the next line of code after the *for loop* 10 times, and each time we go to the next line, the variable trial will increase to the next value in the list (starting with 0). It is called a *for loop* because it loops through a list of values. The principles of the *for loop* in Python are reflected in the *for loop* used in MATLAB:

```
for ii = 1:size(tenSpikeTrains,1)
```

So what's up the line after the *for loop*? There are the ellipses and some spaces before we get to the

Python	MATLAB (suitable)

`>>> fig = plt.figure()`
`>>> ax = plt.subplot(1,1,1)`
`>>> for trial in range(len(tenSpikeTrains)):`
`...  spikeTimes = [i for i,x in enumerate`
`(tenSpikeTrains[trial]) if x==1]`
`...  plt.vlines(spikeTimes,trial,trial+1)`
`>>> plt.axvspan(5,14,alpha=0.1,color='g')`
`>>> plt.xlim([0,20])`
`>>> plt.show()`
	`ax=figure`
	`rectangle('Position',[5,0,9,11],'FaceColor', ...`
	`[0.7 1 0.7],'linestyle','none')`
	`for ii = 1:size(tenSpikeTrains,1)`
	`spikeTimes = find(tenSpikeTrains(ii,:)==1)-`
	`mShift;`
	`line(repmat(spikeTimes,2,1),`
	`repmat([ii-0.5;ii+0.5],1,length`
	`(spikeTimes)),'color','k')`
	`end`
	`xlim([0 20])`
	`shg`

[1] *for loops* are your friend for easy plotting, and your enemy in heavy computation. If you are cycling through a network simulation or calculation, too many nested *for loops* will bring even the most powerful computer to its knees. As you begin to learn more programming tools, always be asking yourself if the code can be written without use of a *for loop*. But for now, using a *for loop* is fine, particularly because it is usually much clearer to understand what is going on when using loops.

[2] We put this line of code front and center to emphasize and talk about it, not for you to retype it into the command line or think that code usually sits centered on the page.

NOTES

spikeTimes line, which is very similar to how we got *spikeTimes* before. Ellipses are a placeholder for white space. In effect, they represent white space explicitly.

```
>>> for trial in range(len(tenSpikeTrains)):

  ... spikeTimes = [i for i,x in enumerate
      (tenSpikeTrains[trial]) if x==1]

  ... plt.vlines(spikeTimes,trial,trial+1)
```

It will keep creating indented lines and staying inside the grasp of the *for loop* until you hit return on a blank line. This indicates to Python that the *for loop* has ended. Notice how the line of code

```
>>> spikeTimes = [i for i,x in enumerate
    (tenSpikeTrains[trial]) if x==1]
```

is similar to how we got **spikeTimes** for a single trial. This time, though, we have our 1s and 0s within lists inside of the list **tenSpikeTrains**, so the variable trial, which iterates through the values 0 to 9, will call each of the 10 lists inside the list **tenSpikeTrains**. **spikeTimes** is now a temporary variable, and each time **trial** is updated in the for loop, and **spikeTimes** is reassigned to a new list of spike times.

Let's reiterate what **enumerate** does. This uniquely Python function allows us to build a for loop within a single line of code, as **enumerate** returns both each value in the list (like the for loop above) as well as the index of the value, where i is the index, and x is the value. Additionally, within this single line we condition returning the index i on the value x equaling 1. Thus, in this single line of code, we return the indices of the values equal to 1, in a Python-like fashion known as a "list comprehension."

The MATLAB code tries to achieve the same result, but note that the ellipses (...) mark the end of the line. They indicate that the command continues in the next line. So the use of ellipses is very different in Python versus MATLAB. In Python, indicating that the command continues in the next line is done via a backslash: "\."

For each trial in **tenSpikeTrains**, we plot vertical lines for the spikes, where the values *trial*, *trial* + *1*, … , *trial* + 9 are stacked sequentially and vertically—and where every "row" of the plot represents a trial.

Python or MATLAB has commenting conventions, which we are starting to use here—and from now on. Anything after the hashtag (#) in Python isn't read as programming code and is ignored by the computer, meaning it is not interpreting it as an instruction and thus not attempting to execute it. In general, we recommend writing a sentence at the beginning of a paragraph of code to explain what that paragraph is supposed to do—and in broad terms, how—and then to comment on critical pieces of code, e.g., note what a variable is supposed to contain. The analogous symbol in MATLAB is the percentage sign (%). Anything written after it is understood to be a comment.

Let us now add figure labels and formatting to get Figure 6:

Python

```
plt.ylim([0,10])
plt.title('this neuron spikes to repeated trials
of the same stimulus')
plt.xlabel('time (in milliseconds)')
plt.ylabel('trial number')
plt.yticks([x+0.5 for x in range(10)],[str(x+1)
for x in range(10)]) #1
```

MATLAB (suitable)

```
ylim([0.5 10.5])
title('this neuron spikes to repeated trials of
the same stimulus')
xlabel('time (in milliseconds)')
ylabel('Trial number')
set(gca,'Layer','top') %2
```

#1 Align the *y*-ticks to be in the middle of each row, while (x + 1) sets first trial to 1.

%2 Set the label axis to the top so that the green rectangle doesn't cover the axis.

A raster plot yields a snapshot of raw spike times elicited by the stimulus across trials. We can see that the first value we calculated, the first spike latency, seems to vary considerably between trials. On some trials the first spike occurs at 8 ms, on some trials the first spike happens at 9 ms, on one trial it comes at 6 ms, on a handful of trials it is 10 ms, and on one trial there is a spike 4 ms before the light even comes on. We can also see that the neuron seems to discharge a spike a few times with each light stimulus, and it often also "fires" after the stimulus has turned off. There are many ways to classify the spiking activity of neurons, some qualitative, some quantitative (such as our example, "first spike latency"). Qualitatively, we can ask ourselves if, once this neuron fires, does it keep firing? That is, is its ongoing activity tied to the stimulus?

NOTES

Pseudocode
Create figure and specify subplot Draw the stimulus presentation area green For each of the ten trials Extract the spike times from the spike train variables Plot each row in the raster plot as vertical lines Set the *x*-axis limits Set the *y*-axis limits Set the title Set the *x*-axis label Set the *y*-axis label

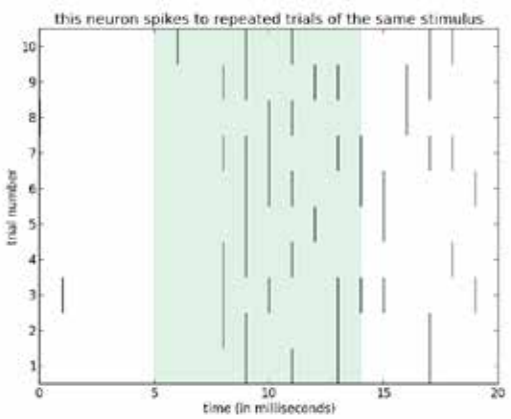

Figure 6. Raster plot of the neural response to 10 trials of a green light stimulus.

There are many ways to quantify this, but it is a good habit to take a look at the raw data in the form of a raster plot to first get a qualitative sense of the spiking characteristics of the neuron. For instance, the raster plot in Figure 6 seems to convey that the neuron responds to this particular light stimulus after 3–5 ms and that its activity is maintained to some degree after the stimulus has turned off.

Does this falsify our hypothesis that the neuron always spikes 4 ms after the light turns on? It certainly looks like it, since the spiking is not nearly as precise as we hoped it to be. But because we formed the hypothesis on a whim (based on the analysis of a single trial), we are free to change it. So let's say we hypothesize that the neuron fires "tonically at a rate of 500 spikes per second" (sp/s) to green light stimuli.

Let's unpack this statement. "Hypothesize that the neuron fires" is simple enough (the neuron discharges spikes), but we go on to make a prediction about the firing: that it fires "tonically." "Tonic firing" is spiking activity that is sustained or ongoing. This contrasts with "phasic" or "transient" firing, which is locked to the timing of the stimulus. We can come up with some metrics for quantifying the tonicity of the firing, but let's posit the qualitative hypothesis that it keeps firing and continue with the quantity "of 500 spikes per second." In our experiment, we didn't stimulate the neuron for an entire, continuous second, so we certainly won't have 500 spikes to count. However, the unit of spike rates is sp/s, even if the instantaneous firing rate is sustained for much less than a second, in the same way that you don't have to drive for an hour in order to travel at 100 miles per hour at some point in your ride.

To know how many spikes we ought to expect in our short interval, we simply have to solve for x in the algebraic equation where x is proportional to 500 spikes as our stimulus length (9 ms—we really should have planned the experiment better in order to get easier math!) is to 1 s:

$$(x \textit{ spikes})/(9 \textit{ milliseconds}) = (500 \textit{ spikes})/(1000 \textit{ milliseconds}) \quad (1)$$

Solving for x, this leads us to expect 4.5 spikes for the duration of our 9 ms stimulus. The tail end of the stated hypothesis above was "to green light stimuli," which we partly covered, and which we'll make more complex just when we start to get a better grasp of our results.

We thus need a way to visualize and condense the many stimulus trials and responses we recorded. We turn to the PSTH to visualize as a bar graph the spiking activity of the neuron over time, just before and after each (peri)stimulus. We also make use of multiple subplots within a single figure to compare the rasters to the PSTH. This is a standard depiction of neural data in early exploratory analysis.

An individual subplot is a subdivision of a figure. The code below indicates that the subplots will be arranged so that there are 2 rows and 1 column of subplots, and that we'll plot in the first of these. Note that although Python indexes lists by starting at 0, subplot indexing starts at 1. If this seems inconsistent, it is because it is.

Pseudocode	Create the figure window using plt (originally imported via matplotlib) Create the first of 2 "subplots."
Python	`>>> fig=plt.figure()` `>>> ax=plt.subplot(2,1,1)`
MATLAB	`>> fig = figure;` `>> ax = subplot(2,1,1);`

NOTES

We next create code for plotting (1) a subplot of spike rasters and (2) a PSTH based on that spiking activity. We highly advise that you comment so that (1) your "future me" can read and remember why you programmed something the way you did and what the variables stand for—which makes the code easier (or even possible) to maintain; and (2) so that other programmers can look at your code and have any chance of understanding what is going on.

In Figure 6, we created the variable **spikeTimes** for each **trial** and plotted those values right away, overwriting **spikeTimes** with each new trial.

Pseudocode	Looping through *trial 0, 1, …, 9*
	  Get the index (time) of each spike and append to allSpikeTimes
	  Plot vertical lines for each trial
	Add the vertically spanning green box
	Set the limits of the *y*-axis to 0 and 10
	Add a title, a *y*-axis label, and an *x*-axis label to this subplot
	#1 Customize the labels of the *y*-ticks

Python	`fig=plt.figure()`
	`ax=plt.subplot(2,1,1)`
	`for trial in range(len(tenSpikeTrains)):`
	`  spikeTimes = [i for i,x in enumerate(tenSpikeTrains[trial]) if x==1]`
	`  plt.vlines(spikeTimes,trial,trial+1)`
	`plt.axvspan(5,14,alpha=0.1,color='g')`
	`plt.ylim([0,10])`
	`plt.title('this neuron still spikes to repeated trials of the same stimulus')`
	`plt.xlabel('time (in milliseconds)')`
	`plt.ylabel('trial number')`
	`plt.yticks([x+0.5 for x in range(10)],[str(x) for x in range(10)]) #1`

MATLAB	`ax = subplot(2,1,1)`
	`rectangle('Position',[5,0,9,11],'FaceColor', [0.7 1 0.7],'linestyle','none')`
	`for ii = 1:size(tenSpikeTrains,1)`
	`  spikeTimes = find(tenSpikeTrains(ii,:)==1)-1`
	`  line(repmat(spikeTimes,2,1),repmat([ii-0.5; ii+0.5],1,length(spikeTimes)),`
	`  'color','k')`
	`end`
	`xlim([-0.5 20])`
	`ylim([0.5 10.5])`
	`title('this neuron still spikes to repeated trials of the same stimulus')`
	`xlabel('time (in milliseconds)')`
	`ylabel('Trial number')`
	`ax.YTick = [0:1:10]`
	`set(gca,'Layer','top')`

We next sum across our **tenSpikeTrains** to see the total number of spikes that occur across all trials, using the function **bar** in Python or MATLAB. This function gives us a bar plot of the spiking as a function of time.

We also save the figure. Note the extension **.png** at the end of the string. We could also specify .**pdf** or **.jpg** or a few other image types.

NOTES

Pseudocode	Now for the PSTH. We create our second subplot. Add the green background during stimulus time. Plot the bar plot #1 format bar (*x*-values, *y*-values, bar width) Add labels to the *x*- and *y*-axes of this subplot Save the figure Let's take a gander.

Python	`>>> ax=plt.subplot(2,1,2)`
	`>>> plt.axvspan(5,14,alpha=0.1,color='g')`
	`>>> ax.bar(range(21),np.sum(tenSpikeTrains,0),1) #1`
	`>>> plt.xlabel('time (in milliseconds)')`
	`>>> plt.ylabel('# of spike occurrences at this time')`
	`>>> plt.savefig('Figure with subplots of rasters and PSTH.png')`
	`>>> plt.show()`

MATLAB	`subplot(2,1,2)`
	`rectangle('Position',[5,0,9,8],'FaceColor',[0.7 1 0.7]...`
	`,'linestyle','none')`
	`hold on`
	`x=0:20;`
	`bar(x,sum(tenSpikeTrains));`
	`xlim([-0.5 20])`
	`ylim([0 8])`
	`xlabel('time (in milliseconds)')`
	`ylabel('# of spikes counted at this time')`

Let's put it all together:

Python	MATLAB

`# The Python way for Figure 7.`
`fig=plt.figure()`
`ax=plt.subplot(211)`
`for trial in range(len(tenSpikeTrains)):`
`  spikeTimes = [i for i,x in enumerate`
`  (tenSpikeTrains[trial]) if x==1]`
`  plt.vlines(spikeTimes,trial,trial+1)`
`plt.axvspan(5,14,alpha=0.1,color='g')`
`plt.ylim([0,10])`
`plt.title('this neuron still spikes to repeated trials of the same stimulus')`
`plt.xlabel('time (in milliseconds)')`
`plt.ylabel('trial number')`
`plt.yticks([x+0.5 for x in range(10)], [str(x+1) for x in range(10)])`
 	
`ax=plt.subplot(212)`
`plt.axvspan(5,14,alpha=0.1,color='g')`
`ax.bar(range(21),np.sum(tenSpikeTrains,0),1)`
`plt.xlabel('time (in milliseconds)')`
`plt.ylabel('# of spike occurrences at this time')`
`# End Python code for Figure 7`
	`% What does the analogous MATLAB code % look like?`
	`figure`
	`subplot(2,1,1)`
	`rectangle('Position',[5,0,9,11],'FaceColor',[0.7 1 0.7],'linestyle','none')`
	`for ii = 1:size(tenSpikeTrains,1)`
	`spikeTimes = find(tenSpikeTrains(ii,:)==1)-1`
	`line(repmat(spikeTimes,2,1),repmat([ii-0.5; ii+0.5],1,length(spikeTimes)),'color','k')`
	`end`
	`xlim([-0.5 20])`
	`ylim([0.5 10.5])`
	`title('this neuron still spikes to repeated trials of the same stimulus')`
	`xlabel('time (in milliseconds)')`
	`ylabel('Trial number')`
	`set(gca,'Layer','top')`
	`subplot(2,1,2)`
	`rectangle('Position',[5,0,9,8],'FaceColor',[0.7 1 0.7],'linestyle','none')`
	`hold on`
	`x=0:20;`
	`bar(x,sum(tenSpikeTrains));`
	`xlim([-0.5 20])`
	`ylim([0 8])`
	`xlabel('time (in milliseconds)')`
	`ylabel('# of spikes counted at this time')`
	`% End MATLAB code`

NOTES

Pseudocode
§ Begin English explanation of code for Figure 7 Create the figure area Specify that there are two rows and one column, and we'll start with the first Plot the vertical ticks lines Shade the area green to indicate the time of light stimulation Set the lower and upper bounds of the *y*-axis Set the title of the plot to 'this neuron still spikes to repeated trials of the same stimulus' Set the *x*-axis label to 'time (in milliseconds)' Set the *y*-axis label to 'trial number' Set the *y*-axis tick locations and labels Specify that we're making a subplot layout with two rows, one column, plotting in the second row Shade the stimulus area green Make a histogram of all the spike times with bins from 0 to 20 Set the *x*-axis and *y*-axis labels § End English explanation of Python code for Figure 7

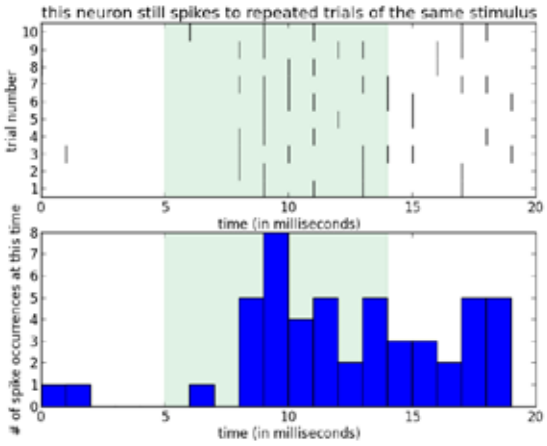

Figure 7. Neuron spikes to repeated trials of the same stimulus.

Let's go back to our hypothesis that the neuron fires "tonically at a rate of 500 spikes per second" to green light stimuli. We calculated that this would mean 4.5 spikes on average for the duration of our particular stimulus. But how can we infer this spike rate from the PSTH above? First, let us plot a new figure in which we scale the number of spike occurrences at each time by the number of stimulus trials (setting aside the rasters for a moment). This will show us, on average, how often the neuron spikes at each time point and give us an estimate of the spike probability for each point in time.

We start by creating the figure and omit the subplot line, noting that you don't need it for figures with single plots. Since we wanted to adjust the number of spikes for all the trials to form an estimate of spike probability, we will plot the **`mean`** of the spikes as a bar plot instead of a **`sum`**.

If the neuron spiked, on average, at a rate of 500 sp/s (every 1000 ms), then we might expect that for every millisecond there will be 0.5 spikes on average. Of course, we could not have performed this estimation with just a single trial, as one cannot count half a spike. By measuring repeated trials, we form a more robust estimate of the spike rate over time, with the prediction (from our hypothesis) that the neuron will maintain a spike probability of 0.5 during the time the stimulus is presented.

Let's draw a horizontal dashed black line (note the **`linestyle`** which can make it dotted or dashed, among other line types, and the color = *'k'* to denote *black*; we could have used *'r'* for *red*, or *'g'* for *green*, or *'b'* for *blue*) at the 0.5 spike probability threshold. Had we wanted a vertical line, we could have used the Python function **`plt.axvline`**.

Pseudocode	Create the figure Plot bar graph of the mean spikes Add horizontal line as a spike threshold

Python	`fig=plt.figure()`
	`plt.bar(range(21),`
	`np.mean(tenSpikeTrains,0),1)`
	`plt.axhline(y=0.5,xmin=0,xmax=`
	`20,linestyle='--',color='k')`

MATLAB	`figure`
	`bar(0:20,sum(tenSpikeTrains)./`
	`size(tenSpikeTrains,1));`
	`line(xlim,[0.5 0.5],'linestyle',`
	`'--','color','k')`

NOTES

Also label the axes and title, save the figure, and show it. Let's put it all together, using the simplified bar plot:

Python

```
# The Python way for Figure 8
fig=plt.figure()
plt.axvspan(5,14,alpha=0.1,color='g')
plt.bar(range(21), np.mean(tenSpikeTrains,0),1)
plt.axhline(y=0.5,xmin=0,xmax=20,linestyle='--
',color='k')
plt.title('spike probability given 10 stimulus
trials')
plt.xlabel('time (in milliseconds)')
plt.ylabel('probability of spike occurrences at
this time')
plt.savefig('Figure 8 normalized PSTH with cutoff.
png')
# End Python code for Figure 8
```

MATLAB

```
%What does the analogous MATLAB code look like?
figure
rectangle('Position',[5,0,9,11],'FaceColor',[0.7
1 0.7],'linestyle','none')
xlim([-0.5 20])
ylim([0 1])
hold on
bar(0:20,sum(tenSpikeTrains)./
size(tenSpikeTrains,1));
line(xlim,[0.5 0.5],'linestyle','--',
'color','k')
title('spike probability given 10 stimulus
trials')
xlabel('time (in milliseconds)')
ylabel('probability of spiking at this time')
% End MATLAB code
```

In MATLAB, we simply normalize by the number of spike trains. Again, we see the power of MATLAB when handling matrices. Thus, represent something as a matrix whenever possible, as it will allow you to bring powerful tools to bear.

Pseudocode

```
§ Begin English explanation of code for Figure 8
Create the figure plotting area
Put the green shading in the
Plot a bar plot of the mean of tenSpikeTrains
Plot a dashed black horizontal line at y = 0.5
Set the title to 'spike probability given 10 stimulus trials'
Set the x-axis label to 'time (in milliseconds)'
Set the y-axis label to 'probability of spike occurrences at this
time'
Save the figure to 'Figure 8 normalized PSTH with cutoff.png'
§ End English explanation of code for Figure 8
```

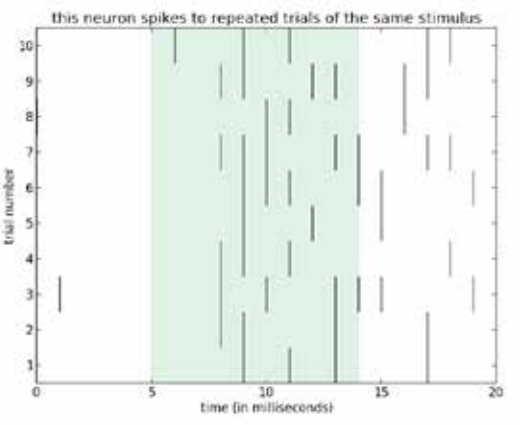

Figure 8. Spike probability given 10 stimulus trials.

At first glance, it seems that our hypothesis of a consistent probability of 0.5 spikes over the stimulus interval does not hold and is false. For now, we note that there are several phenomena that are not consistent with our hypothesis. For instance, there is a latency period before the spikes start, and there is a phasic component of high spike probability around 4 ms and a tonically maintained probability around 0.4 or 0.5 for the remainder of the stimulus thereafter, even continuing for a while after the stimulus turns off. So it looks as though things are more complicated than we initially expected—which in biology should be expected. To further illuminate what is going on, we could vary the intensity of the light and hypothesize that the neuron fires more spikes with a shorter latency for brighter stimuli.

NOTES

Let's load in the Python dictionary called **tenIntensities.pkl** (or in the case of MATLAB, the .mat file *tenIntensities.mat*).

Python dictionaries are arranged in a manner where all of the values are assigned to a key. The key can be either a text string or a number, and the value can be almost anything: a number, string, list, array, or even another dictionary. To view the keys for this dictionary in Python, we type:

```
>>> tenIntensities.keys()
```

```
['4_intensity',
 '2_intensity',
 '8_intensity',
 '0_intensity',
 '7_intensity',
 '5_intensity',
 '9_intensity',
 '6_intensity',
 '3_intensity',
 '1_intensity']
```

Each key corresponds to the intensity of the stimulus, ranging from 0 to 9. So to get the values for, say, the key 4_intensity, we type:

```
>>> tenIntensities['4_intensity']
```

```
[[15.0, 15.0, 22.0, 25.0, 34.0, 23.0],
 [10.0, 32.0, 34.0, 22.0, 34.0],
 [13.0, 17.0],
 [9.0, 30.0, 36.0, 33.0],
 [8.0, 32.0, 31.0, 35.0, 19.0, 36.0, 19.0],
 [30.0, 13.0, 31.0, 36.0],
 [21.0, 31.0, 27.0, 30.0],
 [12.0, 15.0, 23.0, 39.0],
 [23.0, 30.0, 14.0, 23.0, 20.0, 23.0],
 [9.0, 16.0, 13.0, 27.0]]
```

We observe that each list within this value comprises another list of spike times. We use these spike times now to visualize the PSTHs over all stimuli.

Our raster and PSTH plotting techniques here are the same as before, with two main differences. The first (and most obvious from the output figure) is that we now have 20 subplots: 10 rows and two columns. In the Python package **matplotlib**, the first subplot, referenced as 1, is always at the top left. As we increase this index, our plot moves across each row to the right, to the end of the row, before moving down to the next column.

In the Python code, we make use of the **numpy** function **histogram**. It calculates the number of occurrences of values over a given range. It then returns the count of how many values occurred within each bin, and also returns the bins used, assigning these to the variables preceding the equals sign (it doesn't plot anything). We use the variables obtained with **np.histogram** to make our histogram below with the function **bar**. In the code below, we also give **numpy** a nickname, "**np**," which lets us refer to packages in a shorthand manner, e.g., : **np.histogram()**:

NOTES

Python

```
# The Python way for Figure 9
import pickle
with open('tenIntensities.pkl', 'rb') as handle:
    tenIntensities = pickle.load(handle)

fig = plt.figure()
numIntensities = len(tenIntensities)
nbar={}
for key in tenIntensities.keys():
    ax=plt.subplot(numIntensities,2,float
    (key[0])*2+1)
    for trial in range(10): # this relies on
there being 10 trials per stimulus intensity

plt.vlines(tenIntensities[key] [trial],trial,
trial+1)
    plt.xlim([0,20]);plt.ylim([0,10])
    plt.ylabel('intensity: '+str(key[0])+'\
ntrial #',style='italic',fontsize=5)
    plt.yticks(fontsize=5)
    plt.axvspan(5,14,alpha=0.1*float(key[0]),
color='g')
    if float(key[0]) < 9:
        plt.xlabel('');plt.xticks([])
    else:
        plt.xlabel('time in milliseconds')
    if float(key[0]) == 0:
        plt.title('raster plot of spiking for
each intensity',fontsize=10)

    ax=plt.subplot(numIntensities,2,float
(key[0])*2+2)
    plt.axvspan(5,14,alpha=0.1*float(key[0]),
color='g')
    spikeTimes = [a for b in tenIntensities[key]
for a in b] #1
    nOut,bins=np.histogram(spikeTimes,
bins=range(20))
    nbar[float(key[0])] = nOut/10.
    plt.bar(bins[:-1],nOut/10.)
    plt.xlim([0,20]); plt.ylim([0,1])
    plt.yticks(fontsize=5)
    plt.ylabel('spike prob',style='italic',
fontsize = 6)
    if float(key[0]) == 0:
        plt.title('PSTH for each intensity',
fontsize=10)
    if float(key[0]) < numIntensities-1:
        plt.xlabel(''); plt.xticks([])
    else:
        plt.xlabel('time in milliseconds')
plt.savefig('Figure subplot 10 intensity rasters
and psths.png')
# End Python code for Figure 9
```

MATLAB

```
load('tenIntensities.mat')
figure
a= [1:2:20];
b =[2:2:20];
for ii = 1:size(A2,1)
subplot(10,2,a(ii))
if ii == 1
        title('raster plot for each intensity')
    end
rectangle('Position',[5,0,9,11],'FaceColor',[1-
(0.1.*ii) 1 1-(0.1.*ii)],'linestyle','none')
for jj = 1:10
spikeTimes = find(A2{ii,1}(jj,:)==1)-1
line(repmat(spikeTimes,2,1),repmat([jj-0.5; jj+0
.5],1,length(spikeTimes)),'color','k')
xlim([0 20])
ylim([0.5 10.5])
set(gca,'xtick',[])
end
end

%xlabel('time (in milliseconds)')
%ylabel('Trial number')
set(gca,'Layer','top')

for ii = 1:size(A2,1)
subplot(10,2,b(ii))
    if ii == 1
        title('PSTH for each intensity')
    end
rectangle('Position',[5,0,9,8],'FaceColor',[0.7 1
0.7],'linestyle','none')
hold on
x=0:length(A2{ii,1})-1;
bar(x,sum(A2{ii,1}));
xlim([-0.5 20])
ylim([0 8])
set(gca,'xtick',[])
%xlabel('time (in milliseconds)')
%ylabel('# spikes')
end
```

In the Python code, we slipped in the initialization of the *dictionary* **nbar**, which we did with curly braces {}. Later in the code, we assign *values* from **nOut**, which represents the number of occurrences of spikes for particular times, to *keys* of **nbar**. We access all the values of nbar below with **nbar.values()**.

Our ability to measure latency to first spike here becomes quite difficult. We can qualitatively say that higher-intensity stimuli cause shorter latency responses. We will relish for now our ability to make colorful plots out of the spike data.

NOTES

Pseudocode

§ Begin English explanation of code for Figure 9
Create new figure
Declare an empty dictionary *nbar*
For each key in the dictionary *tenIntensities*

First column, raster plots for each intensity of the light stimulus. Plot in subplot corresponding to each intensity (an example *key* is: *'7_intensity'*, so *key[0]* is the *0th* value of the string *'7_intensity'*, which is *'7'*, and *float('7')* equals 7.0, We take that value times 2 and add 1 so that subplot indices count by row. For each trial, this relies on there being 10 trials per stimulus intensity. Plot vertical lines corresponding to the spike times.

Format the raster plots: set the *x*-axis and *y*-axis limits. Set the *y*-axis label to the intensity, use '\n' as a carriage return, label the trial number, italic, and fontsize, set the *y*-ticks' fontsize. Add the green box, use the alpha value so that the transparency scales with intensity. If the intensity is <9, that is, if we are not plotting at the bottom. Do not label the *x*-axis.

Else, that is, if the intensity is 9. Label the *x*-axis as 'time in milliseconds.'

If the intensity is 0, that is, if we are plotting at the top. Set the title to 'raster plot of spiking for each intensity' and fontsize to 10.

First, perform list comprehension to unpack list. In the second column, plot the PSTHs for each intensity of the light stimulus. Plot the subplot in the second column, with each increasing intensity moving down a row. Plot the green box and set the *alpha* value to correspond to the intensity of the stimulus. Extract all the spike times for a stimulus intensity. Get *nOut*, a histogram array binned by the value *bins*. Add the values in *nOut/10.* to the dictionary *nbar* with key *float(key([0]))*. Plot the PSTH with bar function, calling all the bins except the last bin, and scaling *nOut* by the number of trials (10).

Format the PSTHs: set the *x*-axis and *y*-axis limits to [0, 20] and [0, 1], respectively. Set the *y*-axis fontsize. Set the *y*-label to 'spike prob,' make it *italic*, and set the fontsize to 6. If we are in the first row (at the top). Set the title to 'PSTH for each intensity,' with a fontsize of 10.

If we are in any plot above the bottom plot, turn off the *x*-label and *x*-ticks.

Else if we are at the bottom plot, set the *x*-label to 'time in milliseconds.'

Save the figure.
§ End English explanation of code for Figure 9

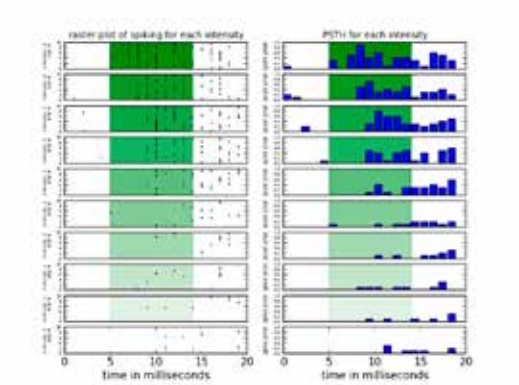

Figure 9. Raster plot of spiking and PSTH for each light stimulus intensity.

Python

```
# Begin the Python way for Figure 10
fig = plt.figure()
ax = plt.subplot(111)
aa = ax.imshow(nbar.values(),cmap='hot',
interpolation='bilinear')
plt.yticks([x for x in range(10)],[str(x) for x
in range(10)[::-1]])
plt.ylabel('stimulus intensity')
plt.xlabel('time in milliseconds')
plt.title('heat map of mean spiking for various
intensity stimuli')
cb = fig.colorbar(aa,shrink=0.5)
cb.ax.set_ylabel('mean spikes per time bin')
# End Python way for Figure 10
```

MATLAB

```
for ii = 1:size(A2,1)
    A3(ii,:) = (sum(A2{ii,1})./10);
end
A3(:,22:100) = [];
figure
h = pcolor(A3);
set(h,'Facecolor','interp')
set(h,'Linestyle','none')
set(gca,'YDir','reverse')
colormap('hot')
h = colorbar;
ylabel(h,'mean spikes per time bin')
xlabel('time (in milliseconds)')
ylabel('stimulus intensity')
title('heat map of mean spiking for stimuli of
varying intensity')
```

NOTES

Pseudocode
§ Begin English explanation of code for Figure 10 Create plotting area Specify that we're making one subplot Plot the values of *nbar* as an image with a hot colormap and the colors bilinearly interpolated Set where the *y*-ticks go and what their labels are Set the *y*-axis label to 'stimulus intensity' Set the *x*-axis label to 'time in milliseconds' Set the title to 'heat map of mean spiking for various intensity stimuli' Create a colorbar, use the *shrink* command to customize its height Set the colorbar label to 'mean spikes per time bin' § End English explanation of code for Figure 10

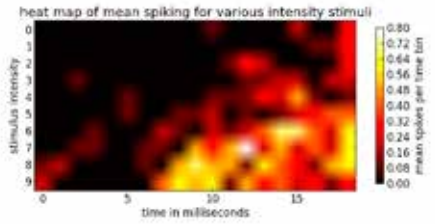

Figure 10. Heat map of mean spiking for various intensity stimuli.

Pseudocode	Create figure plotting area Specify two rows and two columns to plot, select the first one Plot the bar values as image with colormap hot, bilinear interpolation, and aspect 1.2 Turn the *y*-axis and *x*-axis tick marks off Specify two rows and two columns to plot, select the second one Plot the bar values as image with colormap bone, nearest interpolation, and aspect 1.2 Turn the *y*-axis and *x*-axis tick marks off Specify two rows and two columns to plot, select the third one Plot the bar values as image with colormap jet, bicubic interpolation, and aspect 1.2 Turn the *y*-axis and *x*-axis tick marks off Specify two rows and two columns to plot, select the fourth one Plot the bar values as image with colormap cool, nearest interpolation, and aspect 1.2 Turn the *y*-axis and *x*-axis tick marks off Save figure
Python	fig = plt.figure(); ax = plt.subplot(221) aa = ax.imshow(nbar.values(),cmap='hot' ,interpolation='bilinear',aspect=1.2) plt.yticks([]); plt.xticks([]) ax = plt.subplot(222) aa = ax.imshow(nbar.values(),cmap='bone', interpolation='nearest',aspect=1.2) plt.yticks([]); plt.xticks([]) ax = plt.subplot(223); aa = ax.imshow(nbar.values(),cmap='jet', interpolation='bicubic',aspect=1.2) plt.yticks([]); plt.xticks([]) ax = plt.subplot(224) aa = ax.imshow(nbar.values(),cmap='cool', interpolation='nearest',aspect=1.2) plt.yticks([]); plt.xticks([]) plt.savefig('Figure 10—four heat maps labels off.png')
MATLAB	figure ax1 = subplot(2,2,1) h = pcolor(A3); set(h,'Facecolor','interp') set(h,'Linestyle','none') set(gca,'YDir','reverse') colormap(ax1,'hot') axis off ax2 = subplot(2,2,2) h = pcolor(A3); set(h,'Linestyle','none') set(gca,'YDir','reverse') colormap(ax2,'bone') axis off ax3 = subplot(2,2,3) h = pcolor(A3); set(h,'Facecolor','interp') set(h,'Linestyle','none') set(gca,'YDir','reverse') colormap(ax3,'jet') axis off ax4 = subplot(2,2,4) h = pcolor(A3); set(h,'Linestyle','none') set(gca,'YDir','reverse') colormap(ax4,'winter') axis off

NOTES

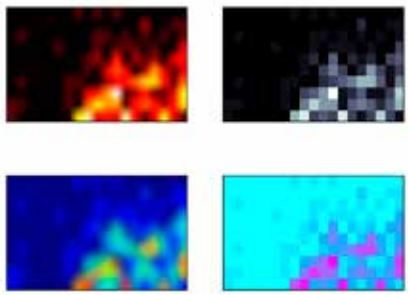

Figure 11. Four heat maps.

So that's it for basic spike wrangling. Admittedly, this was somewhat of a toy example, but we have to get started somewhere. You will find in your own dataset that you will need to employ a variety of wrangling methods to achieve your desired outcome—the goal here is to learn the various tools that help you to do so.

Questions We Did Not Address

In the 10 intensity stimulus-response set, what order were the stimuli presented in? Did all of the bright intensity stimuli occur sequentially before moving to a lower intensity stimulus? If so, might the ordering of the stimuli influence the neuron's response characteristics? Might the neuron exhibit *adaptation*, where its firing probability adapts as a function of previous stimuli and its internal characteristics? How much time of no stimuli was there between trials—what was the intertrial interval, or ITI? How might changing the ITI influence the neuron's spiking?

What other optogenetic tools could we use to study the activity of single neurons? How many photons are required for the activation of channelrhodopsin-2? What sort of latency of activation patterns would we expect for other optogenetic tools? What sort of experimental preparation would be required to mimic the experiment in this chapter? And lastly, what sort of experimental apparatus would be necessary to record from a single neuron?

Thoughts on the Proper Unit of Measurement for Spiking Activity of Single Neurons

Data derive from the outcome of a measurement process. A unit of measurement is the fundamental unit we use to express the quantity of a given quality. For instance, the currently agreed upon unit of measurement for length in the SI system is the meter, defined as "the length of the path travelled by light in vacuum during a time interval of 1/299,792,458 of a second." (Taylor and Thompson, 2008). Consequently, all lengths that we wish to measure are then expressed in multiples of this reference length, e.g., 2 m or 0.5 m. This raises the question of what the appropriate unit of measurement for spiking activity (firing rate) is. The firing rate of a neuron in response to a given stimulus (or even in the absence of a stimulus) is a quality of the neuron. The implication is that the neuron in question discharges action potentials (or "spikes") a certain number of times in a given interval, usually 1 s. The choice of this interval is probably what led to the fundamental confusion that one often sees in neuroscience publications. Firing rates are frequently expressed in terms of "Hz," e.g., "the neuron fired at a rate of 30 Hz." The "Hz" in question refers to the unit of measurement of a periodic, oscillatory process, namely 1 cycle (or period) per second. Unfortunately, this fundamentally mischaracterizes the very nature of action potentials. In contrast to harmonic oscillators (e.g., the motion of guitar strings), action potentials are neither cyclical nor periodic. Most frequently, they are conceptualized as "spikes," or point processes, in which case only the time when they occurred and how often this happened in a given interval are meaningfully interpretable. Spiking activity of single neurons is notoriously aperiodic and highly irregular—interspike intervals in a spike train are close to what would be expected from a Poisson process (Softky and Koch, 1993), and the variance of spike counts upon repeated stimulation suggests overdispersion (Taouali et al., 2016). Finally, it makes practical sense to avoid expressing firing rates in Hz simply in order to avoid the potential confusion when plotting them simultaneously with quantities that are appropriately expressed in Hz, such as the temporal frequency of a stimulus or the power of an analog signal in a certain frequency bin. The debate about the theoretical significance of neural oscillations is heated enough (Shadlen and Movshon, 1999) without implying that spike rates are inherently oscillatory as well.

But if it is not Hz, what is the proper unit for firing rate? Because spikes are typically defined by the voltage trace recorded from an electrode in the brain crossing a reasonable threshold—and recorded as the time at which this crossing happened—and then counted, it makes sense to express spikes in units of impulses per second (ips), sp/s, or simply events (threshold crossings) per second. All of these units are conceptually sound, and it is perhaps this range of equally suitable available options that prevented any of them from catching on as a consensus. In military contexts, the "rate of fire" (of rapid-firing guns) is typically expressed in rounds per second (rps), so by analogy, spikes per second (which is what we care about in a firing rate) is perhaps the most apt.

NOTES

Historically, there has been a movement to replace these units with the eponym "Adrians," in honor of the pioneering Lord Edgar Douglas Adrian, the original discoverer of the rate code (Adrian, 1926), who won the Nobel Prize in 1932 and is the purported great-grandfather of many a neurophysiologist (neurotree.org). However, this unit did not catch on either, and given the problematic nature of eponyms, this is perhaps just as well (Wallisch, 2011). Even so, almost anything would be better than expressing firing rates in Hz, which is fundamentally misleading. To repeat: a rate is not a frequency. For actual frequencies, the entire signal scales with it if the frequency changes. In contrast, the individual action potentials remain invariant regardless of spike rate. These are fast events that have to be sampled frequently—or at high frequency—in order to be captured, even (or particularly) if the actual spike rate is very low.

Correlating Spike Trains

This section introduces the first fully developed data analysis project as well as the conceptual framework to do so in a principled fashion: the canonical data analysis cascade. Here, we go through the entire data analysis cascade—from loading data to cleaning it, to representing it in the right format, to doing the actual analysis, to making output figures and saving the results.

The authors vividly remember our first five years of coding with Python or MATLAB: specifically, writing code to analyze data. The latter is rather different from coding for, let's say, building a GUI. Everything becomes more complicated when data get involved, and we were admittedly rather lost during that entire time, writing convoluted code for programs that included thousands of lines—code that was impossible to maintain or understand even after a short period of time. In essence, we were lacking the principles of software development design for the analysis of data.

Since that time, we have discovered principles that work, which we will outline and detail in this section. It has not escaped our notice that these principles closely resemble what seems to have been implemented by the perceptual system (at least in the primate, to the degree of understanding we have now). This makes sense: perceptual systems are designed to analyze the environment in order to extract relevant actionable information. We will show this using the example of the visual system because it is (to date) the most studied and perhaps best understood. Perceptual systems have been in development for hundreds of millions of years under relentless evolutionary pressure, yielding a high-performance analysis framework. As far as we can tell, all sensory systems (with the exception of olfaction, which is special) follow the five steps outlined next, and there are principled reasons for this.

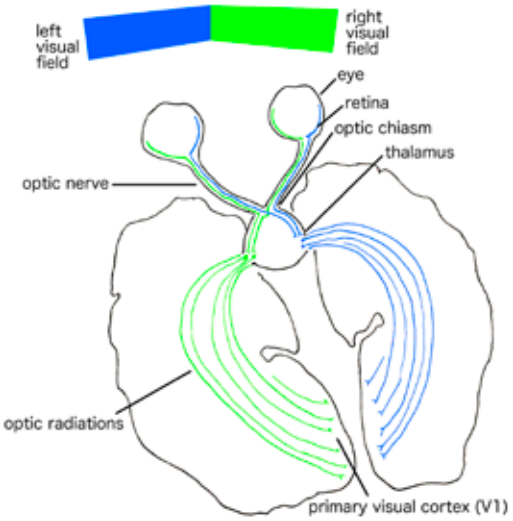

Figure 12. A cartoon of the primate visual system up to area V1. This illustration omits quite a few structures, e.g., the superior colliculus and the suprachiasmatic nucleus, but shows the basic signal flow.

Step 1: Transduction

Every sensory system has to convert some kind of physical energy in the environment into the common currency of the brain. This currency consists of action potentials or spikes. In the case of vision, photons enter the eye through a narrow aperture (the pupil) and are focused on the retina by the lens. The retina transduces (i.e., converts) the physical energy into action potentials. What leaves the eye is a train of action potentials (a spike train, discussed earlier) carried by the optic nerve. The coding equivalent of this is to write dedicated "loader" code whose purpose is to load data from whatever format it was generated in. Each physiological data collection system creates its own data files; for example, .plx files, .nev files, .nex files, .xls files, and .txt files are some popular formats. For one to be able to do anything with these in Python or MATLAB, they have to be converted into a format that Python or MATLAB can use first.

Step 2: Filtering

Once a spike train reaches the cortex, it will be processed. This can be a problem either if it corresponds to information that is not relevant at the time or if it is malformed in some way. In other words, the brain needs a gatekeeper to keep this irrelevant information out. In the brain, this step corresponds to the thalamus—specifically, the LGN in the visual

NOTES

system. This nucleus relays information from the retina to the visual cortex, but does so in a selective fashion. Of course, this raises the question of how the thalamus knows what irrelevant information is before it has been analyzed by cortex. The brain solves this problem in several ways, including via an analysis for low-level salient features, such as fast motion or high contrast, and then feeds that information back to the thalamus; many recurrent feedback loops fine-tune the filtering process.

Similarly, we strongly recommend implementing this step in code. It is very difficult to write analysis code that is flexible enough to handle data that are not usable, whether that is because parts of the data are missing, the study participant is not doing what he or she is supposed to, the data are corrupted, the electrode is not working, or the like. If such data enter the processing cascade, they usually render the result meaningless or break the code altogether. It is best to avoid such data being processed in the first place. This step can be called "cleaning," "pruning," or "filtering" the data. It is important that this step be performed agnostic to the results of the analysis. If you throw out data that do not conform to your hypothesis, you can get data supporting any hypothesis—this is considered "data doctoring," so don't do it. In contrast, this kind of integrity check before performing full-scale analysis is critical. Another analogy for this step is that of a gatekeeper—the CEO (the cortex) can't be bothered with irrelevant information. If all of it were let in, then nothing would get done. That's where a strict personal assistant comes in. Make sure to implement that gatekeeper in code.

Step 3: Formatting

The next step performed by the visual system is a categorical analysis of the data arriving from the thalamus. This step is performed by the early visual system, particularly V1. Here, the visual system determines the location and basic orientation of line segments (Hubel and Wiesel, 2004), for example, what is foreground and what is background figure (Craft et al., 2007). Heuristically, this step can be understood as setting the stage for further analysis, and not so much as doing a lot of analysis here already. The reasons for this will be understood more clearly when discussing the next step. Thus, we conceive of this step as "formatting" the data for further analysis: it is an absolutely critical step for data analysis. Once data are formatted properly, the rest of the analysis is usually rather straightforward. It might be unsettling to the beginner, but it is not unusual to spend *most* of one's time writing analysis code in this step, simply "formatting" the data. Once data structures are set up properly, the actual analysis often corresponds to something very simple, such as "loop through all participants in these conditions, then compare their means." Similarly, the visual system recognizes the importance of this step—in the primate, the "early" visual system makes up approximately half the visual system by area (Wallisch, 2014).

Step 4: Calculator

In the visual system, the calculation step is implemented by the extrastriate cortex—the cortical regions after striate cortex (or primary visual cortex) in the visual processing stream. Interestingly, whereas the previous steps have been done mostly in serial fashion (the feedback to thalamus notwithstanding), this step is better referred to as plural steps because they happen in parallel, meaning that the signal might split into two or more copies so that multiple processes can occur on it simultaneously (Wallisch and Movshon, 2008). The fundamental reason for this is that, in order to achieve the goal of the computation, many computations have to abstract from some aspects of the source information, in effect destroying it. This might be information that is also important but is better computed, in parallel, by another area. In effect, different parts of extrastriate cortex (at a minimum, dorsal and ventral stream) (Mishkin et al., 1983) make copies of the information provided by V1 and work on that toward some outcome. For instance, in order to compute the speed of objects, it might be necessary to abstract from their location and identity information that is also crucial to the organism, but cannot be computed at the same time by the same area or serially. A parallel approach is perhaps the best attempt to solve this problem.

We recommend doing something similar in code to implement this step. Specifically, we recommend creating as many parallel analysis streams as there are analysis goals. The number of analysis goals is given by how many theoretical questions need to be answered for any given project. For instance, it is conceivable that one analysis is concerned with the mean response of neurons under certain conditions, whereas another deals with its variability; in this case, the underlying analyses are best done on copies of the original dataset and are complementary to each other. More analyses (e.g., a correlational analysis) might build on these, as we will attempt here. We recommend labeling these steps 4a, 4b, 4c, etc., in the code, signifying analysis steps that are in principle independent of each other but can rely on each other (e.g., 4c being executed after 4a). Note that parallel processing has a similar meaning

NOTES

in computer science, in which computations are performed on data split onto different machines.

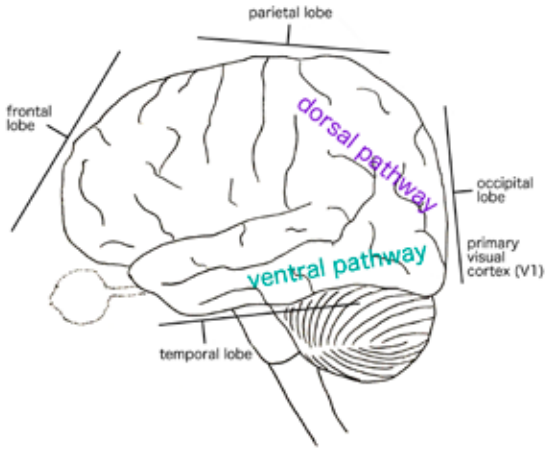

Figure 13. A cartoon of the extrastriate visual system after area V1.

Step 5: Output

This might come as a surprise to people living in the modern age, but the purpose of the visual system is not to provide fancy images for one's viewing pleasure but to improve the survivability of the organism. This is true for sensory systems in general. Perception is not an end in itself: unless it results in motor output, the outcomes of its calculations are irrelevant. Over time, the system has been optimized to provide more adaptive outputs. In primates, the end result of the visual processing cascade in extrastriate areas is linked with motor cortex in order to transform the visual input to real-world outputs and memory systems to store information (the results of the computations on the inputs). We will do the same here, in code.

Specifically, we will hook up the outputs of step 4 (e.g., 4a, 4b) to corresponding outputs (i.e., 5a, 5b). Sometimes, we just want to output some numbers to the screen. Often (but not always) in the real brain, not every computation results in a motor output—this will be a corresponding figure that visualizes the outcome of a computation. Usually, we want to store the results so that we can load them in later without having to redo the entire analysis from scratch (this is particularly important, as some analyses are rather time-consuming). You will want to have as many output functions (or files) as there are outputs of step 4.

Summing Up

And that sums up the general purpose framework ("the canonical data analysis cascade") that can be used for any data analysis project. We believe it is efficient and use it ourselves on a daily basis. As far as we can tell, most sensory systems do as well. Note that sometimes, some of these steps can be combined into one, e.g., by attempting to do the filtering (pruning/cleaning) step at the same time as the formatting step in order to realize further efficiency gains. Sometimes you will want to combine calculation and output steps, although one usually can output the same information in multiple ways. As long as you are careful when doing so, there is no problem with combining steps, although we recommend separating the steps more strictly if you are an absolute novice until all of this becomes second nature.

Concluding Advice: Add Step 0 and Step 6

To conclude, we strongly advise re-creating these five principal steps in code when attempting any full-scale data analysis project. Specifically, we recommend partitioning analysis code into these steps, either by writing individual files that correspond to each step or by dividing up the code into self-contained segments. What is important is that each logical segment fits on a screen (or just about fits); if it does not, it will be very difficult to maintain. In our experience, analysis code has to be revisited with surprising regularity (e.g., when Reviewer 3 asks for additional analyses), and unfortunately, memory for code seems to be rather transient. In other words, if you do not organize and comment your code well, you will not understand the code you wrote, even after a relatively short time. This can put you in a difficult position, for instance, when pressed for time, as when grant deadlines loom or code reviews are pending. So it is best to avoid it.

In addition to these five steps implemented by sensory systems of the brain, we recommend adding a zeroth and a sixth step. The zeroth step is an "initialization" step. In the brain, this might correspond to birth, and similarly, you want to start from as nearly a blank slate as possible. In our experience, many logical errors in programming are caused by something lingering in memory that you forgot about that affects the execution of your code. These errors are hard to track down and can be catastrophic. It is best to avoid them in the first place, and the best way to do that is to clear the memory before doing anything else. In addition, in this step you will want to define some constants, parameters, or analysis flags. The reason to

NOTES

do this here (and not strewn throughout the code) is that you have an easily accessible section—all in one place, at the beginning of the code—that governs its execution, e.g., options to run the analysis with or without normalization, settings for which baseline to use, and similar instructions. Even the brain at birth does not start with a completely blank slate, owing to the complex nature of the development of synaptic organization via both genetics and maternal environment, and there is a reason for that.

Finally, in the spirit of making the code maintainable—and even runable a couple of months after writing it, we recommend writing a file that corresponds to the sixth step. This step is a kind of "wrapper" and "readme" file that, if you wrote a file for each of the five steps, calls up the right files with the right parameters in the right order and provides some kind of documentation of what is going on.

Acknowledgments

This chapter was adapted with permission from Chapters 3 and 4 of the forthcoming book by Pascal Wallisch, *Neural Data Science*, to be published by Elsevier.

References

Adrian ED (1926) The impulses produced by sensory nerve endings. J Physiol 61:49–72.

Boyden ES, Zhang F, Bamberg E, Nagel G, Deisseroth K (2005). Millisecond-timescale, genetically targeted optical control of neural activity. Nat Neurosci 8:1263–1268.

Craft E, Schütze H, Niebur E, Von Der Heydt R (2007) A neural model of figure–ground organization. J Neurophysiol 97:4310–4326.

Hubel DH, Wiesel TN (2004) Brain and visual perception: the story of a 25-year collaboration. New York, NY: Oxford University Press.

Mishkin M, Ungerleider LG, Macko KA (1983) Object vision and spatial vision: two cortical pathways. Trends Neurosci 6:414–417.

Peters T (2004) PEP 20—The Zen of Python. Available at https://www.python.org/dev/peps/pep-0020/

Rieke F (1999) Spikes: exploring the neural code. Cambridge, MA: The MIT Press.

Shadlen MN, Movshon JA (1999) Synchrony unbound: a critical evaluation of the temporal binding hypothesis. Neuron 24:67–77.

Softky WR, Koch C (1993) The highly irregular firing of cortical cells is inconsistent with temporal integration of random EPSPs. J Neurosci 13:334–350.

Taouali W, Benvenuti G, Wallisch P, Chavane F, Perrinet LU (2016) Testing the odds of inherent vs. observed overdispersion in neural spike counts. J Neurophysiol 115:434–444.

Taylor BN, Thompson A (eds) (2008) The international system of units (SI), Ed 8. NIST Special Publication 330. Gaithersburg, MD: National Institute of Standards and Technology. Available at http://physics.nist.gov/Pubs/SP330/sp330.pdf

Tye KM, Deisseroth K (2012) Optogenetic investigation of neural circuits underlying brain disease in animal models. Nat Rev Neurosci 13:251–266.

Wallisch P (2011, January 7) Eponyms are stifling scientific progress. Pascal's Pensées. Available at http://pensees.pascallisch.net/?p=686

Wallisch P (2014, July 21) The relative scale of early visual areas. Pascal's Pensées. Available at http://pensees.pascallisch.net/?p=1788

The Statistical Paradigm

Robert E. Kass, PhD

Department of Statistics and Machine Learning Department
Carnegie Mellon University
Pittsburgh, Pennsylvania

The Contributions of Statistics

Most scientists who analyze data have had little exposure to the field of statistics, and what they know about it has either been self-taught from haphazard sources or come from a single introductory course. Although in many circumstances such limited knowledge is enough to get the job done, users of statistics who lack deeper training rarely appreciate the principles that drive the discipline. The book *Analysis of Neural Data* (Kass et al., 2014), which I wrote with Uri Eden and Emery Brown, attempts to highlight those principles in an accessible exposition, weaving them into the body of methods traditionally discussed in basic courses and including some topics that are more advanced. In this Short Course chapter, I summarize several of the key ideas, quoting liberally from the book. One of the reasons my colleague Emery Brown and I started down the long path toward writing that book was our shared discomfort with treatments of data reported in the many otherwise high-quality papers we were reading in the neuroscience literature. As we say in the book (hereafter, all quotes are from the book without explicit citation):

> *Many researchers have excellent quantitative skills and intuitions, and in most published work statistical procedures appear to be used correctly. Yet, in examining these papers we have been struck repeatedly by the absence of what we might call statistical thinking, or application of the statistical paradigm, and a resulting loss of opportunity to make full and effective use of the data. These cases typically do not involve an incorrect application of a statistical method (though that sometimes does happen). Rather, the lost opportunity is a failure to follow the general approach to the analysis of the data, which is what we mean by the label "the statistical paradigm." (Kass et al., 2014, 2)*

Overview: Two Fundamental Tenets of the Statistical Paradigm

After numerous conversations with colleagues, we have arrived at the conclusion that among many components of the statistical paradigm, summarized below, two are the most fundamental (Kass et al., 2014, 8):

1. Statistical models are used to express knowledge and uncertainty about a signal in the presence of noise, via inductive reasoning.

2. Statistical methods may be analyzed to determine how well they are likely to perform. (Kass et al., 2014, 9)

I will briefly explain these ideas.

Statistical models

Statistical models describe regularity and variability of data in terms of probability distributions. The two examples below are from Kass et al. 2014, pages 9–13.

Example 1

Marshall and Halligan (1988) reported an interesting neuropsychological finding from a patient, identified as P.S. This patient was a 49-year-old woman who had suffered damage to her right parietal cortex that reduced her capacity to process visual information coming from the left side of her visual space. For example, she would frequently read words incorrectly by omitting left-most letters ("smile" became "mile"), and when asked to copy simple line drawings, she accurately drew the right-hand side of the figures but omitted the left-hand side without any conscious awareness of her error. To show that she could actually see what was on the left but was simply not responding to it—a phenomenon known as "blindsight"—the examiners presented P.S. with a pair of cards showing identical green line drawings of a house, except that on one of the cards, bright red flames were depicted on the left side of the house. They presented to P.S. both cards, one above the other (the one placed above being selected at random), and asked her to choose which house she would prefer to live in. She thought this was silly "because they're the same," but when forced to make a response chose the non-burning house on 14 out of 17 trials. This would seem to indicate that she did, in fact, see the left side of the drawings but was unable to fully process the information. But how convincing is it that she chose the non-burning house on 14 out of 17 trials? Might she have been guessing?

If, instead, P.S. had chosen the non-burning house on 17 out of 17 trials, there would have been very strong evidence that her processing of the visual information affected her decision-making; on the other hand, a choice of 9 out of 17 clearly would have been consistent with guessing. The intermediate outcome (14 out of 17) is of interest as a problem in data analysis and scientific inference precisely because it feels fairly convincing but leaves us unsure: a thorough, quantitative analysis of the uncertainty would be very helpful.

The standard way to begin is to recognize the variability in the data, namely, that P.S. did not make the same choice on every trial; we then say that the choice made by P.S. on each trial was a random event, that the probability of her choosing the non-burning house on each trial was a value p, and that the responses on the different trials were independent of each other. These three assumptions use probability to describe the variability in the data. Once these three assumptions

NOTES

NOTES

are made, it becomes possible to quantify the uncertainty about p and the extent to which the data are inconsistent with the value $p = 0.5$, which would correspond to guessing. In other words, it becomes possible to make statistical inferences. Specifically, using standard statistical methods, we obtain an approximate 95% confidence interval (CI) of (0.64, 1.0). This CI expresses strong confidence that, based on these data, p is substantially larger than 0.5.

In this example, we introduced an abstract quantity p in order to describe the variability of outcomes for the repeated-choice framework of the investigation. More specifically, we can label the choice "burning" as a value 1 and the choice "non-burning" as a value 0; we let X be the abstract quantity that determines the choice so that on a given trial, we have either $X = 1$ or $X = 0$; we say that p is the probability that $X = 1$, often written as $p = P(X = 1)$; and, finally, if we let Y be the sum of the values of X across all 17 trials, then Y could, in principle, take any value from 0 to 17 and, beginning with $p = P(X = 1)$, we can use probability theory to compute the probability of any value taken by in terms of p, such as $P(Y = 14)$. The probability distribution for Y is called a "binomial distribution," and Y itself is called a "random variable." Random variables are abstract mathematical objects that have probability distributions. We refer to the representation of the data by the random variable Y, together with the binomial distribution for Y, as a "statistical model." It is important to note that there is both a systematic part of the variation, which is what we will care about and call "signal" (here it is represented by p), together with a remaining component of the variation, captured by the binomial distribution, which we call "noise."

Notice here that I have followed a standard statistical convention in using a capital letter to stand for a random variable. This can be helpful in more complicated settings because it allows us to identify quickly key elements of the model that are assumed to be random variables.

An additional statistical convention is to use subscripts on random variables to identify specific instances. In this example, the instances would be the 17 trials, so we would have 17 outcome variables $X_1, X_2, \ldots, X_{17}$ and we would write

$$Y = \sum_{i=1}^{17} X_i = X_1 + X_2 + \cdots X_{17}.$$

The binomial model in this example combines signal and noise in a somewhat subtle way: the signal is a parameter of the binomial distribution, in the sense that we can compute the probability of each outcome once we know p (together with the number of trials). Conceptually, we might write

"*outcome = signal + noise*",

where I have used quotes to indicate that we didn't actually add the signal and noise. In the next example, involving a linear regression model, we do add signal and noise to get the outcome random variable. In linear regression, we relate x and y variables. The signal here has the form

$$y = f(x)$$
$$f(x) = \beta_0 + \beta_1 x,$$

where I have written the intercept and slope of the line as β_0 and β_1 to be consistent with the most common notation used in statistics. The corresponding statistical model introduces the random variable Y for the outcome of y and includes a noise random variable customarily denoted by ε. Because the data come as a set of (x, y) pairs, we use the subscript i to refer to data pair i, and the statistical model becomes

$$Y_i = f(x_i) + \varepsilon_i \qquad (1)$$

and by again taking $f(x) = \beta_0 + \beta_1 x$, we have a linear model.

Example 2

Hursh (1939) presented data on the relationship between a neuron's conduction velocity and its axonal diameter, in adult cats. Hursh measured maximal velocity among fibers in several nerve bundles, and then measured the diameter of the largest fiber in the bundle. The resulting data, together with a fitted line, are shown in Figure 1. The fitted line is determined by least-squares. In this case, the line $y = \beta_0 + \beta_1 x$ represents the approximate linear relationship between maximal velocity y and diameter x. The data follow the line pretty closely, with the intercept β_0 being nearly equal to zero. This implies, for example, that if one fiber has twice the diameter of another, the first will propagate an action potential approximately twice as fast as the second. For the slope, we found $\hat{\beta}_0 = 6.07$ with standard error $\mathrm{SE}(\hat{\beta}_1) = 0.14$. We would report this by saying that, on average, action potential velocity increases by 6.07 ± 0.14 m/s for every micrometer increase in diameter of a neuron. An approximate 95% CI for the slope of the regression line is 6.07 ± 2(0.14) or (5.79, 6.35).

The distinction between data quantities and theoretical quantities

Fundamental to statistical reasoning is the distinction between random variables and the data they represent. For example, histograms are often used to display variation in the data, and this is usually represented

NOTES

by the mathematical notion of a random variable having a probability distribution, with its probability density function (pdf) corresponding to the histogram. It is easy to confuse the two. When we speak of "the distribution of the data," we may be referring to the way the histogram looks, but the data themselves do not follow a probability distribution—probability distributions apply only to random variables. Similarly, we may speak of the mean or variance of some assortment of numbers, and we can also speak of the mean and variance of a probability distribution. When we conceptualize data variation using probability distributions, it is easy to be sloppy in using a term like "the mean" by failing to say whether we are referring to the mean in the data or the mean in the probability distributions. It is fine to be sloppy sometimes, but it is important to recognize the fundamental distinction between data-based quantities and theoretical quantities.

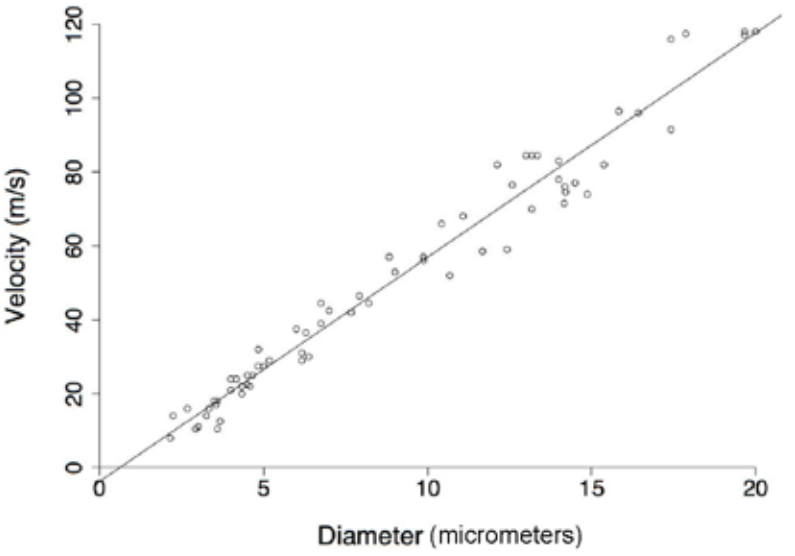

Figure 1. Conduction velocity of action potentials as a function of diameter. The *x*-axis is the diameter in micrometers; the *y*-axis is velocity in m/s. Also shown is the least-squares regression line. Reprinted with permission from Kass et al. (2014), *Analysis of Neural Data,* copyright 2014, Springer Science+Business Media.

Specifically, the mean of a set of numbers $x_1, x_2, \ldots, x_n$ is

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i ,$$

whereas the mean or expectation of a random variable X having pdf $f_X(x)$ is

$$\mu_X = E(X) = \sum_x x f_X(x),$$

where the sum is over all the possible values of x. When there is a continuum of possible x values, we say that the distribution is "continuous" and we write

$$\mu_x = E(X) = \int x f_X(x) dx.$$

Similarly, the variance of X in the continuous case is

$$\sigma^2{}_X = V(X) = \int (x - \mu_X)^2 f_X(x) dx.$$

Often the subscript X is dropped, and the most common statistical notations for a theoretical mean and variance are μ and σ^2, with $\sigma = \sqrt{\sigma^2}$ being the SD.

When data consist of a set of numbers $x_1, x_2, \ldots, x_n$ representing repeated observation or measurement of some quantity under nearly identical experimental conditions, it is common to represent the observations, theoretically, as a set of random variables $X_1, X_2, \ldots, X_n$, all of which follow the same distribution, with all being assumed to be statistically independent. In this case, the random variables constitute a random sample from the distribution of each X. In Example 1, the 17 trials were assumed to be homogeneous and independent in this sense—they would not be independent if the subject suffered from fatigue or some kind of tendency to answer based on previous answers. In Example 2, the ε_i variables were assumed to be homogeneous and independent. The most serious departures from this assumption of linear regression occur when there is temporal structure in the data, with adjacent values of "noise" being related to each other.

Statistical Theory

Statistical theory is used to understand the behavior of statistical procedures under various probabilistic assumptions. In statistics, the greek letter theta (θ) is used to denote a parameter to be estimated from the data. This parameter could be either a scalar or a vector. For instance, θ might be a binomial parameter, as in Example 1, where $\theta = p$, or it could be the pair of

NOTES

coefficients in a line, as in Example 2, where $\theta = (\beta_0, \beta_1)$. A common notation for an "estimator," meaning a random variable that is used to estimate a parameter θ, is $\hat{\theta}$. Sometimes, especially when it is important to stress the generality of an estimation process (so that the quantity it is estimating is implicit rather than explicit), an estimator is written as a random variable T.

Additional information about essential ideas in statistical theory, including elaboration of the points made below, may be found in Kass et al. 2014, Chapter 8.

Mean-squared error

A relatively simple and very commonly applied criterion for evaluating how well an estimator T is able to estimate a parameter θ is mean squared error (MSE), defined by

$$MSE\,(T) = E((T - \theta)^2).$$

An interesting and important feature of MSE, which can be derived with a few lines of algebra, is that it combines two ways an estimator can perform poorly. The first involves the systematic tendency for the estimator T to miss its target value θ. An estimator's "bias" is $\text{Bias}(T) = E(T) - \theta$. When the bias is large, on average T will not be close to θ. The second is the variance $V(T)$. If $V(T)$ is large, then T will rarely be close to θ. Figure 2 illustrates, by analogy with shooting at a bull's-eye target, the situations in which only the bias is large, only the variance is large, both are large (the worst case), and finally, both are small (the best case). Part of the appeal of MSE is that it combines bias and variance in a beautifully simple way:

$$MSE(T) = \text{Bias}(T)^2 + \text{Variance}(T). \quad (2)$$

Optimality

Using MSE, it is possible to evaluate alternative estimators of a parameter θ. A fundamental result

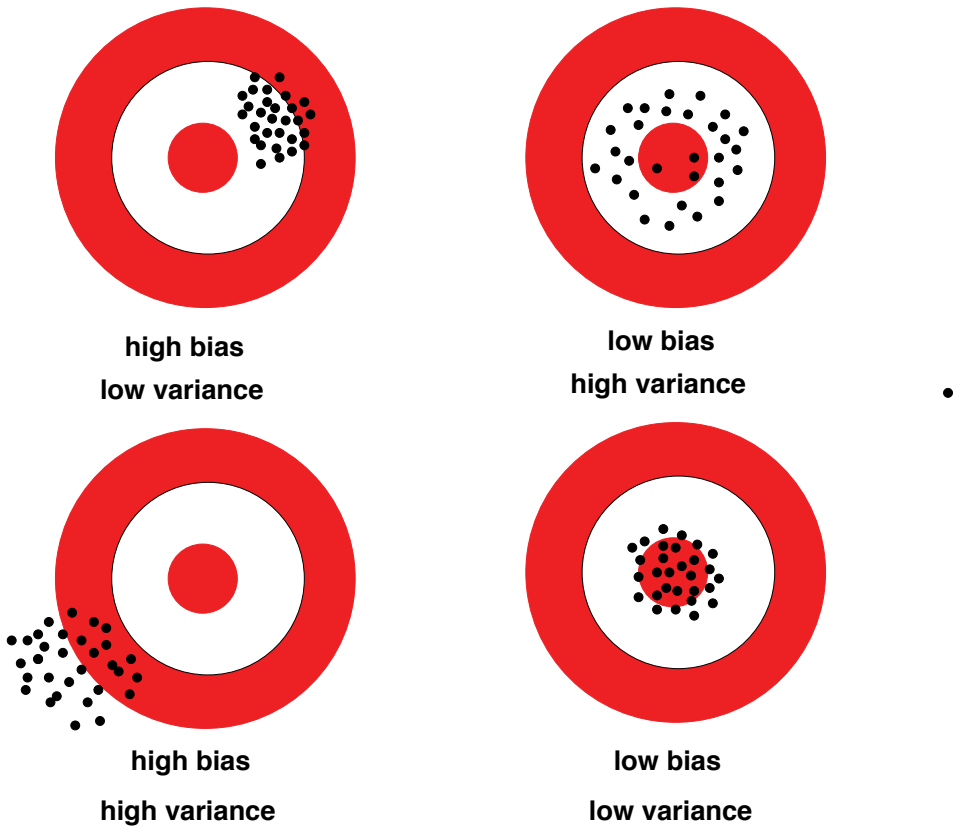

Figure 2. Illustration of shots aimed at a bull's-eye to illustrate the way estimates can miss their "target." They may be systematically biased, or they may have high variability, or both. The best situation, of course, is when there is little systematic bias and little variability. Reprinted with permission from Kass et al. (2014), *Analysis of Neural Data*, copyright 2014, Springer Science+Business Media.

(essentially, first arrived at by Fisher in 1922, but made more precise and general by many subsequent authors in work running through the 1970s) is that, in large samples of data, the method of maximum likelihood (ML) minimizes MSE. In this sense, maximum likelihood estimators (MLEs) are optimal. In addition, it may be shown that Bayes estimators are also optimal in this sense, and in fact, MLEs and Bayes estimators are approximately equal (again, for large samples of data). A different, but also fundamental, mathematical result is that Bayes classifiers are optimal in the sense of minimizing the average number of misclassified observations.

One key qualification to keep in mind about these optimality results is that they hold true for random variables that follow specified probability distributions. That is, they hold when it is assumed that a random variable follows a specific probability distribution (so that all probabilities for values of the random variable become known once the value of the parameter θ is known). When a knowledgeable data analyst chooses not to use ML or Bayes estimation, or not to apply a Bayes classifier, it is because he or she is worried that the probabilistic assumptions needed to justify optimality may be highly inaccurate representations of the variation in the data. Other procedures can be shown to perform relatively well, in certain circumstances, with less restrictive assumptions.

Confidence Intervals and the Bootstrap

The idea of a confidence interval

By themselves, estimates are of little value without some notion of their accuracy. Theory shows that in many cases, the squared bias in Equation 2 is much smaller than the variance, so the variance represents accuracy. The square root of the estimator variance has the same units as the estimator and the parameter being estimated. It is called the standard error (SE) of the estimator:

$$SE = \sqrt{V(T)}$$

This definition of SE should not be confused with the standard error of mean (SEM), which is a special case. In fact, the SEM is applied so frequently that many people use SE to refer only to this special case. Here, however, I am using the general statistical terminology for SE.

In the many common situations where SE summarizes accuracy, it also typically happens that the estimator T is approximately normally distributed, for large samples. Specifically, because the bias is small, this means that T follows, approximately, a normal distribution with mean θ and SD SE. When a random variable X follows a normal distribution with mean μ and SD σ, the probability that X will take a value in the interval $(\mu - 2\sigma, \mu + 2\sigma)$ is 0.95 (rounding to two digits). Therefore, in these common situations, for large samples, there will be a probability of 0.95 that T will fall within 2SE of the target value θ. Now writing the estimator as $\hat{\theta} = T$, we have the formulation of the usual ~95% CI:

$$\sim 95\%\ CI = (\hat{\theta} - 2SE, \hat{\theta} + 2SE).$$

This is the way we obtained intervals reported at the end of Examples 1 and 2. See Kass et al., 2014, Chapter 7, for further discussion of CIs.

Getting a distribution via computer simulation

In standard situations, there are readily available formulas for SE that are furnished by statistical software. However, as problems become somewhat more complicated, or if the quantity being estimated is not one considered by the software developers, it is necessary to obtain SE a different way. Often the most convenient method is to apply computer simulation. To write down the algorithm, we have to use the formula for T, i.e., the formula used to compute the value of T from the data (as in the formula for the sample mean used earlier). As a general notation, let us write this formula as a function $h(x_1, x_2, \ldots, x_n)$ so that $T = h(X_1, X_2, \ldots, X_n)$ where X_i is the random variable representing the ith data value. I am assuming that all of the X_i random variables have the same distribution (and that they are statistically independent), which I will call the distribution of X. Here is the algorithm:

(1) For $g = 1$ to G,

Generate a sample $U_1^{(g)}, U_2^{(g)}, \ldots, U_n^{(g)}$ based on the distribution of X.

Compute $W^{(g)} = h(U_1^{(g)}, U_2^{(g)}, \ldots, U_n^{(g)})$

(2) Compute $\overline{W} = \frac{1}{G}\sum_{i=1}^{G} W^{(g)}$, and then

$$SE_{sim}(T) = \sqrt{\frac{1}{G-1}\sum_{g=1}^{G}(W^{(g)} - \overline{W})^2}.$$

Step 1 of this scheme would evaluate the estimator T on all the sets of "pseudo-data" $U_1^{(g)}, U_2^{(g)}, \ldots, U_n^{(g)}$ for $g = 1,\ldots,G$. Each set of simulated values $U_1^{(g)}, U_2^{(g)}, \ldots, U_n^{(g)}$ may also be called a "sample of pseudo-data."

NOTES

The squared value $SE_{sim}(T)^2$ is simply the sample variance of the $W^{(g)}$ random variables, and for large G, it would become close to the variance $V(T)$. Thus, for large G, we would get $SE_{sim}(T)$ as an approximate value of the desired SE.

The "plug-in" idea of the parametric bootstrap

The only problem with the algorithm for computing SE (approximately) given above is that, even when we assume the distribution of X has a particular distribution that depends on a parameter θ, we cannot yet simulate from this distribution until we pick a value of θ. However, it may be shown that if we use the value $\theta = \hat{\theta}$, where θ is the MLE, then the resulting SE is approximately correct, and thus the ~95% CI given above will have the correct probability ~0.95 of containing the unknown parameter value θ.

The nonparametric bootstrap

In some situations, we may have trouble writing down a probability distribution that we think will do a good job representing variability in the data. In addition, sometimes even when we do have such a probability distribution, it may be cumbersome to write code to simulate from this distribution. Often, it is possible to do something different, which is known as the nonparametric "bootstrap." The key idea requires the notion of the "distribution function" of a random variable X, written $F_X(x)$, where the lowercase x is a possible value of the random variable X. The value $F_X(x)$ is the probability that X is less than or equal to the number x. Suppose we have a random sample $X_1, X_2, \ldots, X_n$ from a distribution having distribution function $F_X(x)$. We can resample these variables by picking one of them at random, with equal probability of drawing each, then repeating by picking another (again, from the original set $X_1, X_2, \ldots, X_n$ with equal probability of drawing each, so that it is possible to pick the same variable twice), and then picking another, etc., until we again have n values. Typically we will have a different set of n variables than the original set because some of them will be repeated and some will be missing. In general, this kind of sampling is called "sampling with replacement" (because we replace each value we pick at random before picking another one), and here, with resampling, it becomes "bootstrap sampling." To each of the bootstrap samples we can apply the estimator T. It turns out that (for large n) the distribution of T applied to bootstrap samples will be nearly the same as the correct distribution of T from the distribution having distribution function $F_X(x)$. This means we can define a relatively simple and general algorithm for approximating SE. Again letting $T = h(X_1, \ldots, X_n)$ to get the nonparametric bootstrap SE = SE(T), we proceed as follows:

(1) For $g = 1$ to G,

Generate a sample $U_1^{(g)}, U_2^{(g)}, \ldots, U_n^{(g)}$, by resampling, with replacement, the observations $x_1, \ldots, x_n$.

Compute $T^{(g)} = h(U_1^{(g)}, U_2^{(g)}, \ldots, U_n^{(g)})$

(2) Compute $\overline{T} = \frac{1}{G}\sum_{i=1}^{G} T^{(g)}$, and then

$$SE(T) = \sqrt{\frac{1}{G-1}\sum_{g=1}^{G}(T^{(g)} - \overline{T})^2}.$$

The bootstrap is useful in situations where we have to write computer code to get an estimator T, the code being represented above as a computation $T = h(X_1, \ldots, X_n)$. To obtain SE, we apply the same code to our resampled data, as specified in the algorithm above.

One caveat is that arbitrary "shuffles" of the data do not necessarily correspond to bootstrap samples. The fundamental assumption is that the data values being resampled are observations from random variables (more generally, random vectors) that have the same distribution and are independent of each other. In many situations, it takes extra work to figure out how to sample in such a way that this fundamental assumption is reasonable. Additional remarks about bootstrap methods may be found in Kass et al., 2014, Chapter 9.

Conclusion

Beginnning with a summary of the statistical paradigm in the form of two basic tenets, I have tried to emphasize the notion of a statistical model, which involves a theoretical abstraction based on probability distributions for random variables that aim to describe variation in data. Advanced statistical training not only provides students with knowledge of the inner workings of many statistical tools and procedures, it also indoctrinates them with the idea that there are principled approaches to data analysis and that the principles always invoke statistical models. I focused on CIs and bootstrap CIs partly because they are important, but also because they are derived from principles (although I was able to mention these principles only briefly). I hope a much deeper understanding will come from studying carefully the material in *Analysis of Neural Data*.

Acknowledgments

This chapter was excerpted with permission from Robert E. Kass, Uri T. Eden, and Emery M. Brown, Analysis of Neural Data, part of the *Springer Series in Statistics*, copyright 2014, Springer Science+Business Media New York.

References

Fisher RA (1922) On the mathematical foundations of theoretical statistics. Philos Trans R Soc A 222:309–368.

Hursh JB (1939) Conduction velocity and diameter of nerve fibers. Am J Physiol 127:131–139.

Kass R, Eden UT, Brown EN (2014) Analysis of neural data. New York: Springer.

Marshall JC, Halligan PW (1988) Blindsight and insight in visuo-spatial neglect. Nature 336:766–768.

NOTES

Preface to Chapters by Jonathan Pillow, PhD

In the following two Short Course chapters, we will discuss more advanced statistical techniques for modeling neural spike trains. The Poisson generalized linear model (GLM) is a nonlinear regression model for identifying the relationship between external and internal covariates of the response and instantaneous probability of spiking in a single neuron. In the chapter entitled "Likelihood-Based Approaches to Modeling the Neural Code," we will discuss the basic formulation of GLMs and techniques for estimating their parameters from data.

In the chapter that follows, "Spatiotemporal Correlations and Visual Signaling in a Complete Neuronal Population," we will discuss extensions that allow the GLM to incorporate dependencies on spike history, as well as the spike histories of other neurons in a multineuron recording. Incorporating spike history gives the model the ability to capture a rich spectrum of non-Poisson spiking behaviors, including refractoriness, bursting, adaptation, regular and irregular firing, bistability, and type I and type II firing-rate curves. Incorporating dependencies on the spike history of other neurons in the population gives the model the ability to identify functional connectivity and capture rich spatiotemporal noise correlations among neurons. Lastly, we will discuss techniques for regularization, allowing model parameters and connectivity to be accurately estimated from limited data.

Likelihood-Based Approaches to Modeling the Neural Code

Jonathan W. Pillow, PhD

Princeton Neuroscience Institute
Princeton, New Jersey

NOTES

Introduction

One of the central problems in systems neuroscience is that of characterizing the functional relationship between sensory stimuli and neural spike responses. Investigators call this the "neural coding problem" because the spike trains of neurons can be considered a code by which the brain represents information about the state of the external world. One approach to understanding this code is to build mathematical models of the mapping between stimuli and spike responses; the code can then be interpreted by using the model to predict the neural response to a stimulus, or to decode the stimulus that gave rise to a particular response. In this chapter, we will examine "likelihood-based" approaches, which use the explicit probability of response to a given stimulus for both fitting the model and assessing its validity. We will show how the likelihood can be derived for several types of neural models, and discuss theoretical considerations underlying the formulation and estimation of such models. Finally, we will discuss several ideas for evaluating model performance, including time-rescaling of spike trains and optimal decoding using Bayesian inversion of the likelihood function.

The Neural Coding Problem

Neurons exhibit stochastic variability. Even for repeated presentations of a fixed stimulus, a neuron's spike response cannot be predicted with certainty. Rather, the relationship between stimuli and neural responses is probabilistic. Understanding the neural code can therefore be framed as the problem of determining $p(y \mid x)$: the probability of response y conditional on a stimulus x. For a complete solution, we need to be able compute $p(y \mid x)$ for any x, meaning a description of the full response distribution for any stimulus we might present to a neuron. Unfortunately, we cannot hope to get very far trying to measure this distribution directly, owing to the high dimensionality of stimulus space (e.g., the space of all natural images) and the finite duration of neurophysiology experiments. Figure 1 illustrates the general problem.

A classical approach to the neural coding problem has been to restrict attention to a small, parametric family of stimuli (e.g., flashed dots, moving bars, or drifting gratings). The motivation underlying this approach is the idea that neurons are sensitive only to a restricted set of "stimulus features" and that we can predict the response to an arbitrary stimulus simply by knowing the response to these features. If $x_{\{\psi\}}$ denotes a parametric set of features to which a neuron modulates its response, then the classical approach posits that $p(y \mid x) \approx p(y \mid x_\psi)$, where x_ψ is the stimulus feature that most closely resembles x.

Although the "classical" approach to neural coding is not often explicitly framed in this way, it is not so different in principle from the "statistical modeling" approach that has gained popularity in recent years,

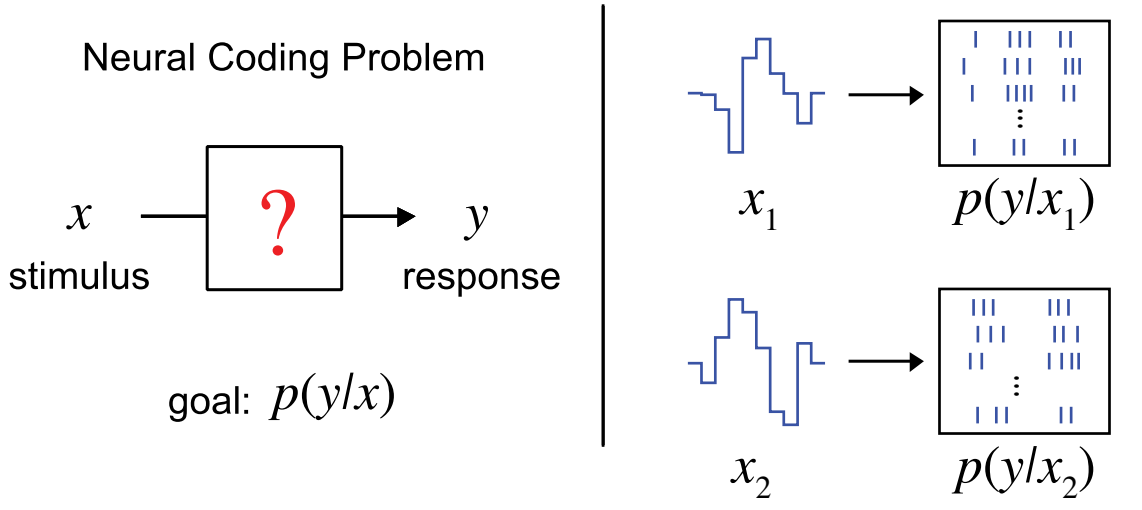

Figure 1. Illustration of the neural coding problem. The goal is to find a model mapping x to y that provides an accurate representation of the conditional distribution $p(y \mid x)$. Right, Simulated distribution of neural responses to two distinct stimuli, x_1 and x_2 illustrating (1) stochastic variability in responses to a single stimulus and (2) that the response distribution changes as a function of x. A complete solution involves predicting $p(y \mid x)$ for any x.

NOTES

and which we pursue here. In this framework, we assume a probabilistic model of the neural response and attempt to fit the model parameters θ so that $p(y \mid x, \theta)$—the response probability under the model, provides a good approximation to $p(y \mid x)$. Although the statistical approach is often applied using stimuli drawn stochastically from a high-dimensional ensemble (e.g., Gaussian white noise) rather than a restricted parametric family (e.g., sine gratings), the goals are essentially similar: to find a simplified and computationally tractable description of $p(y \mid x)$. The statistical framework differs primarily in its emphasis on detailed quantitative prediction of spike responses, and in offering a unifying mathematical framework (likelihood) for fitting and validating models.

Model Fitting with Maximum Likelihood

Let us now turn to the problem of using likelihood for fitting a model of an individual neuron's response. Suppose we have a set of stimuli $\boldsymbol{x} = \{x_i\}$ and a set of spike responses $\boldsymbol{y} = \{y_i\}$ obtained during a neurophysiology experiment, and we would like to fit a model that captures the mapping from $\boldsymbol{x}$ to $\boldsymbol{y}$. Given a particular model, parametrized by the vector θ, we can apply a tool from classical statistics known as "maximum likelihood" (ML) to obtain an asymptotically optimal estimate of θ. For this, we need an algorithm for computing $p(\boldsymbol{y} \mid \boldsymbol{x}, \theta)$, which, considered as a function of θ, is called the "likelihood" of the data. The ML estimate $\hat{\theta}$ is the set of parameters under which these data are most probable, or the maximizer of the likelihood function:

$$\hat{\theta} = \arg\max_{\theta} p(\boldsymbol{y} \mid \boldsymbol{x}, \theta). \qquad (1)$$

Although this solution is easily stated, it is unfortunately the case that for many models of neural response (e.g., detailed biophysical models such as Hodgkin–Huxley) it is difficult or impossible to compute likelihood. Moreover, even when we can find simple algorithms for computing likelihood, maximizing it can be quite difficult; in most cases, θ lives in a high-dimensional space, containing tens to hundreds of parameters (e.g., describing a neuron's receptive field and spike-generation properties). Such nonlinear optimization problems are often intractable.

In the following sections, we will introduce several probabilistic neural spike models, derive the likelihood function for each model, and discuss the factors affecting ML estimation of its parameters. We will also compare ML with standard (e.g., moment-based) approaches to estimating model parameters.

The linear–nonlinear–Poisson model

One of the best-known models of neural response is the linear–nonlinear–Poisson (LNP) model, which is alternately referred to as the linear–nonlinear "cascade" model. The model, which is schematized in the left panel of Figure 2, consists of a linear filter (k), followed by a point nonlinearity (f), followed by Poisson spike generation. Although many interpretations are possible, a simple description of the model's components holds that:

- k represents the neuron's space–time receptive field, which describes how the stimulus is converted to intracellular voltage;
- f describes the conversion of voltage to an instantaneous spike rate, accounting for such nonlinearities as rectification and saturation;
- instantaneous rate is converted to a spike train via an inhomogeneous Poisson process.

The parameters of this model can be written as $\theta = \{k, \varphi_f\}$, where φ_f are the parameters governing f. Although the LNP model is not biophysically realistic (especially the assumption of Poisson spiking), it provides a compact and reasonably accurate description of average responses, e.g., peristimulus time histogram (PSTH), in many early sensory areas.

Another reason for the popularity of the LNP model is the existence of a simple and computationally efficient fitting algorithm, which consists of using spike-triggered average (STA) as an estimate for k and a simple histogram procedure to estimate φ_f (Bryant and Segundo, 1976; Chichilnisky, 2001). It is a well-known result that the STA (or "reverse correlation") gives an unbiased estimate of the direction of k (i.e., the STA converges to αk, for some unknown α) if the raw stimulus distribution $p(x)$ is spherically symmetric, and f shifts the mean of the spike-triggered ensemble away from zero (i.e., the expected STA is not the zero vector) (Bussgang, 1952; Paninski, 2003). However, the STA does *not* generally provide an optimal estimate of k, except in a special case we will examine in more detail below (Paninski, 2004).

First, we derive the likelihood function of the LNP model. The right panel of Figure 2 shows the dependency structure (also known as a graphical

NOTES

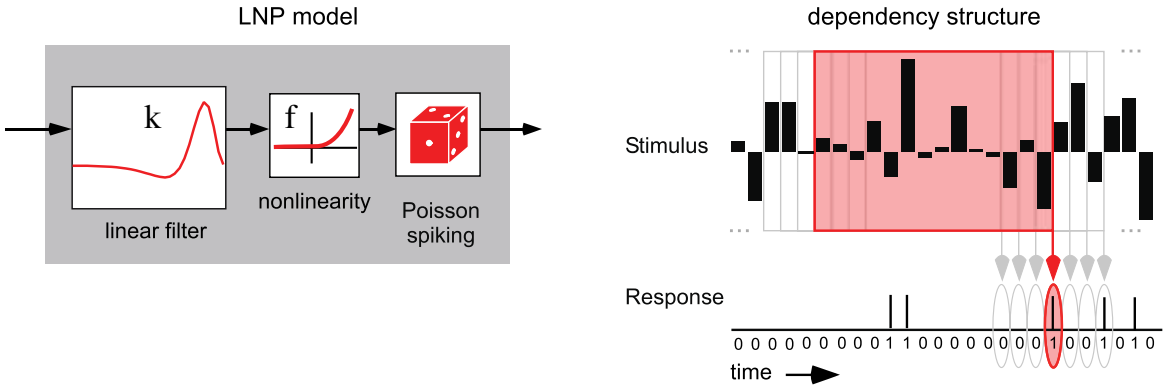

Figure 2. Schematic and dependency structure of the linear–nonlinear–Poisson (LNP) model. Left, LNP model consists of a linear filter k, followed by a point nonlinearity f, followed by Poisson spike generation. Right: Depiction of a discretized white noise Gaussian stimulus (above) and spike response (below). Arrows indicate the causal dependency entailed by the model between portions of the stimulus and portions of the response. The highlighted gray box and gray oval show this dependence for a single time bin of the response, while gray boxes and arrows indicate the (time-shifted) dependency for neighboring bins of the response. As indicated by the diagram, all time bins of the response are conditionally independent, given the stimulus (Eq. 2).

model) between stimulus and response, where arrows indicate conditional dependence. For this model, the bins of the response are conditionally independent of one another, given the stimulus—an essential feature of Poisson processes. This means that the probability of the entire spike train factorizes as

$$p(\boldsymbol{y} \mid \boldsymbol{x}, \theta) = \prod_i p(y_i \mid x_i, \theta), \qquad (2)$$

where y_i is the spike count in the ith time bin, and x_i is the stimulus vector causally associated with this bin. Equation 2 asserts that the likelihood of the entire spike train is the product of the single-bin likelihoods. Under this model, single-bin likelihood is given by the Poisson distribution with rate parameter $\Delta f(k \cdot x_i)$, where $k \cdot x_i$, is the dot product of k with x_i, and Δ is the width of the time bin. The probability of having y_i spikes in the ith bin is therefore

$$p(y_i \mid x_i, \theta) = \frac{1}{y_i!}[\Delta f(k \cdot x_i)]^{y_i} e^{-\Delta f(k \cdot x_i)}, \qquad (3)$$

and the likelihood of the entire spike train can be rewritten as:

$$p(\boldsymbol{y} \mid \boldsymbol{x}, \theta) = \Delta^n \prod_i \frac{f(k \cdot x_i)^{y_i}}{y_i!} e^{-\Delta f(k \cdot x_i)}, \qquad (4)$$

where n is the total number of spikes.

We can find the ML estimate $\hat{\theta} = \{\hat{k}, \hat{\varphi}\}$ by maximizing the log of the likelihood function (which is monotonically related to likelihood), and given by

$$\log p(\boldsymbol{y} \mid \boldsymbol{x}, \theta) = \sum_i y_i \log f(k \cdot x_i) - \Delta \sum_i f(k \cdot x_i) + c, \qquad (5)$$

where c is a constant that does not depend on k or f. Because there is an extra degree of freedom between the amplitude of k and input scaling of f, we can constrain k to be a unit vector, and consider only the angular error in estimating k. By differentiating the log-likelihood with respect to k and setting it to zero, we find that the ML estimate satisfies:

$$\lambda \hat{k} = \sum_i y_i \frac{f'(\hat{k} \cdot x_i)}{f(\hat{k} \cdot x_i)} x_i - \Delta \sum_i f'(\hat{k} \cdot x_i) x_i, \qquad (6)$$

where λ is a Lagrange multiplier introduced to constrain k to be a unit vector. As noted in Paninski (2004), the second term on the right hand converges to a vector proportional to k if the stimulus distribution $p(x)$ is spherically symmetric. (It is the expectation over $p(x)$ of a function radially symmetric around k.) If we replace this term by its expectation, we are left with just the first term, which is a weighted STA, since y_i is the spike count and x_i is the stimulus preceding the ith bin. This term is proportional to the (ordinary) STA if f'/f is constant, which occurs only when $f(z) = e^{az+b}$.

Therefore, the STA corresponds to the ML estimate for k whenever f is exponential; conversely, if f differs significantly from exponential, Equation 6 specifies a different weighting of the spike-triggered stimuli, and

NOTES

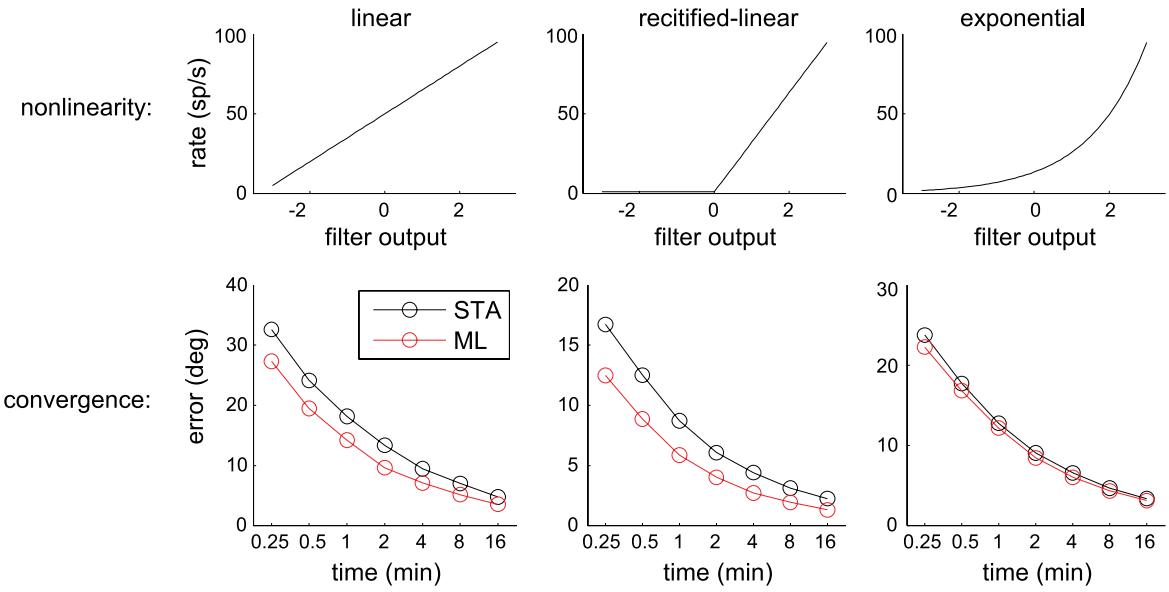

Figure 3. Comparison of STA and ML estimates of the linear filter k in an LNP model. Top row, three different types of nonlinearity f: a linear function (left), a half-wave rectified linear function (middle), and an exponential function. For each model, the true k was a 20-tap temporal filter with biphasic shape similar to that found in retinal ganglion cells. The stimulus was temporal Gaussian white noise with a frame rate of 100 Hz, and k was normalized so that filter output had unit SD. Bottom row, Plots show the convergence behavior for each model as a function of the amount of data collected. Error is computed as the angle between the estimate and the true k, averaged more than 100 repeats at each stimulus length. deg: degrees; sp/s: spikes per second.

the traditional STA is suboptimal. Figure 3 illustrates this point with a comparison between the STA and the ML estimate for k on spike trains simulated using three different nonlinearities. In the simulations, we found the ML estimate by directly maximizing log-likelihood (Eq. 5) for both k and φf, beginning with the STA as an initial estimate for k. As expected, the ML estimate outperforms the STA except when f is exponential (rightmost column).

Figure 4 shows a similar analysis comparing ML with an estimator derived from spike-triggered covariance (STC) analysis, which uses the principal eigenvector of the STC matrix to estimate k. Recent work has devoted much attention to fitting LNP models with STC analysis, which is relevant particularly in cases where the f is approximately symmetric (de Ruyter van Steveninck and Bialek, 1988; Schwartz et al., 2002; Touryan et al., 2002; Aguera y Arcas and Fairhall, 2003; Simoncelli et al., 2004; Bialek and de Ruyter van Steveninck, 2005; Schwartz et al., 2006). The left column of Figure 4 shows a simulation where f is a quadratic, shifted slightly from the origin so that both the STA and the first eigenvector of the STC provide consistent (asymptotically convergent) estimates of k. Both, however, are significantly outperformed by the ML estimator. Although it is beyond the scope of this chapter, a derivation similar to the one above shows that there is an f for which the ML estimator and the STC estimate are identical. The relevant f is a quadratic in the argument of an exponential, which can also be represented as a ratio of two Gaussians (Pillow and Simoncelli, 2006). The right column of Figure 4 shows results obtained with such a nonlinearity. If we used a similar nonlinearity in which the first term of the quadratic is negative, e.g., $f(x) = \exp(-x^2)$, then f produces a reduction in variance along k, and the STC eigenvector with the smallest eigenvalue is comparable with the ML estimate (Pillow and Simoncelli, 2006).

Before closing this section, it is useful to review several other general characteristics of ML estimation in LNP models. First, note that the LNP model can be generalized to include multiple linear filters and a multidimensional nonlinearity, all of which can be fit using ML. In this case, the likelihood function is the same as in Equation 4, only the instantaneous spike rate is now given by:

$$\text{rate}(x_i) = f(k_1 \cdot x_i, k_2 \cdot x_i, \ldots, k_m \cdot x_i), \qquad (7)$$

NOTES

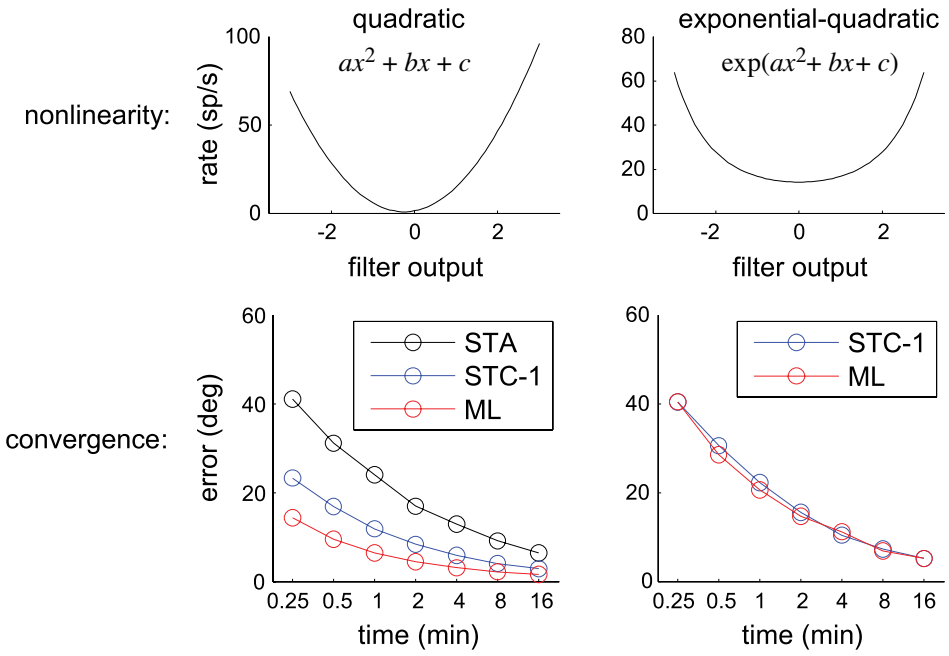

Figure 4. Comparison of STA, STC, and ML estimates of k in an LNP model. Top row, Two types of nonlinearity functions used to generate responses; a quadratic function (left), and a quadratic raised to an exponential (right). Stimulus and true k as in Fig. 3. Bottom row, Convergence behavior of the STA, first (maximum-variance) eigenvector of the STC, and ML estimate. The STA is omitted from the right plot, as it fails to converge under a symmetric nonlinearity. deg: degrees; sp/s: spikes per second.

where $\{k_1, k_2, \ldots, k_m\}$ is a collection of filters, and f is an m-dimensional point nonlinearity. Second, ML estimation of the LNP model enjoys the same statistical advantages as several information-theoretic estimators that have been derived for finding "maximally informative dimensions" or features of the stimulus space (Paninski, 2003; Sharpee et al., 2004). Specifically, the ML estimator is unbiased even when the raw stimulus distribution lacks spherical symmetry (e.g., "naturalistic stimuli"), and it is sensitive to higher-order statistics of the spike-triggered ensemble, making it somewhat more powerful and more general than STA or STC analysis. Unfortunately, ML also shares the disadvantages of these information-theoretic estimators: it is computationally intensive, difficult to use for recovering multiple (e.g., > 2) filters (in part due to the difficulty of choosing an appropriate parametrization for f), and cannot be guaranteed to converge to the true maximum using gradient ascent, owing to the existence of multiple local maxima in the likelihood function.

We address this last shortcoming in the next two sections, which discuss models constructed to have likelihood functions that are free from suboptimal local maxima. These models also introduce dependence of the response on spike-train history, eliminating a second major shortcoming of the LNP model: the assumption of Poisson spike generation.

The generalized linear model

The generalized linear model (GLM), schematized in Figure 5, generalizes the LNP model to incorporate feedback from the spiking process, allowing the model to account for history-dependent properties of neural spike trains such as the refractory period, adaptation, and bursting (Paninski, 2004; Truccolo et al., 2004). As shown in the dependency diagram (right panel, Fig. 5), the responses in distinct time bins are no longer conditionally independent, given the stimulus; rather, each bin of the response depends on some time window of the recent spiking activity. Luckily, this does not prevent us from factorizing the likelihood, which can now be written as

$$p(\boldsymbol{y} \mid \boldsymbol{x}, \theta) = \prod_i p(y_i \mid x_i, y_{[i-k:i-1]}\theta), \qquad (8)$$

where $y_{[i-k:i-1]}$ is the vector of recent spiking activity from time bin $i - k$ to $i - 1$. This factorization holds because, by Bayes' rule, we have

$$p(y_i, y_{[i-k:i-1]} \mid \boldsymbol{x}, \theta) = p(y_i \mid y_{[i-k:i-1]}, \boldsymbol{x}, \theta) p(y_{[i-k:i-1]} \mid \boldsymbol{x}, \theta), \qquad (9)$$

NOTES

and we can apply this formula recursively to obtain Equation 8. (Note, however, that no such factorization is possible if we allow loopy, e.g., bidirectional, causal dependence between time bins of the response.)

Except for the addition of a linear filter, h, operating on the neuron's spike-train history, the GLM is identical to the LNP model. We could therefore call it the "recurrent LNP" model, although its output is no longer a Poisson process, owing to the history-dependence induced by h. The GLM likelihood function is similar to that of the LNP model. If we let

$$r_i = f(k \cdot x_i + h \cdot y_{[i-k:i-1]}) \qquad (10)$$

denote the instantaneous spike rate (or "conditional intensity" of the process), then the likelihood and log-likelihood (following Eq. 4 and 5), respectively, are given by:

$$p(\mathbf{y}\,\mathbf{x}, \theta) = \Delta^n \prod_i \frac{r_i^{y_i}}{y_i!} e^{-\Delta r_i} \qquad (11)$$

$$\log p(\boldsymbol{y} \mid \boldsymbol{x}, \theta) = \sum_i y_i \log r_i - \Delta \sum_i r_i + c. \qquad (12)$$

Unfortunately, we cannot use moment-based estimators (STA and STC) to estimate k and h for this model, because the consistency of those estimators relies on spherical symmetry of the input (or "Gaussianity," for STC), which the spike-history input term $y_{[i-k:i-1]}$ fails to satisfy (Paninski, 2003).

As mentioned above, a significant shortcoming of the ML approach to neural characterization is that it may be quite difficult in practice to find the ML function. Gradient ascent fails if the likelihood function is rife with local maxima, and more robust optimization techniques (e.g., simulated annealing) are computationally exorbitant and require delicate oversight to ensure convergence.

One solution to this problem is to constrain the model so that we guarantee that the likelihood function is that of free-from (nonglobal) local maxima. If we can show that the likelihood function is "log-concave," meaning that the negative log-likelihood function is convex, then we can be assured that the only maxima are global maxima. Moreover, the problem of computing the ML estimate $\hat{\theta}$ is reduced to a convex optimization problem, for which there are tractable algorithms even in very high-dimensional spaces.

As shown by Paninski (2004), the GLM has a concave log-likelihood function if the nonlinearity f is itself convex and log-concave. These conditions are satisfied if the second derivative of f is nonnegative and the second derivative of $\log f$ is non-positive. Although this may seem like a restrictive set of conditions (e.g., it rules out symmetric nonlinearities), a number of suitable functions seem like reasonable choices for describing the conversion of intracellular voltage to instantaneous spike rate, for example:

- $f(z) = \max(z + b, 0)$
- $f(z) = e^{z+b}$
- $f(z) = \log(1 + e^{z+b})$,

where b is a single parameter that we also estimate with ML.

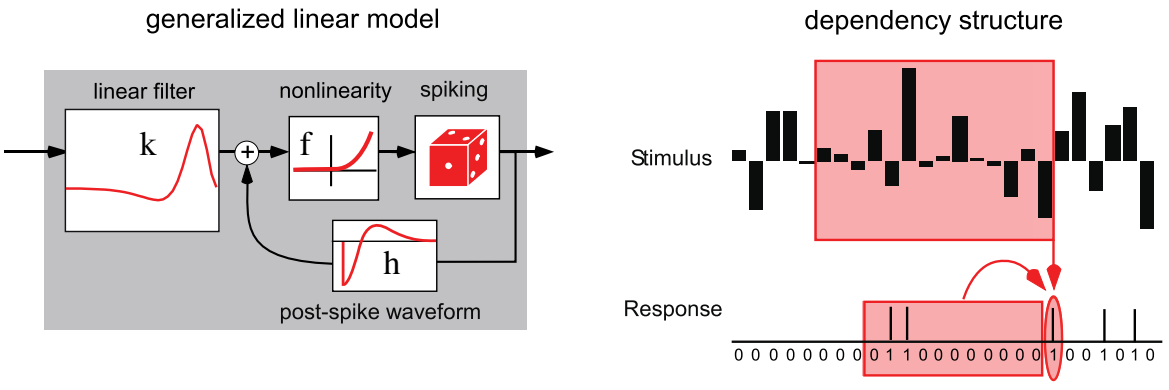

Figure 5. Diagram and dependency structure of a GLM. Left, Model schematic, showing the introduction of history-dependence in the model via a feedback waveform from the spiking process. In order to ensure convexity of the negative log-likelihood, we now assume that the nonlinearity f is exponential. Right, Graphical model of the conditional dependencies in the GLM. The instantaneous spike rate depends on both the recent stimulus and recent history of spiking.

NOTES

Thus, for appropriate choice of f, ML estimation of a GLM becomes computationally tractable. Moreover, the GLM framework is quite general and can easily be expanded to include additional linear filters that capture dependence on spiking activity in nearby neurons, behavior of the organism, or additional external covariates of spiking activity. ML estimation of a GLM has been successfully applied to the analysis of neural spike trains in a variety of sensory, motor, and memory-related brain areas (Chornoboy et al., 1988; Truccolo et al., 2004; Okatan et al., 2005; Pillow et al., 2005a).

Generalized integrate-and-fire model

We now turn our attention to a dynamical systems model of the neural response, for which the likelihood of a spike train is not so easily formulated in terms of a conditional intensity function (i.e., the instantaneous probability of spiking, conditional on stimulus and spike-train history). Recent work has shown that the leaky integrate-and-fire (IF) model, a canonical but simplified description of intracellular spiking dynamics, can reproduce the spiking statistics of real neurons (Reich et al., 1998; Keat et al., 2001). It can also mimic important dynamical behaviors of more complicated models like Hodgkin–Huxley (Gerstner and Kistler, 2002; Jolivet et al., 2003). It is therefore natural to ask whether likelihood-based methods can be applied to models of this type. Figure 6 shows a schematic diagram of the generalized IF model (Paninski, 2004; Pillow et al., 2005b), which is a close relative of the well-known spike response model (Jolivet et al., 2003). The model generalizes the classical IF model so that injected current is a linear function of the stimulus and spike-train history, plus a Gaussian noise current that introduces a probability distribution over voltage trajectories. The model dynamics (written here in discrete time, for consistency) are given by

$$\frac{v_{i+1} - v_i}{\Delta} = -\frac{1}{\tau}(v_i - v_L) + (k \cdot x_i) + (h \cdot y_{[i-k:i-1]}) + \sigma N_i \Delta^{-\frac{1}{2}}, \qquad (13)$$

where v_i is the voltage at the ith time bin, which obeys the boundary condition that whenever $v_i \geq 1$, a spike occurs and v_i is reset instantaneously to zero. Δ is the width of the time bin of the simulation, and N_i is a standard Gaussian random variable, drawn independently on each i. The model parameters k and h are the same as in the GLM: linear filters operating on the stimulus and spike-train history (respectively), and the remaining parameters are: τ, the time constant of the membrane leak; v_L, the leak current reversal potential; and σ, the amplitude of the noise.

The lower left panel of Figure 6 depicts the dependency structure of the model as it pertains to computing the likelihood of a spike train. In this case, we can regard the probability of an entire interspike interval (ISI) as depending on a relevant portion of the stimulus and spike-train history. The lower right panel illustrates how we might compute this likelihood for a single ISI under the generalized IF model using Monte Carlo sampling. Computing the likelihood in this case is also known as the "first-passage time" problem. Given a setting of the model parameters, we can sample voltage trajectories from the model, drawing independent noise samples for each trajectory and following each trajectory until it hits threshold. The gray traces show five such sample paths, while the blue trace shows the voltage path obtained in the absence of noise. The probability of a spike occurring at the ith bin is simply the fraction of voltage paths crossing threshold at this bin. The black trace above shows the probability distribution obtained by collecting the first passage times of a large number of paths. Evaluated at the actual spike, this density gives the likelihood of the relevant ISI. Because of voltage reset following a spike, all ISIs are conditionally independent, and we can again write the likelihood function as a product of conditionally independent terms:

$$p(\boldsymbol{y} \mid \boldsymbol{x}, \theta) = \prod_{t_j} p(y_{[t_{j-1}+1:t_j]} \mid \boldsymbol{x}, \boldsymbol{y}_{[0:t_j]}, \theta), \qquad (14)$$

where $\{t_j\}$ is the set of spike times emitted by the neuron, $y_{[t_j - 1 + 1 : t_j]}$ is the response in the set of time bins in the jth ISI, and $y_{[0:t_j]}$ is the response during time bins previous to that interval.

The Monte Carlo approach to computing likelihood of a spike train can in principle be performed for any probabilistic dynamical-systems–style model. In practice, however, such an approach would be unbearably slow and would likely prove intractable, particularly because the likelihood function must be computed many times in order find the ML estimate for θ. However, for the generalized IF model, there exists a much more computationally efficient method for computing the likelihood function using the Fokker–Planck equation. Although beyond the scope of this chapter, the method works by "density propagation" of a numerical representation of the probability density over subthreshold voltage, which can be quickly computed using sparse matrix methods. More important, results have shown that the log-likelihood function for the generalized IF model (like that of the GLM) is concave. This means that the likelihood function contains a unique global maximum, and that gradient ascent can be used to find the ML estimate of the model parameters

NOTES

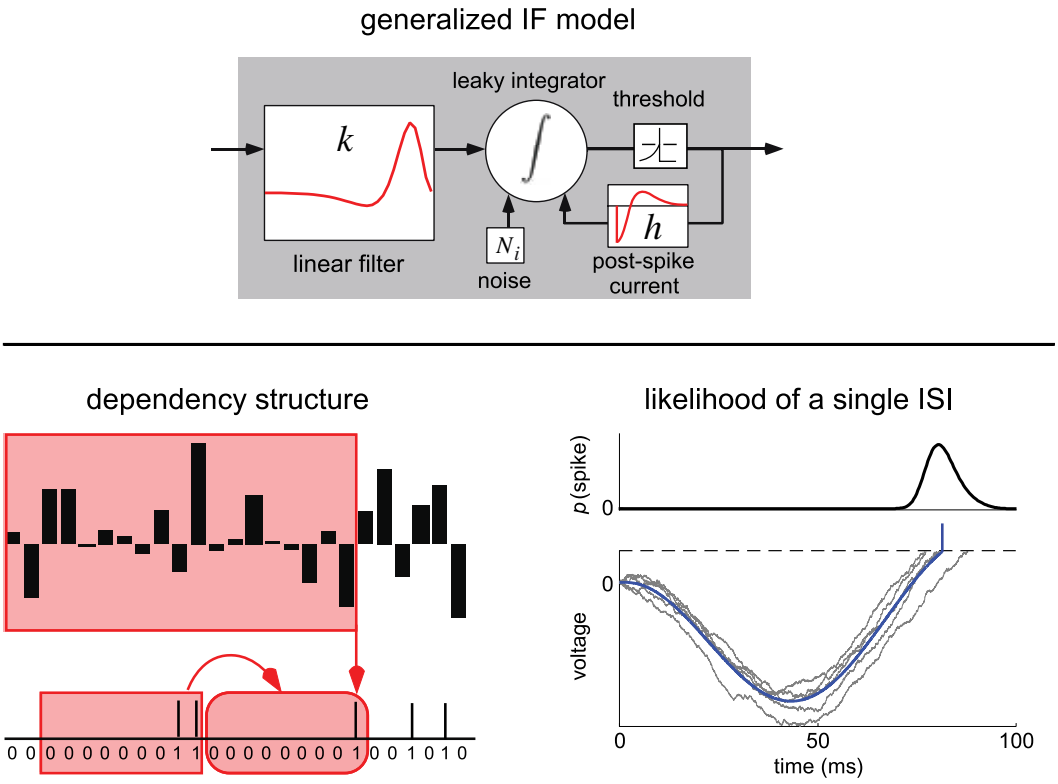

Figure 6. Generalized IF model. Top: Schematic diagram of model components, including a stimulus filter *k* and a postspike current *h* that is injected into a leaky integrator following every spike, and independent Gaussian noise to account for response variability. Bottom left: Graphical model of dependency structure, showing that the likelihood of each ISI is conditionally dependent on a portion of the stimulus and spike-train history prior to the ISI. Bottom right, Schematic illustrating how likelihood could be computed with Monte Carlo sampling. Blue trace shows the voltage (and spike time) from simulating the model without noise, while gray traces show sample voltage paths (to the first spike time) with noise. The likelihood of the ISI is shown above, as a function of the spike time (black trace). Likelihood of an ISI is equal to the fraction of voltage paths crossing threshold at the true spike time.

(Paninski et al., 2004). Other work has applied the generalized IF model to the responses of macaque retinal ganglion cells using ML, showing that the model can be used to capture stimulus dependence, spike-history dependence, and noise statistics of neural responses recorded *in vitro* (Pillow et al., 2005b).

Model Validation

Once we have a used ML to fit a particular model to a set of neural data, there remains the important task of validating the quality of the model fit. In this section, we discuss three simple methods for assessing the goodness-of-fit of a probabilistic model using the same statistical framework that motivated our approach to fitting.

Likelihood-based cross-validation

Recall that the basic goal of our approach is to find a probabilistic model such that we can approximate the true probabilistic relationship between stimulus and response, $p(y \mid x)$, by the model-dependent $p(y \mid x, \theta)$. Once we have fit θ using a set of training data, how can we tell if the model provides a good description of $p(y \mid x)$? To begin with, let us suppose that we have two competing models, p_A and p_B, parametrized by θ_A and θ_B, respectively, and we wish to decide which model provides a better description of the data. Unfortunately, we cannot simply compare the likelihood of the data under the two models, $p_A(y \mid x, \theta_A)$ versus $p_B(y \mid x, \theta_B)$, owing to the problem of "overfitting." Even though one model assigns the fitted data a higher likelihood than the other, it may not generalize as well to new data.

As a toy example of the phenomenon of overfitting, consider a dataset consisting of five points drawn from a Gaussian distribution. Let model A be a single Gaussian distribution, fit with the mean and SD of the sample points (i.e., the ML estimate for this model).

NOTES

For model B, suppose that the data come from a mixture of five very narrow Gaussian distributions, and fit this model by centering one of these narrow Gaussian distributions at each of the five sample points. Clearly, the second model assigns higher likelihood to the data (because it concentrates all probability mass near the sample points), but it fails to generalize; as a result, it will assign very low probability to new data points drawn from the true distribution that do not happen to lie very near the five original samples.

This suggests a general solution to the problem of comparing models, which goes by the name "cross-validation." Under this procedure, we generate a new set of "test" stimuli, $\boldsymbol{x}^\star$, and present them to the neuron to obtain a new set of spike responses, $\boldsymbol{y}^\star$. (Alternatively, we could set aside a small portion of the data at the beginning.) By comparing the likelihood of these new data sets under the two models, $p_A(\boldsymbol{y}^\star \mid \boldsymbol{x}^\star, \theta_A)$ versus $p_B(\boldsymbol{y}^\star \mid \boldsymbol{x}^\star, \theta_B)$, we get a fair test of the models' generalization performance. Note that, under this comparison, we do not actually care about the number of parameters in the two models: increasing the number of parameters in a model does not actually improve its ability to generalize. (In the toy example above, model B has more parameters but generalizes much more poorly. We can view techniques like regularization as methods for reducing the effective number of parameters in a model so that overfitting does not occur.) Although we may prefer a model with fewer parameters for aesthetic or computational reasons, from a statistical standpoint, we should care only about which model provides a better account of the novel data.

Time-rescaling

Another powerful idea for testing the validity of a probabilistic model is to use the model to convert spike times into a series of independent and identically distributed random variables. This conversion will be successful only if we have accurately modeled the probability distribution of each spike time. This idea, which goes under the name "time-rescaling" (Brown et al., 2002), is a specific application of the general result that we can convert any random variable into a uniform random variable using its cumulative density function (CDF).

First, let us derive the CDF of a spike time under the LNP and GLM models. If r_i is the conditional intensity function of the ith time bin (i.e., $f(k \cdot x_i)$ under the LNP model), then the probability that the "next" spike t_{j+1} occurs on or before bin k, given that the previous spike occurred at t_j, is simply 1 minus the probability that no spikes occur during the time bins $t_j + 1$ to k. This gives

$$p(t_{j+1} \le k \mid t_j) = 1 - \left(\prod_{i \in [t_j + 1,\, k]} e^{-\Delta r_i} \right), \quad (15)$$

which we can rewrite:

$$p(t_{j+1} \le k \mid t_j) = 1 - \exp\left(-\Delta \sum_{t_j + 1}^{k} r_i \right). \quad (16)$$

Note that the argument of the exponential is simply the negative integral of the intensity function since the time of the previous spike.

For the generalized IF model, computing the likelihood function involves computing the probability density function (PDF) over each interspike interval (as depicted in Fig. 6), which we can simply integrate to obtain the CDF (Paninski et al., 2004).

Given the CDF for a random variable, a general result from probability theory holds that it provides a remapping of that variable to the one randomly distributed unit interval [0, 1]. Even though the CDF for each spike time is different, if we remap the entire spike train using $t_j \rightarrow C\,DF_j(t_j)$, where $C\,DF_j$ is the cumulative density of the jth spike time, then, if the model is correct, we should obtain a series of independent, uniform random variables. This suggests we test the validity of the model by testing the remapped spike times for independence; any correlation (or some other form of dependence) between successive pairs of remapped spike times, for example, indicates a failure of the model. We can also examine the marginal distribution of the remapped times (e.g., using a Kolmogorov–Smirnov test) to detect deviations from uniformity. The structure of any deviations may be useful for understanding the model's failure modes: an excess of small-valued samples, for example, indicates that the model predicts too few short interspike intervals. If we wish to compare multiple models, we can use time-rescaling to examine which model produces the most nearly independent and most nearly uniform remapped spike times.

Model-based decoding

A third tool for assessing the validity of a probabilistic model is to perform stimulus decoding using the model-based likelihood function. Given the fitted model parameters, we can derive the posterior probability distribution over the stimulus given a spike train by inverting the likelihood function using the Bayes' rule:

$$p(\boldsymbol{x} \mid \boldsymbol{y}, \theta) = \frac{p(\boldsymbol{y} \mid \boldsymbol{x}, \theta) p(\mathbf{x})}{p(\boldsymbol{y} \mid \theta)}, \quad (17)$$

NOTES

where $p(x)$ is the prior probability of the stimulus (which we assume to be independent of θ), and the denominator is the probability of response y, given θ. We can obtain the most likely stimulus to have generated the response y by maximizing the posterior for x, which gives the maximum a posteriori (MAP) estimate of the stimulus, which we can denote

$$\hat{\boldsymbol{x}}_{MAP} = \arg\max_{x} p(\boldsymbol{y} \mid \boldsymbol{x}, \theta) p(\boldsymbol{x}) \qquad (18)$$

since the denominator term $p(\boldsymbol{y} \mid \theta)$ does not vary with $\boldsymbol{x}$.

For the GLM and generalized IF models, the concavity of the log-likelihood function with respect to the model parameters also extends to the posterior with respect to the stimulus, since the stimulus interacts linearly with model parameters k. Concavity of the log-posterior holds so long as the prior $p(\boldsymbol{x})$ is itself log-concave (e.g., Gaussian, or any distribution of the form $\alpha e^{-(x/\sigma)^\gamma}$, with $\gamma \geq 1$). This means that, for both of these two models, we can perform MAP decoding of the stimulus using simple gradient ascent of the posterior.

If we wish to perform decoding with a specified loss function (e.g., mean-squared error), optimal decoding can be achieved with Bayesian estimation, which is given by the estimator with minimum expected loss. In the case of mean-squared error, this estimator is given by

$$\hat{\boldsymbol{x}}_{Bayes} = E[\boldsymbol{x} \mid \boldsymbol{y}, \theta], \qquad (19)$$

which is the conditional expectation of $\boldsymbol{x}$, or the mean of the posterior distribution over stimuli. Computing this estimate, however, requires sampling from the posterior distribution, which is difficult to perform without advanced statistical sampling techniques and is a topic of ongoing research.

Considered more generally, decoding provides an important test of model validity, and it allows us to ask different questions about the nature of the neural code. Even though it may not be a task carried out explicitly in the brain, decoding allows us to measure how well a particular model preserves the stimulus-related information in the neural response. This is a subtle point, but one worth considering: we can imagine a model that performs worse under cross-validation or time-rescaling analyses but performs better at decoding, and therefore gives a better account of the stimulus-related information that is conveyed to the brain. For example, consider a model that fails to account for the refractory period (e.g., an LNP model) but gives a slightly better description of the stimulus-related probability of spiking. This model assigns non-zero probability to spike trains that violate the refractory period, thereby "wasting" probability mass on spike trains whose probability is actually zero, and performs poorly under cross-validation. The model also performs poorly under time-rescaling owing to the fact that it overpredicts spike rate during the refractory period. However, when decoding a *real* spike train, we do not encounter violations of the refractory period, and the "wasted" probability mass affects only the normalizing term $p(\boldsymbol{y} \mid \theta)$. Here, the model's improved accuracy for predicting the stimulus-related spiking activity leads to a posterior that is more reliably centered around the true stimulus. Thus, even though the model fails to reproduce certain statistical features of the response, it provides a valuable tool for assessing what information the spike train carries about the stimulus and gives a perhaps more valuable description of the neural code. Decoding may therefore serve as an important tool for validating likelihood-based models, and a variety of exact or approximate likelihood-based techniques for neural decoding have been explored (Warland et al., 1997; Brown et al., 1998; Barbieri et al., 2004; Pillow et al., 2005b).

Summary

We have shown how to compute likelihood and perform ML fitting of several types of probabilistic neural models. In simulations, we have shown that ML outperforms traditional moment-based estimators (STA and STC) when the nonlinear function of filter output does not have a particular exponential form. We have also discussed models whose log-likelihood functions are provably concave, making ML estimation possible even in high-dimensional parameter spaces and with non-Gaussian stimuli. These models can also be extended to incorporate dependence on spike-train history and external covariates of the neural response, such as spiking activity in nearby neurons. We have examined several statistical approaches to validating the performance of a neural model, which allow us to decide which models to use and to assess how well they describe the neural code.

In addition to the insight they provide into the neural code, the models we have described may be useful for simulating realistic input to downstream brain regions, and in practical applications such as neural prosthetics. The theoretical and statistical tools that we have described here, as well as the vast computational resources that make them possible, are still a quite recent development in the history of theoretical neuroscience. Understandably, their

NOTES

achievements are still quite modest: we are far away from a "complete" model that predicts responses to any stimulus (e.g., incorporating the effects of spatial and multiscale temporal adaptation, network interactions, and feedback). There remains much work to be done both in building more powerful and accurate models of neural responses, and in extending these models (perhaps in cascades) to the responses of neurons in brain areas more deeply removed from the periphery.

Acknowledgments

This chapter was previously published as Pillow J (2007) Likelihood-based approaches to modeling the neural code. In: Bayesian brain: probabilistic approaches to neural coding, Chap 7 (Doya K, Ishii S, Pouget A, Rao R, eds), pp 53–70. Cambridge, MA: MIT Press.

References

Aguera y Arcas B, Fairhall AL (2003) What causes a neuron to spike? Neural Comput 15:1789–1807.

Barbieri R, Frank L, Nguyen D, Quirk M, Solo V, Wilson M, Brown E (2004) Dynamic analyses of information encoding in neural ensembles. Neural Comput 16:277–307.

Bialek W, de Ruyter van Steveninck R (2005) Features and dimensions: motion estimation in fly vision. Quantitative Biology—Neurons and Cognition arXiv:qbio.NC/0505003.

Brown E, Frank L, Tang D, Quirk M, Wilson M (1998) A statistical paradigm for neural spike train decoding applied to position prediction from ensemble firing patterns of rat hippocampal place cells. J Neurosci 18:7411–7425.

Brown E, Barbieri R, Ventura V, Kass R, Frank L (2002) The time-rescaling theorem and its application to neural spike train data analysis. Neural Comput 14:325–346.

Bryant H, Segundo J (1976) Spike initiation by transmembrane current: a white-noise analysis. J Physiol 260:279–314.

Bussgang J (1952) Crosscorrelation functions of amplitude-distorted Gaussian signals. RLE Technical Reports 216. Boston: MIT Research Laboratory of Electronics.

Chichilnisky EJ (2001) A simple white noise analysis of neuronal light responses. Network 12:199–213.

Chornoboy E, Schramm L, Karr A (1988) Maximum likelihood identification of neural point process systems. Biol Cybern 59:265–275.

de Ruyter van Steveninck R, Bialek W (1988) Real-time performance of a movement-sensitive neuron in the blowfly visual system: coding and information transmission in short spike sequences. Proc R Soc Lond B 234:379–414.

Doya K, Ishii S, Pouget A, Rao R (eds). Bayesian brain: probabilistic approaches to neural coding. Cambridge, MA: MIT Press.

Gerstner W, Kistler W (2002) Spiking neuron models: single neurons, populations, plasticity. Cambridge, UK: University Press.

Jolivet R, Lewis TJ, Gerstner W (2003) The spike response model: a framework to predict neuronal spike trains. In: Springer lecture notes in computer science, Vol 2714 (Kaynak O, ed), pp 846–853. Berlin: Springer.

Keat J, Reinagel P, Reid R, Meister M (2001) Predicting every spike: a model for the responses of visual neurons. Neuron 30:803–817.

Okatan M, Wilson M, Brown E (2005) Analyzing functional connectivity using a network likelihood model of ensemble neural spiking activity. Neural Comput 17:1927–1961.

Paninski L (2003) Convergence properties of some spike-triggered analysis techniques. Network 14:437–464.

Paninski L (2004) Maximum likelihood estimation of cascade point-process neural encoding models. Network 15:243–262.

Paninski L, Pillow J, Simoncelli E (2004) Maximum likelihood estimation of a stochastic integrate-and-fire neural model. Neural Comput 16:2533–2561.

Pillow JW, Simoncelli EP (2006) Dimensionality reduction in neural models: an information-theoretic generalization of spike-triggered average and covariance analysis. J Vis 6:414–428.

Pillow JW, Shlens J, Paninski L, Chichilnisky EJ, Simoncelli EP (2005a) Modeling the correlated spike responses of a cluster of primate retinal ganglion cells. Soc Neurosci Abstr 31:591.3.

Pillow JW, Paninski L, Uzzell VJ, Simoncelli EP, Chichilnisky EJ (2005b) Prediction and decoding of retinal ganglion cell responses with a probabilistic spiking model. J Neurosci 25:11003–11013.

Reich DS, Victor JD, Knight BW (1998) The power ratio and the interval map: spiking models and extracellular recordings. J Neurosci 18:10090–10104.

NOTES

Schwartz O, Chichilnisky EJ, Simoncelli EP (2002) Characterizing neural gain control using spike-triggered covariance. In: Advances in neural information processing systems (Dietterich TG, Becker S, Ghahramani Z, eds), Vol 14, pp 269–276. Cambridge, MA: MIT Press.

Schwartz O, Pillow JW, Rust NC, Simoncelli EP (2006) Spike-triggered neural characterization. J Vis 6:484–507.

Sharpee T, Rust N, Bialek W (2004) Analyzing neural responses to natural signals: maximally informative dimensions. Neural Comput 16:223–250.

Simoncelli E, Paninski L, Pillow J, Schwartz O (2004) Characterization of neural responses with stochastic stimuli. In: The cognitive neurosciences III (Gazzaniga M, ed), pp 327–338. Cambridge, MA: MIT Press.

Touryan J, Lau B, Dan Y (2002) Isolation of relevant visual features from random stimuli for cortical complex cells. J Neurosci 22:10811–10818.

Truccolo W, Eden UT, Fellows MR, Donoghue JP, Brown EN (2004) A point process framework for relating neural spiking activity to spiking history, neural ensemble and extrinsic covariate effects. J Neurophysiol 93:1074–1089.

Warland D, Reinagel P, Meister M (1997) Decoding visual information from a population retinal ganglion cells. J Neurophysiol 78:2336–2350.

Spatiotemporal Correlations and Visual Signaling in a Complete Neuronal Population

Jonathan W. Pillow, PhD,[1] Jonathon Shlens, PhD,[2] Liam Paninski, PhD,[3] Alexander Sher, PhD,[4] Alan M. Litke, PhD,[4] E. J. Chichilnisky, PhD,[2] and Eero P. Simoncelli, PhD[5]

[1]Princeton Neuroscience Institute
Princeton, New Jersey

[2]Department of Neurosurgery and Department of Ophthalmology
Stanford University
Stanford, California

[3]Department of Statistics and Center for Theoretical Neuroscience
Kavli Institute for Brain Science
Columbia University
New York, New York

[4]Santa Cruz Institute for Particle Physics
University of California, Santa Cruz
Santa Cruz, California

[5]Howard Hughes Medical Institute
Center for Neural Science
Courant Institute of Mathematical Sciences
New York University
New York, New York

NOTES

Introduction

Statistical dependencies in the responses of sensory neurons govern both the amount of stimulus information conveyed and the means by which downstream neurons can extract it. Although a variety of measurements indicate the existence of such dependencies (Mastronarde, 1989; Meister et al., 1995; Shadlen and Newsome, 1998), their origin and importance for neural coding are poorly understood. Here we analyze the functional significance of correlated firing in a complete population of macaque parasol retinal ganglion cells (RGCs) using a model of multineuron spike responses (Paninski, 2004; Truccolo et al., 2004). The model, with parameters fit directly to physiological data, simultaneously captures both the stimulus dependence and detailed spatiotemporal correlations in population responses, and provides two insights into the structure of the neural code. First, neural encoding at the population level is less noisy than one would expect from the variability of individual neurons: spike times are more precise, and can be predicted more accurately when the spiking of neighboring neurons is taken into account. Second, correlations provide additional sensory information: optimal, model-based decoding that exploits the response correlation structure extracts 20% more information about the visual scene than decoding under the assumption of independence, and preserves 40% more visual information than optimal linear decoding (Warland et al., 1997). This model-based approach reveals the role of correlated activity in the retinal coding of visual stimuli and provides a general framework for understanding the importance of correlated activity in populations of neurons.

Neuronal Spiking Activity and the Sensory Environment

How does the spiking activity of a neural population represent the sensory environment? The answer depends critically on the structure of neuronal correlations, or the tendency of groups of neurons to fire temporally coordinated spike patterns. The statistics of such patterns have been studied in a variety of brain areas, and their significance in the processing and representation of sensory information has been debated extensively (Meister et al., 1995; Dan et al., 1998; Shadlen and Newsome, 1998; Nirenberg et al., 2001; Panzeri et al., 2001; Nirenberg and Latham, 2003; Schneidman et al., 2003; Averbeck and Lee, 2004; Latham and Nirenberg, 2005).

Previous studies have examined visual coding by pairs of neurons (Nirenberg and Latham, 2003) and the statistics of simultaneous firing patterns in larger neural populations (Schneidman et al., 2006; Shlens et al., 2006). However, no previous approach has addressed how correlated spiking activity in complete neural populations depends on the pattern of visual stimulation, or has answered the question of how such dependencies affect the encoding of visual stimuli.

Here we introduce a model-based methodology for studying this problem. We describe the encoding of stimuli in the spike trains of a neural population using a generalized linear model (GLM) (Fig. 1*a*), a generalization of the well-known linear–nonlinear–Poisson (LNP) cascade model (Plesser and Gerstner, 2000; Paninski, 2004; Simoncelli et al., 2004; Truccolo et al., 2004). In this model, each cell's input is described by a set of linear filters: a stimulus filter, or spatiotemporal receptive field; a postspike filter, which captures dependencies on spike-train history (e.g., refractoriness, burstiness, and adaptation); and a set of coupling filters, which captures dependencies on the recent spiking of other cells. For each neuron, the summed filter responses are exponentiated to obtain an instantaneous spike rate. This is equivalent to exponentiating the filter outputs and then multiplying; the exponentiated postspike and coupling filters (as plotted in Fig. 1) may therefore be interpreted as spike-induced gain adjustments of the neuron's firing rate.

Although this model is strictly phenomenological, its components can be loosely compared to biophysical mechanisms: the stimulus filter approximates the spatiotemporal integration of light in the outer retina and passive dendritic filtering; the postspike filter mimics voltage-activated currents following a spike; coupling filters resemble synaptic or electrical interactions between cells (and can mimic the effects of shared input noise); and the exponential nonlinearity implements a "soft threshold," converting membrane potential to instantaneous spike probability. Note that the postspike and coupling filters, which allow stochastic spiking in one cell to affect subsequent population activity, give rise to shared, non-Poisson variability in the model response.

A Model-Based Analysis of RGC Encoding

We fit the model to data recorded *in vitro* from a population of 27 ON and OFF parasol RGCs in a small patch of isolated macaque monkey retina, stimulated with 120 Hz spatiotemporal binary white noise. The receptive fields of each of the two cell types formed a complete mosaic covering a small

NOTES

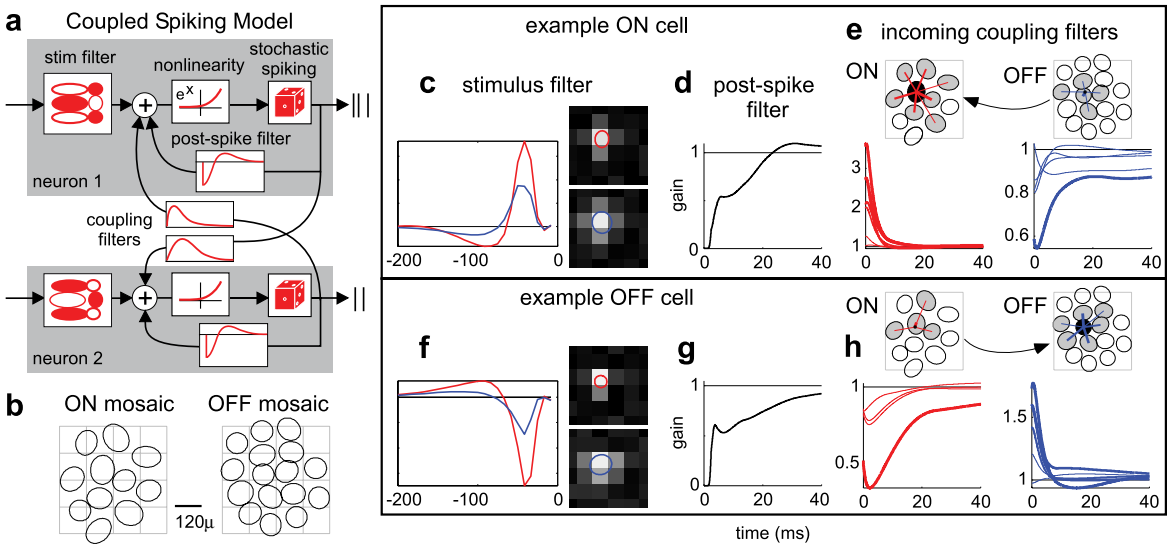

Figure 1. Multineuron encoding model and fitted parameters. ***a***, Model schematic for two coupled neurons: each neuron has a stimulus filter, a postspike filter, and coupling filters that capture dependencies on spiking in other neurons. Summed filter output passes through an exponential nonlinearity to produce the instantaneous spike rate. ***b***, Mosaics of 11 ON and 16 OFF RGC receptive fields, tiling a small region of visual space. Ellipses represent 1 SD of a Gaussian fit to each receptive field center; the square grid indicates stimulus pixels. Scale bar, 120 µm. ***c–e***, Parameters for an example ON cell. ***c***, Temporal and spatial components of center (red) and surround (blue) filter components, the difference of which is the full stimulus filter. ***d***, Exponentiated postspike filter, which may be interpreted as multiplying the spike rate after a spike at time zero. It produces a brief refractory period and gradual recovery (with a slight overshoot). ***e***, Connectivity and coupling filters from other cells in the population. The black-filled ellipse is this cell's RF center, and blue and red lines show connections from neighboring OFF and ON cells, respectively (line thickness indicates coupling strength). Below, exponentiated coupling filters show the multiplicative effect on this cell's spike rate after a spike in a neighboring cell. ***f–h***, Analogous plots for an example OFF cell.

region of visual space (Fig. 1*b*), indicating that every parasol cell in this region was recorded (Frechette et al., 2005; Shlens et al., 2006). Such complete recordings, which have not been achieved elsewhere in the mammalian nervous system, are essential for understanding visual coding in neural populations.

The model contains many parameters that specify the shapes of all filters, but fitting by maximizing likelihood remains highly tractable (Paninski, 2004). A penalty on coupling filters was used to obtain a minimally sufficient set of coupling filters, which yields an estimate of the network's functional connectivity (Okatan et al., 2005; Rigat et al., 2006).

Figure 1 shows the estimated filters describing input to example ON and OFF cells. The stimulus filters exhibit center-surround receptive field organization consistent with previous characterizations of parasol cells. Postspike filters show the time course of recovery from refractoriness after a spike, and coupling filters show the effects of spikes from nearby cells: for the ON cell (top), spikes in neighboring ON cells elicit a large, transient excitation (increasing the instantaneous spike rate by a factor of three), whereas spikes in nearby OFF cells elicit suppression. These effects are reversed in the OFF cell, which is excited or suppressed by spikes in neighboring OFF/ON cells. Both populations exhibit approximate nearest-neighbor connectivity, with coupling strength falling as a function of distance between receptive field centers (Shlens et al., 2006). We found that fitted stimulus filters have smaller surrounds than the spike-triggered average, indicating that a portion of the classical surround can be explained by interactions between cells (DeVries, 1999).

To assess accuracy in capturing the statistical dependencies in population responses, we compared the pairwise cross-correlation function (CCF) of RGCs and simulated model spike trains (Fig. 2). For nearby ON–ON and OFF–OFF pairs, the CCF exhibits a sharp peak at zero, indicating the prevalence of synchronous spikes; however, for ON–OFF pairs, a trough at zero indicates an absence of synchrony. For all 351 possible pairings, the model accurately reproduces the CCF (Figs. 2*a–c*, *e*, *f*).

To examine whether interneuronal coupling was necessary to capture the response correlation

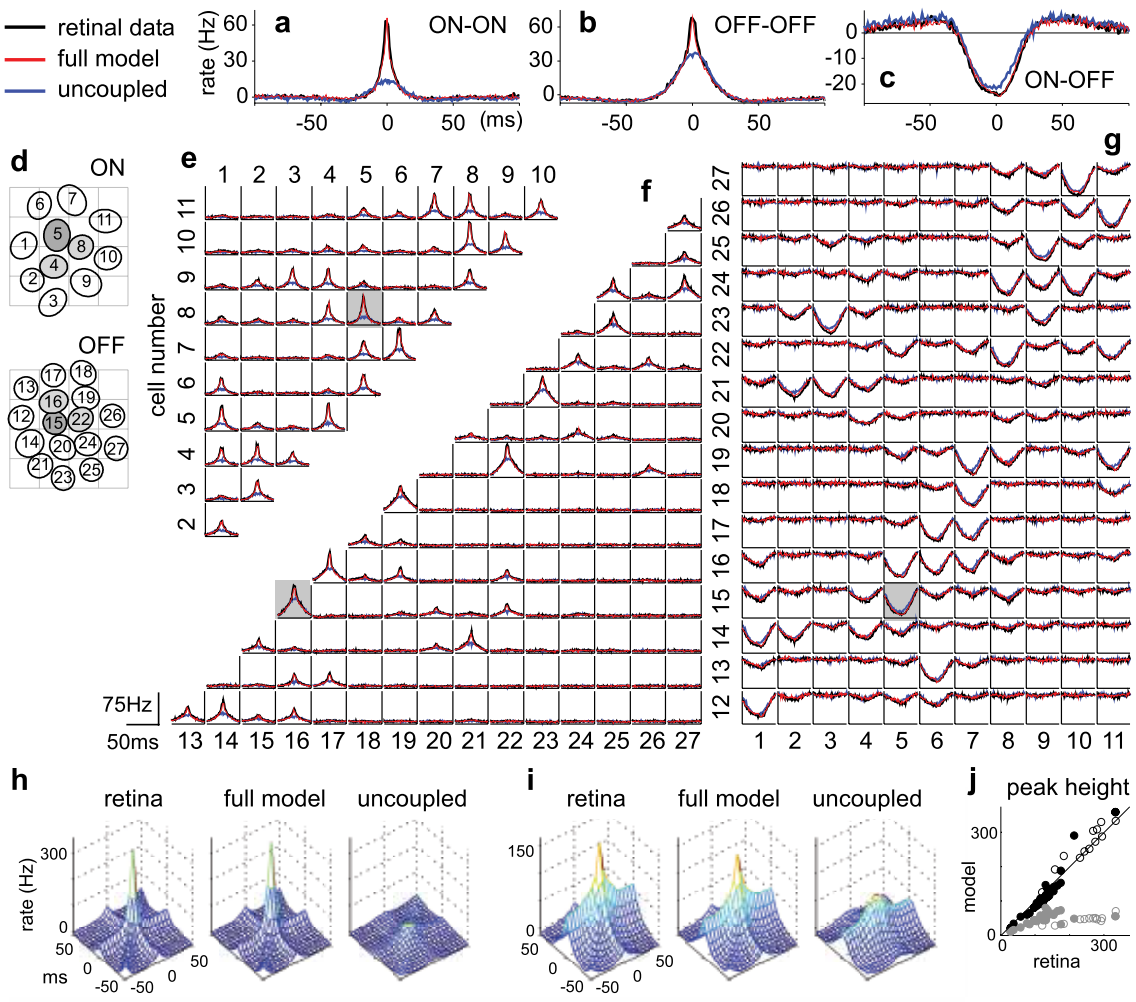

Figure 2. Analysis of response correlations. ***a–c***, Example CCFs of retinal responses, and simulated responses of the full and uncoupled models, for two ON cells (***a***), two OFF cells (***b***) and an ON–OFF pair (***c***). The baseline is subtracted so that units are in spikes per s above (or below) the cell's mean rate. ***d***, Receptive field mosaic overlaid with arbitrary labels. Dark gray indicates cells shown in Fig. 1; light gray indicates cells used for triple correlations (***h, i***). ***e***, CCFs between all ON pairs, where the *i,j*th plot shows the CCF between cell *i* and cell *j*. The gray box indicates the CCF plotted in ***a, f, g***, CCFs between all OFF–OFF pairs (***f***), and all ON–OFF pairs (***g***; abscissa height = 30 Hz). ***h***, Third-order (triplet) CCF between three adjacent ON cells, showing the instantaneous spike rate of cell 5 as a function of the relative spike time in cells 4 and 8 (left, RGCs; middle, full model; right, uncoupled model). ***i***, Analogous triplet CCF for OFF cells 15, 16, and 22. ***j***, Comparison of the triplet CCF peak in RGC and model responses (full model, black; uncoupled, gray) for randomly selected triplets of adjacent ON (open) and OFF (filled) cells.

structure, we refitted the model without coupling filters (that is, so that each cell's response depends only on the stimulus and its own spike-train history). This "uncoupled model" assumes that cells encode the stimulus independently, although correlations may still arise from the overlap of stimulus filters. However, the uncoupled model fails to reproduce the sharp CCF peaks observed in the data. These peaks are also absent from CCFs computed on trial-shuffled data, indicating that fast-timescale correlations are not stimulus-induced and therefore cannot be captured by any independent encoding model.

Higher-order statistical dependencies were considered by inspecting correlations in three-neuron groups: triplet CCFs show the spike rate of one cell as a function of the relative time to spikes in two other cells (Figs. 2*e–g*) (Shlens et al., 2006). For adjacent neurons of the same type, triplet CCFs have substantial peaks at zero ("triplet synchrony"), which are well matched by the full model.

Although the full and uncoupled models differ substantially in their statistical dependencies, the two models predict average light responses in

NOTES

individual cells with nearly identical accuracy, capturing 80–95% of the variance in the peristimulus time histogram (PSTH) in 26 out of 27 cells (Figs. 3*a*–*c*). Both models therefore accurately describe average single-cell responses to new stimuli. However, the full model achieves higher accuracy, predicting multineuronal spike responses on a single trial (8% ± 3% more bits per spike; Fig. 3*d*). This discrepancy can be explained by the fact that noise is shared across neurons. Shared variability means that population activity carries information about a single cell's response (owing to coupling between cells) beyond that provided by the stimulus alone. Individual neurons therefore appear less noisy when conditioned on spiking activity in the rest of the population than they appear in raster plots.

We measured the effect of correlations on single-trial, single-cell spike-train prediction by using the model to draw samples of a single cell's response, given both the stimulus and the spiking activity in the rest of the population on a single trial (Figs. 3*e*, *f*). Averaging the resulting raster plot gives a prediction of the cell's single-trial spike rate, or "population-conditioned" PSTH for a single trial. We compared these predictions with the cell's true spike times (binned at 2 ms) across all trials and found that on nearly every trial, the model-based prediction is

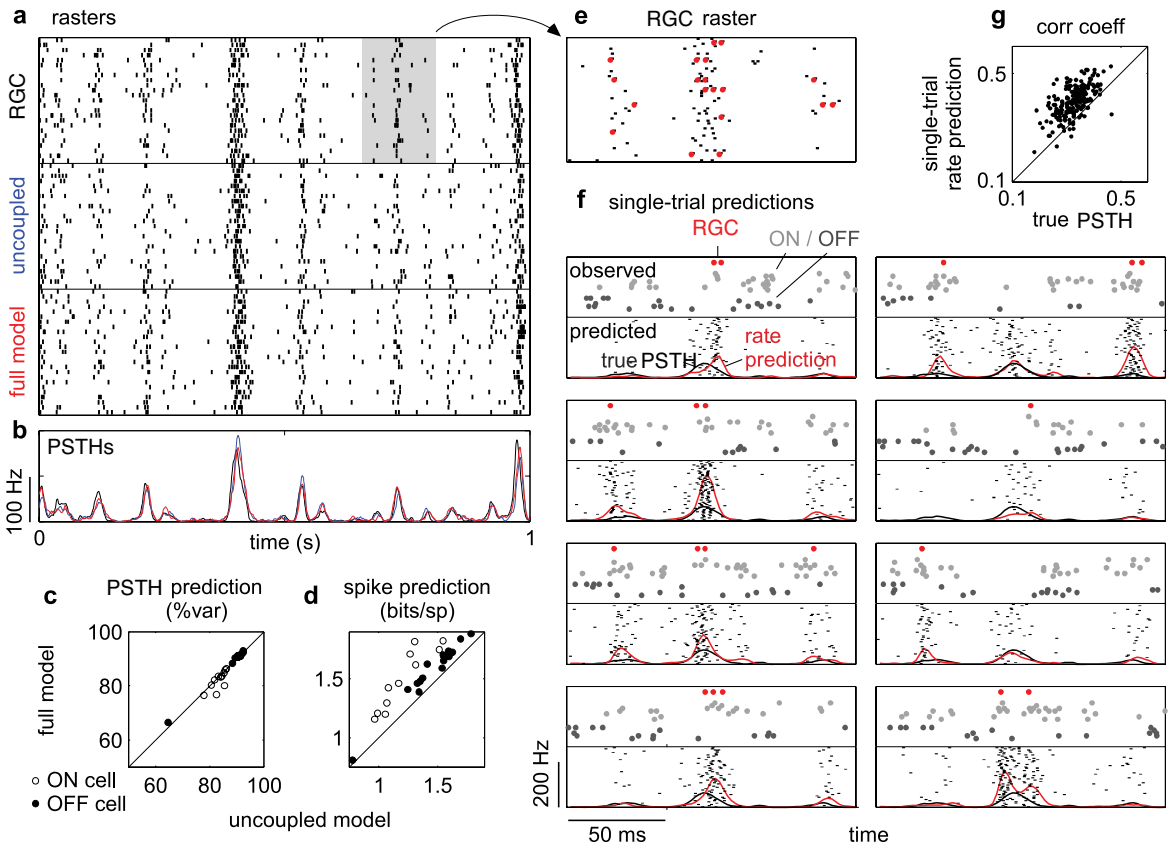

Figure 3. Spike-train prediction comparison. ***a***, Raster of responses of an ON RGC to 25 repeats of a novel 1 s stimulus (top), and responses of uncoupled (middle) and full (bottom) models to the same stimulus. ***b***, PSTH of the RGC (black), uncoupled (blue) and coupled (red) model; both models account for ~84% of the variance of the true PSTH. ***c***, PSTH prediction by full and uncoupled models, showing that coupling confers no advantage in predicting average responses. ***d***, Log-likelihood of novel RGC spike responses under full and uncoupled models; the full model provides 8% more information about novel spike trains. ***e***, Magnified 150 ms portion of RGC raster and PSTH (gray box in ***a***). Red dots highlight RGC spike times on selected individual trials (replotted in ***f***). ***f***, Single-trial spike-train prediction using the coupled model. The top half of each plot shows the population activity on a single trial: true spike times of the cell (red dots), coupled ON cells (light gray dots), and coupled OFF cells (dark gray dots; each line in the raster shows the spike times of a different cell). The bottom half of each plot shows a raster of 50 predicted responses of the cell in question, using both the stimulus and coupled responses (shown above) to predict spike trains. The red trace shows the single-trial rate prediction (population-conditioned PSTH), compared with true PSTH of the cell (black trace, identical in all plots). ***g***, Correlation coefficient of true spike trains with the PSTH (*x*-axis) and with population-conditioned predictions (*y*-axis); the full model predicts single-trial responses with higher accuracy than the true PSTH.

NOTES

more highly correlated with the observed spikes than the neuron's full PSTH (Fig. 3*g*). Note that the full PSTH achieves the highest correlation possible for any trial-independent prediction. Thus, by exploiting the correlation structure, the coupled model predicts single-neuron spike times more accurately than any independent encoding model.

Bayesian Decoding of the Retinal Ganglion Cell Population Response

Although the full model accurately captures dependencies in the activity of RGCs, it is not obvious a priori whether these dependencies affect the amount of sensory information conveyed by RGC responses. In principle, the correlation structure could be necessary to predict the responses, but not to extract the stimulus information that the responses carry (Latham and Nirenberg, 2005). To examine this issue directly, we used the full and uncoupled models to perform Bayesian decoding of the population response (Fig. 4*a*), which optimally reconstructs stimuli, given an accurate description of the encoding process. For comparison, we also performed Bayesian decoding under a Poisson (i.e., LNP) model and optimal linear decoding (Warland et al., 1997).

Each decoding method was used to estimate short (150 ms) segments of the stimulus given all relevant spike times from the full population (Fig. 4*b*). Bayesian decoding under the coupled model recovers 20% more information than Bayesian decoding under the uncoupled model, indicating that knowledge of the correlation structure is critical for extracting all sensory information contained in the population response. This improvement was invariant to enhancements of the model's stimulus filters and nonlinearities, indicating that the difference in performance arises specifically from the coupled model's ability to incorporate the correlation structure. Our results also show that spike history is relevant for decoding (a Poisson model preserves 6% less information than the uncoupled model) (Pillow et al., 2005) and that restricting to a linear decoder further reduces the information that can be recovered from RGC responses.

Decoding analysis can also be used to examine the coding fidelity of specific stimulus features. As a simple illustration, we examined the temporal frequency spectrum of reconstructed stimuli and found that the response correlation structure is most important for decoding those stimulus frequencies (6–20 Hz) that are encoded with highest fidelity (Fig. 4*c*).

Results and the Limitations of the Generalized Linear Model

These results demonstrate that the responses of a population of RGCs are well described by a GLM, and that correlations in the response can be exploited

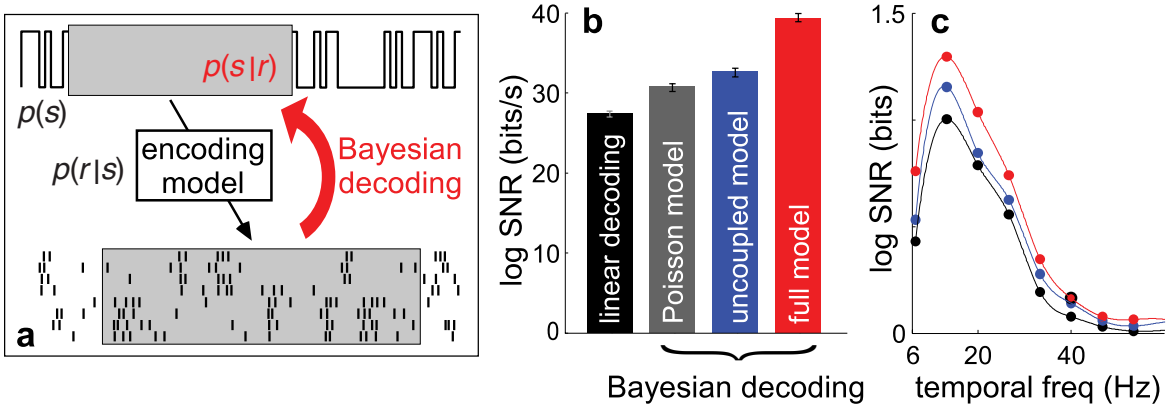

Figure 4. Decoding performance comparison. ***a***, a Bayesian decoding schematic: to estimate an unknown stimulus segment from a set of observed spike times (highlighted in boxes), the stimulus prior distribution $p(s)$ is multiplied by the model-defined likelihood $p(r \mid s)$ to obtain the posterior $p(s \mid r)$. The posterior mean is the Bayes' least-squares stimulus estimate. ***b***, Log of the SNR for linear decoding, as well as for Bayesian decoding under the Poisson, uncoupled, and full models (Warland et al., 1997). The full model preserves 20% more information than the uncoupled model, which indicates that there is additional sensory information available from the population response when correlations are taken into account. Error bars show 95% confidence intervals based on 2000 bootstrap resamplings of 3000 decoded stimulus segments. ***c***, Log SNR decomposed as a function of temporal frequency for various decoding methods (Poisson omitted for clarity).

NOTES

to recover 20% more visual information than if responses were regarded as independent, given the stimulus. In contrast, previous studies have reported this information gain to be <10% for pairs of neurons (Nirenberg et al., 2001; Averbeck and Lee, 2004). However, pairwise analyses provide little evidence about the importance of correlations across an entire population. Second-order correlations between pairs of neurons could give rise to either much larger (scaling with the number of neurons n) or much smaller (falling as $1/n$) gains for a full population. To compare more directly with previous findings, we performed Bayesian decoding using isolated pairs of neurons from the same population; we found a ≤10% gain in sensory information when correlations were included. This is consistent with previous findings and shows that the information gain for a complete population is larger than that observed for pairs. We also compared the model with a pairwise maximum-entropy model, which has recently been shown to capture the instantaneous spiking statistics of groups of RGCs (Schneidman et al., 2006; Shlens et al., 2006). The coupled model exhibits similar accuracy in capturing these statistics, but has the advantage that it accounts for the temporal correlation structure and stimulus dependence of responses, which are essential for assessing the effect of correlations on sensory coding.

Although it provides an accurate functional description of correlated spike responses, the GLM does not reveal the biophysical mechanisms underlying the statistical dependencies between neurons: coupling does not necessarily imply anatomical connections between cells but could (for example) reflect dependencies due to shared input noise (Mastronarde, 1989). The model also lacks several mechanisms known to exist in RGCs (e.g., contrast gain-control) (Shapley and Victor, 1978), which may be required for characterizing responses to a wider variety of stimuli. One additional caveat is that Bayesian decoding provides a tool for measuring the sensory information available in the population response, but it does not reveal whether the brain makes use of this information. Physiological interpretations of the model and mechanisms for neural readout of sensory information in higher brain areas are thus important directions for future research.

Nevertheless, the GLM offers a concise, computationally tractable description of the population encoding process and provides the first generative description of the space–time dependencies in stimulus-induced population activity. It allows us to quantify the relative contributions of stimulus, spike history, and network interactions to the encoding and decoding of visual stimuli and clarifies the relationship between single-cell and population variability. More generally, the model can be used to assess which features of the visual environment are encoded with highest and lowest fidelity and to determine how the structure of the neural code constrains perceptual capabilities. We expect this framework to extend to other brain areas and to have an important role in revealing the information processing capabilities of spiking neural populations (Harris et al., 2003; Paninski et al., 2004; Truccolo et al., 2004; Okatan et al., 2005).

Methods Summary

Data

Multielectrode extracellular recordings were obtained *in vitro* from a segment of isolated, peripheral macaque monkey (*Macaca mulatta*) retina, and analysis was restricted to two cell types (ON and OFF parasol) (Watanabe and Rodieck, 1989; Litke et al., 2004; Shlens et al., 2006). A standard spike-sorting procedure, followed by a specialized statistical method for detecting simultaneous spikes, was used to sort spikes (Segev et al., 2004). The retina was stimulated with a photopic, achromatic, optically reduced spatiotemporal binary white-noise stimulus refreshing at 120 Hz, with a root-mean-square contrast of 96%.

Fitting

Model parameters were fitted to 7 min of spike responses to a nonrepeating stimulus. Each cell's parameters consisted of a stimulus filter (parametrized as a rank 2 matrix), a spike-history filter, a set of incoming coupling filters, and a constant. Temporal filters were represented in a basis of cosine "bumps" (Pillow et al., 2005). Parameters for the uncoupled and Poisson (LNP) models were fitted independently. Parameters were fitted by penalized maximum likelihood (Paninski, 2004; Truccolo et al., 2004) using an L1 penalty on the vector length of coupling filters to eliminate unnecessary connections.

Encoding

Spike prediction was cross-validated using the log-likelihood of 5 min of novel spiking data (scaled to units of bits/s). Repeat rasters were obtained using 200 presentations of a novel 10 s stimulus. Population-conditional rasters were obtained from the coupled model by sampling the model-defined probability distribution over the neuron's response, given the stimulus and surrounding-population activity on a single trial (Pillow et al., 2008).

NOTES

Decoding

Population responses were decoded using the Bayes' least-squares estimator (posterior mean) to reconstruct 18-sample single-pixel stimulus segments (cross-validation data). Linear decoding was performed using the optimal linear estimator (Warland et al., 1997). Decoding performance was quantified using the log signal-to-noise ratio (SNR) of each technique, which gives an estimate of mutual information. Breakdown by temporal frequency was obtained by computing the Fourier power spectra of the stimuli and residuals and then computing log SNR.

Acknowledgments

We thank M. Bethge, C. Brody, D. Butts, P. Latham, M. Lengyel, S. Nirenberg, and R. Sussman for comments and discussions; G. Field, M. Greschner, J. Gauthier, and C. Hulse for experimental assistance; M.I. Grivich, D. Petrusca, W. Dabrowski, A. Grillo, P. Grybos, P. Hottowy, and S. Kachiguine for technical development; H. Fox, M. Taffe, E. Callaway, and K. Osborn for providing access to retinas; and S. Barry for machining. Funding was provided by a Royal Society USA/Canada Research Fellowship to J.W.P.; a National Science Foundation (NSF) Integrative Graduate Education and Research Training Grant DGE-03345 to J.S.; a National Eye Institute Grant EY018003 to E.J.C., L.P., and E.P.S.; a Gatsby Foundation Pilot Grant to L.P.; a Burroughs Wellcome Fund Career Award at the Scientific Interface to A.S.; an NSF Grant PHY-0417175 to A.M.L.; McKnight Foundation support to A.M.L. and E.J.C.; and a Howard Hughes Medical Institute grant to J.W.P., L.P., and E.P.S.

This chapter was excerpted with permission from Pillow JW et al. (2008) Spatio-temporal correlations and visual signalling in a complete neuronal population, Nature 454:995–999. Copyright 2008, Nature Publishing Group. A complete description of Methods and associated references are available in the online version of the paper at www.nature.com/nature.

References

Averbeck BB, Lee D (2004) Coding and transmission of information by neural ensembles. Trends Neurosci 27:225–230.

Dan Y, Alonso JM, Usrey WM, Reid RC (1998) Coding of visual information by precisely correlated spikes in the lateral geniculate nucleus. Nat Neurosci 1:501–507.

DeVries SH (1999) Correlated firing in rabbit retinal ganglion cells. J Neurophysiol 81:908–920.

Frechette ES, Sher A, Grivich MI, Petrusca D, Litke AM, Chichilnisky EJ (2005) Fidelity of the ensemble code for visual motion in primate retina. J Neurophysiol 94:119–135.

Harris K, Csicsvari J, Hirase H, Dragoi G, Buzsaki G (2003) Organization of cell assemblies in the hippocampus. Nature 424:552–556.

Latham P, Nirenberg S (2005) Synergy, redundancy, and independence in population codes, revisited. J Neurosci 25:5195–5206.

Litke AM, N Bezayiff, Chichilnisky EJ, Cunningham W, Dabrowski W, Grillo AA, Grivich M, Grybos P, Hottowy P, Kachiguine S, Kalmar RS, Mathieson K, Petrusca D, Rahman M, Sher A (2004) What does the eye tell the brain? Development of a system for the large scale recording of retinal output activity. IEEE Trans Nucl Sci 51:1434–1440.

Mastronarde DN (1989) Correlated firing of retinal ganglion cells. Trends Neurosci 12:75–80.

Meister M, Lagnado L, Baylor DA (1995) Concerted signaling by retinal ganglion cells. Science 270:1207–1210.

Nirenberg S, Carcieri S, Jacobs A, Latham P (2001) Retinal ganglion cells act largely as independent encoders. Nature 411:698–701.

Nirenberg S, Latham PE (2003) Decoding neuronal spike trains: How important are correlations? Proc Natl Acad Sci USA 100:7348–7353.

Okatan M, Wilson M, Brown E (2005) Analyzing functional connectivity using a network likelihood model of ensemble neural spiking activity. Neural Comput 17:1927–1961.

Paninski L (2004) Maximum likelihood estimation of cascade point-process neural encoding models. Network Comp Neural Syst 15:243–262.

Paninski L, Fellows M, Shoham S, Hatsopoulos N, Donoghue J (2004) Superlinear population encoding of dynamic hand trajectory in primary motor cortex. J Neurosci 24:8551–8561.

Panzeri S, Golledge H, Zheng F, Tovee MP, Young MJ (2001) Objective assessment of the functional role of spike train correlations using information measures. Vis Cogn 8:531–547.

Pillow JW, Latham P (2008) Neural characterization in partially observed populations of spiking neurons. In: Advances in neural information processing systems, Vol 20 (Platt JC, Koller D, Singer Y, Roweis S, eds), pp 1161–1168. Cambridge, MA: MIT Press.

NOTES

Pillow JW, Paninski L, Uzzell VJ, Simoncelli EP, Chichilnisky EJ (2005) Prediction and decoding of retinal ganglion cell responses with a probabilistic spiking model. J Neurosci 25:11003–11013.

Plesser H, Gerstner W (2000) Noise in integrate-and-fire neurons: from stochastic input to escape rates. Neural Comput 12:367–384.

Rigat F, de Gunst M, van Pelt J (2006) Bayesian modelling and analysis of spatio-temporal neuronal networks. Bayes Anal 1:733–764.

Schneidman E, Bialek W, Berry MJ (2003) Synergy, redundancy, and independence in population codes. J Neurosci 21:11539–11553.

Schneidman E, Berry M, Segev R, Bialek W (2006) Weak pairwise correlations imply strongly correlated network states in a neural population. Nature 440:1007–1012.

Segev R, Goodhouse J, Puchalla J, Berry MJ (2004) Recording spikes from a large fraction of the ganglion cells in a retinal patch. Nat Neurosci 7:1155–1162.

Shadlen M, Newsome W (1998) The variable discharge of cortical neurons: implications for connectivity, computation, and information coding. J Neurosci 18:3870–3896.

Shapley RM, Victor JD (1978) The effect of contrast on the transfer properties of cat retinal ganglion cells. J Physiol 285:275–298.

Shlens J, Field GD, Gauthier JL, Grivich MI, Petrusca D, Sher A, Litke AM, Chichilnisky EJ. (2006) The structure of multi-neuron firing patterns in primate retina. J Neurosci 26:8254–8266.

Simoncelli EP, Paninski L, Pillow J, Schwartz O (2004) Characterizations of neural responses with stochastic stimuli. In: The cognitive neurosciences III (Gazzaniga M, ed), pp 327–338. Cambridge, MA: MIT Press.

Truccolo W, Eden UT, Fellows MR, Donoghue JP, Brown EN (2004) A point process framework for relating neural spiking activity to spiking history, neural ensemble and extrinsic covariate effects. J Neurophysiol 93:1074–1089.

Warland D, Reinagel P, Meister M (1997) Decoding visual information from a population of retinal ganglion cells. J Neurophysiol 78:2336–2350.

Watanabe M, Rodieck RW (1989) Parasol and midget ganglion cells of the primate retina. J Comp Neurol 289:434–454.