ALGORITHMS FOR UNDERSTANDING MOTOR CORTICAL PROCESSING AND NEURAL PROSTHETIC SYSTEMS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

John Patrick Cunningham

August 2009

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Krishna V. Shenoy) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Maneesh Sahani)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Stephen P. Boyd)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(William T. Newsome)

Approved for the University Committee on Graduate Studies.

Abstract

Our seemingly effortless ability to make coordinated movements belies the sophisticated computational machinery at work in our nervous system. Much has been learned about motor cortical processing with classic systems neuroscience approaches. In recent years, the field has been dramatically expanding the complexity of its data, recording technologies, and experiments. This shift seeks to deliver a much deeper understanding of cortical processing and a much improved ability to control neural prosthetic devices (also called brain-machine interfaces). Realizing this payoff, however, requires analytical and computational methods that can exploit this changing paradigm. This dissertation describes algorithmic developments for understanding cortical processing and for prosthetic systems. The first part focuses on our signal processing efforts to extract useful signals underlying noisy neural activity in the motor system of the primate brain, both for single neurons and for populations of simultaneously recorded neurons. Specifically, I discuss analyzing single neurons using machine learning techniques (Gaussian Processes) in a point process framework, and I describe an approach to mitigate the significant optimization challenges of this method. I then discuss extending this idea across many simultaneously recorded neurons (with a factor analysis-like algorithm) to extract population-level signatures of neural activity. This part also points to broader implications of this work for engineering and statistics. The second part focuses on algorithmic challenges in neural prosthetic systems, first describing performance improvements available via algorithmic optimization of a prosthetic interface. I then discuss other algorithmic areas of prosthetic design, reviewing a number of areas for improvement and pointing to future work to address these current problems. As a whole, this dissertation offers analytical methods that push forward advanced research into the brain's motor system and our ability to meaningfully interface with it.

Acknowledgement

With great thanks to all the family, friends, and loved ones of Richard Smellinger, and to Richard himself.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In many biomedical applications, including neuroscience, data analysis needs are increasing dramatically. My research and this dissertation focus on motor cortical processing and the medically relevant field of neural prosthetic systems. This dissertation should be viewed in the context of the broader goal of developing algorithms to address critical analytical needs in this field, significantly advancing our ability to understand the brain, and enabling clinically viable neural prosthetic devices.

Classic systems neuroscience involves tightly controlled experiments, during which a subject repeats trials of a task specifically designed to elucidate a feature of cortex. Electrophysiologists typically record a single neuron's spiking activity on each trial, and they correlate neural activity with features of physical behavior. This paradigm has produced virtually all the knowledge we now have about the brain's systems-level function. However, new technologies are altering this classic approach. First, multi-electrode arrays are being implanted in different areas of cortex, allowing simultaneous recordings from hundreds of neurons across the brain. Second, wireless technology is enabling always-on recordings from hundreds of neurons, 24 hours a day, during free, "real world" physical behavior that is not under any experimental control. Third, in addition to intracortical electrode techniques, technologies such as optogenetics are becoming available, offering a multitude of stimulation and recording modalities.

These technologies alone mark a fundamental shift in systems neuroscience that parallels technological shifts seen previously in other domains. When communications technology shifted from dedicated single channels to MIMO systems and network multi-user systems, communications algorithms were fundamentally redesigned,

resulting in a profound technological revolution. Neuroscience may benefit from a similar path of algorithmic development. The mammalian brain is among the most complex systems in existence, and understanding such complexity must require focused research in both technology and methodology. While this goal is hugely rewarding from a basic science perspective, it also should significantly advance the important medical application of neural prosthetic devices for the millions of people with motor deficits due to ALS, paralysis, and other conditions. Currently, neural prostheses work reasonably well in highly controlled scenarios (*e.g.*, making discrete choices or making very slow movements), but the field will require major algorithmic progress before a prosthesis can rival the capabilities of a functional human arm.

With this high level motivation, the following sections introduce the two major parts of this dissertation, which are followed by an outline of the specific work included in each chapter.

1.1 Part I

It has long been known that certain brain areas are intimately involved in the process of planning and executing movement (Evarts, 1968; Tanji and Evarts, 1976; Rosenbaum, 1980; Weinrich et al., 1984; Riehle and Requin, 1989) (to name just a few). In this dissertation, as is the case in much of the motor cortical literature, we discuss particularly the arm reaching system, as it provides an excellent experimental paradigm with which to study how the brain (specifically primary motor (M1) and dorsal premotor (PMd) cortices) plans and executes movement. For example, early work that showed a linear correspondence between parameters of a reach (direction of movement, speed, etc.) and neural activity (Georgopoulos et al., 1986). A vast number of follow-on studies have found many more parameters (force, joint torque, etc.) that are correlated with neural activity, and some of these studies have called into question the generality and appropriateness of a simple "representational" relationship between parameters of a reach and neural activity (Todorov and Jordan, 2002; Todorov, 2000)[1]. This and much more prior work illustrates that a great deal of interesting motor control occurs in these brain areas and, second, that the field has

[1]The reviews and discussion in Todorov and Jordan (2002); Todorov (2000); Moran and Schwartz (2000) provide a thorough background of this interesting and sometimes contentious debate.

by no means reached a central conclusion about *how* this motor control is conducted in these brain areas. Questions of motor cortical processing remain deeply important to a broader understanding of how the brain computes, and there exists a large and active field interested in this scientific pursuit.

Typically, motor cortical processing is studied with the classic systems neuroscience paradigm described above, by recording and averaging many experimental trials of spiking neural data and correlating that neural activity to behavioral features. This paradigm allows researchers to ask how a neuron responds on average to a particular behavioral feature, but it does not necessarily allow investigation into the computational mechanisms in the motor system that actually control movement. For example (and in anticipation of Chapters 2 and 5), if the neural responses are more a reflection of internal processing rather than external stimulus drive, the timecourse of the neural responses may differ on nominally identical trials. This is particularly true of behavioral tasks involving perception, decision making, attention, or motor planning. In such settings, it is critical that the neural data not be averaged across trials, but instead be analyzed on a trial-by-trial basis (Arieli et al., 1996; Nawrot et al., 1999; Horwitz and Newsome, 2001; Ventura et al., 2005; Briggman et al., 2006; Yu et al., 2006; Churchland et al., 2007; Jones et al., 2007; Czanner et al., 2008).

This high-level example, which is described in much greater detail in 5.1, points out that not all questions of motor cortical processing can be answered with averaged neural responses gathered with a classic systems neuroscience paradigm. Instead, one might ideally want a direct view of the time-evolution of neural activity (across a population of many neurons in the motor areas) on a single trial, as such a view could allow deeper investigation into the computational mechanisms employed by the motor system. This dissertation discusses these problems and algorithmic opportunities in understanding motor cortical processing.

1.2 Part II

Debilitating diseases like Amyotrophic Lateral Sclerosis (ALS/Lou Gehrig's disease) can leave a human without voluntary motor control. However, in many cases, the brain itself maintains normal function. The same is true with spinal cord injuries that result in severe paralysis. Millions of people worldwide suffer motor deficits due

to these diseases and injuries that result in significantly diminished ability to interact with the physical world. Indeed, tetrapalegic humans list regaining "arm/hand function" as *the* top priority for improving their quality of life, as restoring this function would allow significant independence (Anderson, 2004). To address this huge medical need, neural prosthetic systems seek to access the information in the brain and use that information to control a prosthetic device such as a robotic arm or a computer cursor. Such systems, if successful, would have large quality of life impact for many people living with these debilitating medical conditions.

In the last decade, advances in neural recording technologies have accelerated research in neural prosthetic systems (also called brain-computer interfaces or brain-machine interfaces). Technologies used in neural prosthetic systems include minimally invasive electroencephalography (EEG), electrocorticography (ECoG, below the skull), and invasive penetrating electrode or microwire arrays (see Lebedev and Nicolelis (2006) for a review). Each technology can record tens to hundreds of channels, but the tradeoff for surgical invasiveness is spatial and temporal resolution of the recorded signals. EEG has shown promise in allowing users control of a cursor on a 2D screen (Wolpaw and McFarland, 2004; Blankertz et al., 2004). To design a prosthetic arm that can be controlled continuously with high precision, most work has focused on penetrating electrodes implanted directly into motor cortical areas (Schwartz, 2004). Another important context for neural prosthetic systems is in a communications prosthesis setting (Shenoy et al., 2003). Here, the goal is to select from a number of discrete targets (such as keys on a keyboard), rather than to decode moment-by-moment parameters of a reaching arm. In both motor and communications prostheses, researchers use nonhuman primates (rhesus monkeys, as we do in this work) or, rarely, human participants (Hochberg et al., 2006; Kim et al., 2008). There are many medical, scientific, and engineering challenges in developing such a system (Lebedev and Nicolelis, 2006; Schwartz, 2004), but all neural prosthetic systems must have a decode algorithm. Decode algorithms map neural activity into physical commands such as kinematic parameters to control a robotic arm or select from among several target choices.

Much work has gone into this domain, and many experimental paradigms and decoding approaches have been developed and used (Georgopoulos et al., 1986; Wessberg et al., 2000; Serruya et al., 2002; Taylor et al., 2002; Carmena et al., 2003; Shenoy

et al., 2003; Wu et al., 2004; Hatsopoulos et al., 2004; Lebedev et al., 2005; Carmena et al., 2005; Wu et al., 2006; Hochberg et al., 2006; Santhanam et al., 2006; Chestek et al., 2007; Velliste et al., 2008; Kim et al., 2008; Brown et al., 1998; Gao et al., 2002; Eden et al., 2004; Kemere et al., 2004; Brockwell et al., 2004; Paninski et al., 2004b; Shoham et al., 2005; Kim et al., 2006; Shakhnarovich et al., 2006; Srinivasan et al., 2006; Srinivasan and Brown, 2007; Yu et al., 2007; Srinivasan et al., 2007; Artemiadis et al., 2007; Mulliken et al., 2008; Wu and Hatsopoulos, 2008; Ventura, 2008). Despite this abundance of work, our ability to decode arm movements accurately remains limited. To decode an arbitrary, continuous reach, the current state-of-the-art algorithm is the Kalman filter (introduced nearly fifty years ago in Kalman (1960), used in this context in Wu et al. (2006); Kim et al. (2008)), which is the only algorithm that has been vetted in online human experiments as having better performance than some competing possibilities (Kim et al., 2008). Current achievable performance is, loosely, that the decoded reach moves in roughly the correct direction but fails to get near or stop at the intended target (see Wu et al. (2006) or the review Cunningham et al. (2009)). While meaningful information throughput in communications prostheses has been reported (Santhanam et al., 2006), the field still has not produced a robust system with adequately high performance to be widely deployed in human patients. This dissertation discusses problems and algorithmic opportunities in both communications and motor prostheses.

1.3 Outline

With that scientific background to the dissertation, we here lay out the specific motivation for each of the subsequent chapters. Each of these chapters contributes to the overall goal of developing and testing algorithms that can help improve the field's understanding of motor cortical processing and help push the field towards a clinically viable neural prosthetic system.

- Part I describes signal processing research that has allowed deeper insight into motor cortical processing. These applied algorithms also raised research-grade computational questions and pointed to other non-neuroscientific algorithmic developments, which are also detailed in this part.

- Chapter 2 approaches the challenges inherent in analyzing neural spike trains due to their noisy, spiking nature. Many studies of neuroscientific and neural prosthetic importance rely on a smoothed, denoised estimate of the spike train's underlying firing rate. Current techniques to find time-varying firing rates require *ad hoc* choices of parameters, offer no confidence intervals on their estimates, and can obscure potentially important single trial variability. We present a new method, based on Gaussian Process regression from the field of machine learning, for inferring probabilistically optimal estimates of firing rate functions underlying single or multiple neural spike trains. We test the performance of the method on simulated data and experimentally gathered neural spike trains, and we demonstrate improvements over conventional estimators.
- Chapter 3 considers the serious computational requirements of the method introduced in Chapter 2, both in terms of memory and runtime requirements. Using large scale optimization techniques, this work shows orders of magnitude computational improvement and elimination of the memory burden of such a method. As this firing rate inference method is meant to be used by scientific researchers in real applied settings, such computational improvements may mean the difference between an academically interesting method and one that becomes well-used in practice.
- Chapter 4 describes a non-neuroscientific finding of broad statistical interest that was developed during the investigation of computational methods described in Chapter 3. The Gaussian is the most fundamental and widely used probability distribution in existence, but no closed-form formula exists for the cumulative distribution function. Extremely fast and accurate methods have been developed for univariate Gaussians, but a similar algorithm for multivariate Gaussians has not been found. We turn to the seemingly unrelated field of Bayesian inference - specifically Expectation Propagation - to develop a much simpler, more principled, and computationally faster method for calculating multivariate probabilities with high accuracy. This method is also the first to produce an analytical solution, which is largely important in many applied settings.

  - Chapter 5 uses and advances the developments described in previous chapters to investigate motor cortical processing across populations of simultaneously recorded neurons. We consider the problem of extracting smooth, low-dimensional *neural trajectories* that summarize the activity recorded simultaneously from many neurons on individual experimental trials. Beyond the benefit of visualizing the high-dimensional, noisy spiking activity in a compact form, such trajectories can offer insight into the dynamics of the neural circuitry underlying the recorded activity. We present a novel method for extracting neural trajectories, Gaussian-process factor analysis (GPFA), and we apply these methods to the activity of 61 neurons recorded simultaneously in premotor and motor cortices during reach planning and execution.

    These chapters conclude the algorithmic efforts we have developed to improve the field's ability to understand motor cortical processing and neural signals more generally.

- Part II describes algorithmic research focused on improving the performance of neural prosthetic systems.

  - Chapter 6 develops an algorithm to automate the target placement process and increase decode accuracy in communication prostheses by selecting target locations based on the neural population at hand. We present an optimal target placement algorithm that approximately maximizes decode accuracy with respect to target locations. We test and show statistically significant performance improvements in simulated neural spiking data. We also trained a monkey in this paradigm and tested the algorithm with experimental neural data to confirm some of the results found in simulation.
  - Chapter 7 turns to the question of firing rate estimation raised in Part I. The Gaussian Process method of Chapter 2 is one of many algorithmic approaches that has been developed in recent years to address this neural signal processing challenge, but no systematic comparison has been made between these methods. In an effort to understand the relevance of

these methods to the field of neural prostheses, we also apply these estimators to experimentally-gathered neural data from a prosthetic arm-reaching paradigm. Using these estimates of firing rate, we apply standard prosthetic decoding algorithms to compare the performance of the different firing rate estimators, and, perhaps surprisingly, we find minimal differences. This study serves as a review of available spike train smoothers and a first quantitative comparison of their performance for brain-machine interfaces.

– Lastly, Chapter 8 discusses current problems and future directions for algorithmic developments in neural prosthetic systems. This chapter highlights several outstanding problems that exist in most current approaches to decode algorithm design. These include two problems that may not result in further dramatic increases in performance, specifically spike sorting and spiking models. We also discuss three issues that have been less examined in the literature, and we argue that addressing these issues may result in dramatic future increases in performance. These include: non-stationarity of recorded waveforms, limitations of a linear mappings between neural activity and movement kinematics, and the low signal to noise ratio of the neural data. We demonstrate these problems with data from 39 experimental sessions with a non-human primate performing reaches and with recent literature. In all, this study suggests that research in cortically-controlled prosthetic systems may require reprioritization to achieve performance that is acceptable for a clinically viable human system.

These chapters conclude the algorithmic efforts we have investigated to improve the field's ability to decode physical behavior from neural activity.

Taken together, these chapters carefully investigate state-of-the-art algorithms in neural signal processing and neural prosthetic systems, and they offer advances for both scientific and applied biomedical goals. It is important to note that most of the following work was done in collaboration with other researchers, to whom this dissertation and I owe a debt of gratitude. In particular, this dissertation reflects very heavily the efforts of Krishna Shenoy, Maneesh Sahani, Byron Yu, and Vikash Gilja. I describe the relevant researchers at the beginning of each chapter.

Part I

Understanding Motor Cortical Processing

Chapter 2

Inferring Firing Rates from Neural Spike Trains

To move towards a better understanding of motor cortical processing, we first ask if we can develop a method to smooth and denoise the signals that we record from neurons in motor cortex. These neural spike trains have long presented challenges to analytical efforts due to their noisy, spiking nature. Many studies of neuroscientific and neural prosthetic importance rely on a smoothed, denoised estimate of the spike train's underlying firing rate. However, current techniques to find time-varying firing rates require *ad hoc* choices of parameters, offer no confidence intervals on their estimates, and can obscure potentially important single trial variability. Here we present a new method, based on the machine learning technology of Gaussian Process regression, for inferring probabilistically optimal estimates of firing rate functions underlying single or multiple neural spike trains. We test the performance of the method on simulated data and experimentally gathered neural spike trains, and we demonstrate improvements over conventional estimators. This work, which has been published as Cunningham et al. (2008c), was done jointly with Byron Yu, Maneesh Sahani, and Krishna Shenoy.

2.1 Introduction

Neuronal activity, particularly in cerebral cortex, is highly variable. Even when experimental conditions are repeated closely, the same neuron may produce quite different

spike trains from trial to trial. This variability may be due to both randomness in the spiking process and to differences in cognitive processing on different experimental trials. One common view is that a spike train is generated from a smooth underlying function of time (the firing rate) and that this function carries a significant portion of the neural information. If this is the case, questions of neuroscientific and neural prosthetic importance may require an accurate estimate of the firing rate. Unfortunately, these estimates are complicated by the fact that spike data gives only a sparse observation of its underlying rate. Typically, researchers average across many trials to find a smooth estimate (averaging out spiking noise). However, averaging across many roughly similar trials can obscure important temporal features (Yu et al., 2006). Thus, estimating the underlying rate from only one spike train (or a small number of spike trains believed to be generated from the same underlying rate) is an important but challenging problem.

The most common approach to the problem has been to collect spikes from multiple trials in a *peri-stimulus-time histogram* (PSTH), which is then sometimes smoothed by convolution or splines (Kass et al., 2005; DiMatteo et al., 2001). Bin sizes and smoothness parameters are typically chosen *ad hoc* (but see Shimazaki and Shinomoto (2007b); Endres et al. (2008)) and the result is fundamentally a multi-trial analysis. An alternative is to convolve a single spike train with a kernel. Again, the kernel shape and time scale are frequently *ad hoc*. For multiple trials, researchers may average over multiple kernel-smoothed estimates. Kass et al. (2005) gives a thorough review of classic methods.

More recently, point process likelihood methods have been adapted to spike data (Barbieri et al., 2001; Brown et al., 2002; Truccolo et al., 2005). These methods optimize (implicitly or explicitly) the conditional intensity function $\lambda(t|x(t), H(t))$ — which gives the probability of a spike in $[t, t + dt)$, given an underlying rate function $x(t)$ and the history of previous spikes $H(t)$ — with respect to $x(t)$. In a regression setting, this rate $x(t)$ may be learned as a function of an observed covariate, such as a sensory stimulus or limb movement. In the unsupervised setting of interest here, it is constrained only by prior expectations such as smoothness. Probabilistic methods enjoy two advantages over kernel smoothing. First, they allow explicit modeling of interactions between spikes through the history term $H(t)$ (*e.g.*, refractory periods). Second, as we will see, the probabilistic framework provides a principled way to share

information between trials and to select smoothing parameters.

In neuroscience, most applications of point process methods use maximum likelihood estimation. In the unsupervised setting, it has been most common to optimize $x(t)$ within the span of an arbitrary basis (such as a spline basis (DiMatteo et al., 2001)). In other fields, a theory of generalized Cox processes has been developed, where the point process is conditionally Poisson, and $x(t)$ is obtained by applying a link function to a draw from a random process, often a Gaussian process (GP) (*e.g.* Moller et al. (1998)). In this approach, parameters of the GP, which set the scale and smoothness of $x(t)$, can be learned by optimizing the (approximate) marginal likelihood or evidence, as in GP classification or regression. However, the link function, which ensures a nonnegative intensity, introduces possibly undesirable artifacts. For instance, an exponential link leads to a process that grows less smooth as the intensity increases.

Here, we make two advances. First, we adapt the theory of GP-driven point processes to incorporate a history-dependent conditional likelihood, suitable for spike trains. Second, we formulate the problem such that nonnegativity in $x(t)$ is achieved without a distorting link function or sacrifice of tractability. We also demonstrate the power of numerical techniques that makes application of GP methods to this problem computationally tractable. We show that GP methods employing evidence optimization outperform both kernel smoothing and maximum-likelihood point process models.

2.2 Gaussian Process Model For Spike Trains

Spike trains can often be well modeled by gamma-interval point processes (Barbieri et al., 2001; Miura et al., 2007). We assume the underlying nonnegative firing rate $x(t) : t \in [0, T]$ is a draw from a GP, and then we assume that our spike train is a conditionally inhomogeneous gamma-interval process (IGIP), given $x(t)$. The spike train is represented by a list of spike times $\mathbf{y} = \{y_0, \ldots, y_N\}$. Since we will model this

spike train as an IGIP[1], $\mathbf{y} \mid x(t)$ is by definition a renewal process, so we can write:

$$p(\mathbf{y} \mid x(t)) = \prod_{i=1}^{N} p(y_i \mid y_{i-1}, x(t)) \cdot p_0(y_0 \mid x(t)) \cdot p_T(T \mid y_N, x(t)), \tag{2.1}$$

where $p_0(\cdot)$ is the density of the first spike occuring at y_0, and $p_T(\cdot)$ is the density of no spikes being observed on $(y_N, T]$; the density for IGIP intervals (of order $\gamma \geq 1$) (see *e.g.* Barbieri et al. (2001)) can be written as:

$$p(y_i \mid y_{i-1}, x(t)) = \frac{\gamma x(y_i)}{\Gamma(\gamma)} \left(\gamma \int_{y_{i-1}}^{y_i} x(u)du \right)^{\gamma-1} \exp\left\{ -\gamma \int_{y_{i-1}}^{y_i} x(u)du \right\}. \tag{2.2}$$

The true $p_0(\cdot)$ and $p_T(\cdot)$ under this gamma-interval spiking model are not closed form, so we simplify these distributions as intervals of an inhomogeneous Poisson process (IP). This step, which we find to sacrifice very little in terms of accuracy, helps to preserve tractability. Note also that we write the distribution in terms of the inter-spike-interval distribution $p(y_i|y_{i-1}, x(t))$ and not $\lambda(t|x(t), H(t))$, but the process could be considered equivalently in terms of conditional intensity.

We now discretize $x(t) : t \in [0, T]$ by the time resolution of the experiment (Δ, here 1 ms), to yield a series of n evenly spaced samples $\mathbf{x} = [x_1, \ldots, x_n]'$ (with $n = \frac{T}{\Delta}$). The events $\mathbf{y}$ become $N + 1$ time indices into $\mathbf{x}$, with N much smaller than n. The discretized IGIP output process is now (ignoring terms that scale with Δ):

$$p(\mathbf{y} \mid \mathbf{x}) = \prod_{i=1}^{N} \left[\frac{\gamma x_{y_i}}{\Gamma(\gamma)} \left(\gamma \sum_{k=y_{i-1}}^{y_i - 1} x_k \Delta \right)^{\gamma-1} \exp\left\{ -\gamma \sum_{k=y_{i-1}}^{y_i-1} x_k \Delta \right\} \right] \cdot x_{y_0} \exp\left\{ -\sum_{k=0}^{y_0-1} x_k \Delta \right\} \cdot \exp\left\{ -\sum_{k=y_N}^{n-1} x_k \Delta \right\}, \tag{2.3}$$

where the final two terms are $p_0(\cdot)$ and $p_T(\cdot)$, respectively (Daley and Vere-Jones, 2002). Our goal is to estimate a smoothly varying firing rate function from spike

[1]The IGIP is one of a class of renewal models that works well for spike data (much better than inhomogeneous Poisson; see Barbieri et al. (2001); Miura et al. (2007)). Other log-concave renewal models such as the inhomogeneous inverse-Gaussian interval can be chosen, and the implementation details remain unchanged.

times. Loosely, instead of being restricted to only one family of functions, GP allows all functions to be possible; the choice of kernel determines which functions are more likely, and by how much. Here we use the standard squared exponential (SE) kernel. Thus, $\mathbf{x} \sim \mathcal{N}(\mu\mathbf{1}, \Sigma)$, where Σ is the positive definite covariance matrix defined by

$$\Sigma = \left\{K(t_i, t_j)\right\}_{i,j\in\{1,\dots,n\}} \quad \text{where} \quad K(t_i, t_j) = \sigma_f^2 \exp\left\{-\frac{\kappa}{2}(t_i - t_j)^2\right\} + \sigma_v^2\delta_{ij}. \tag{2.4}$$

For notational convenience, we define the hyperparameter set $\theta = [\mu; \gamma; \kappa; \sigma_f^2; \sigma_v^2]$. Typically, the GP mean μ is set to 0. Since our intensity function is nonnegative, however, it is sensible to treat μ instead as a hyperparameter and let it be optimized to a positive value. We note that other standard kernels - including the rational quadratic, Matern $\nu = \frac{3}{2}$, and Matern $\nu = \frac{5}{2}$ - performed similarly to the SE; thus we only present the SE here. For an in depth discussion of kernels and of GP, see Rasmussen and Williams (2006).

As written, the model assumes only one observed spike train; it may be that we have m trials believed to be generated from the same firing rate profile. Our method naturally incorporates this case: define $p(\{\mathbf{y}\}_1^m \mid \mathbf{x}) = \prod_{i=1}^m p(\mathbf{y}^{(i)} \mid \mathbf{x})$, where $\mathbf{y}^{(i)}$ denotes the ith spike train observed.[2] Otherwise, the model is unchanged.

2.3 Finding an Optimal Firing Rate Estimate

2.3.1 Algorithmic Approach

Ideally, we would calculate the posterior on firing rate $p(\mathbf{x} \mid \mathbf{y}) = \int_\theta p(\mathbf{x} \mid \mathbf{y}, \theta)p(\theta)d\theta$ (integrating over the hyperparameters θ), but this problem is intractable. We consider two approximations: replacing the integral by evaluation at the modal θ, and replacing the integral with a sum over a discrete grid of θ values. We first consider choosing a modal hyperparameter set (ML-II model selection, see Rasmussen and Williams (2006)), *i.e.* $p(\mathbf{x} \mid \mathbf{y}) \approx q(\mathbf{x} \mid \mathbf{y}, \theta^*)$ where $q(\cdot)$ is some approximate posterior, and

[2] Another reasonable approach would consider each trial as having a different rate function $\mathbf{x}$ that is a draw from a GP with a nonstationary mean function $\boldsymbol{\mu}(t)$. Instead of inferring a mean rate function $\mathbf{x}^*$, we would learn a distribution of means. We are considering this choice for future work.

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, p(\theta \mid \mathbf{y}) = \underset{\theta}{\operatorname{argmax}}\, p(\theta)p(\mathbf{y} \mid \theta) = \underset{\theta}{\operatorname{argmax}}\, p(\theta) \int_{\mathbf{x}} p(\mathbf{y} \mid \mathbf{x}, \theta)p(\mathbf{x} \mid \theta)d\mathbf{x}. \tag{2.5}$$

(This and the following equations hold similarly for a single observation $\mathbf{y}$ or multiple observations $\{\mathbf{y}\}_1^m$, so we consider only the single observation for notational brevity.) Specific choices for the hyperprior $p(\theta)$ are discussed in Results. The integral in Eq. 2.5 is intractable under the distributions we are modeling, and thus we must use an approximation technique. Laplace approximation and Expectation Propagation (EP) are the most widely used techniques (see Kuss and Rasmussen (2005) for a comparison). The Laplace approximation fits an unnormalized Gaussian distribution to the integrand in Eq. 2.5. Below we show this integrand is log concave in $\mathbf{x}$. This fact makes reasonable the Laplace approximation, since we know that the distribution being approximated is unimodal in $\mathbf{x}$ and shares log concavity with the normal distribution. Further, since we are modeling a non-zero mean GP, most of the Laplace approximated probability mass lies in the nonnegative orthant (as is the case with the true posterior). Accordingly, we write:

$$p(\mathbf{y} \mid \theta) = \int_{\mathbf{x}} p(\mathbf{y} \mid \mathbf{x}, \theta)p(\mathbf{x} \mid \theta)d\mathbf{x} \quad \approx \quad p(\mathbf{y} \mid \mathbf{x}^*, \theta)p(\mathbf{x}^* \mid \theta)\frac{(2\pi)^{\frac{n}{2}}}{|\Lambda^* + \Sigma^{-1}|^{\frac{1}{2}}}, \tag{2.6}$$

where $\mathbf{x}^*$ is the mode of the integrand and $\Lambda^* = -\nabla^2_{\mathbf{x}} \log\, p(\mathbf{y} \mid \mathbf{x}, \theta)\,|_{\mathbf{x}=\mathbf{x}^*}$. Note that in general both Σ and Λ^* (and $\mathbf{x}^*$, implicitly) are functions of the hyperparameters θ. Thus, Eq. 2.6 can be differentiated with respect to the hyperparameter set, and an iterative gradient optimization (we used conjugate gradients) can be used to find (locally) optimal hyperparameters. Algorithmic details and the gradient calculations are typical for GP; see Rasmussen and Williams (2006). The Laplace approximation also naturally provides confidence intervals from the approximated posterior covariance $(\Sigma^{-1} + \Lambda^*)^{-1}$.

We can also consider approximate integration over θ using the Laplace approximation above. The Laplace approximation produces a posterior approximation $q(\mathbf{x} \mid \mathbf{y}, \theta) = \mathcal{N}(\mathbf{x}^*, (\Lambda^* + \Sigma^{-1})^{-1})$ and a model evidence approximation $q(\theta \mid \mathbf{y})$ (Eq. 2.6). The approximate integrated posterior can be written as $p(\mathbf{x} \mid \mathbf{y}) =$

$E_{\theta|\mathbf{y}}[p(\mathbf{x} \mid \mathbf{y}, \theta)] \approx \sum_j q(\mathbf{x} \mid \mathbf{y}, \theta_j) q(\theta_j \mid \mathbf{y})$ for some choice of samples θ_j (which again gives confidence intervals on the estimates). Since the dimensionality of θ is small, and since we find in practice that the posterior on θ is well behaved (well peaked and unimodal), we find that a simple grid of θ_j works very well, thereby obviating MCMC or another sampling scheme. This approximate integration consistently yields better results than a modal hyperparameter set, so we will only consider approximate integration for the remainder of this report.

For the Laplace approximation at any value of θ, we require the modal estimate of firing rate $\mathbf{x}^*$, which is simply the MAP estimator:

$$\mathbf{x}^* = \underset{\mathbf{x} \succeq \mathbf{0}}{\operatorname{argmax}}\, p(\mathbf{x} \mid \mathbf{y}) = \underset{\mathbf{x} \succeq \mathbf{0}}{\operatorname{argmax}}\, p(\mathbf{y} \mid \mathbf{x}) p(\mathbf{x}). \tag{2.7}$$

Solving this problem is equivalent to solving an unconstrained problem where $p(\mathbf{x})$ is a truncated multivariate normal (but this is not the same as individually truncating each marginal $p(x_i)$; see Horrace (2005)). Typically a link or squashing function would be included to enforce nonnegativity in $\mathbf{x}$, but this can distort the intensity space in unintended ways. We instead impose the constraint $\mathbf{x} \succeq 0$, which reduces the problem to being solved over the (convex) nonnegative orthant. To pose the problem as a convex program, we define $f(\mathbf{x}) = -\log\, p(\mathbf{y} \mid \mathbf{x}) p(\mathbf{x})$:

$$\begin{aligned} f(\mathbf{x}) \;=\; & \sum_{i=1}^{N} \left(-\log\, x_{y_i} - (\gamma - 1) \log \Big(\sum_{k=y_{i-1}}^{y_i - 1} x_k \Delta \Big) \right) + \sum_{k=y_0}^{y_N - 1} \gamma x_k \Delta \\ & -\log\, x_{y_0} + \sum_{k=1}^{y_0 - 1} x_k \Delta + \sum_{k=y_N}^{n-1} x_k \Delta + \frac{1}{2} (\mathbf{x} - \mu \mathbf{1})^T \Sigma^{-1} (\mathbf{x} - \mu \mathbf{1}) + C, \end{aligned} \tag{2.8}$$

where C represents constants with respect to $\mathbf{x}$. From this form follows the Hessian

$$\nabla^2_{\mathbf{x}} f(\mathbf{x}) = \Sigma^{-1} + \Lambda \text{ where } \Lambda = -\nabla^2_{\mathbf{x}} \log\, p(\mathbf{y} \mid \mathbf{x}, \theta) = B + D, \tag{2.9}$$

where $D = \mathbf{diag}(x_{y_0}^{-2}, \ldots, 0, \ldots, x_{y_i}^{-2} \ldots, 0, \ldots, x_{y_N}^{-2})$ is positive semidefinite and diagonal. B is block diagonal with N blocks. Each block is rank 1 and associates its positive, nonzero eigenvalue with eigenvector $[0, \ldots, 0, \mathbf{b}_i^T, 0, \ldots, 0]^T$. The remaining $n - N$ eigenvalues are zero. Thus, B has total rank N and is positive semidefinite. Since Σ is positive definite, it follows then that the Hessian is also positive definite,

proving convexity. Accordingly, we can use a log barrier Newton method to efficiently solve for the global MAP estimator of firing rate $\mathbf{x}^*$(Boyd and Vandenberghe, 2004).

In the case of multiple spike train observations, we need only add extra terms of negative log likelihood from the observation model. This flows through to the Hessian, where $\nabla^2_{\mathbf{x}} f(\mathbf{x}) = \Sigma^{-1} + \Lambda$ and $\Lambda = \Lambda_1 + \ldots + \Lambda_m$, with $\Lambda_i \;\; \forall\, i \in \{1, \ldots, m\}$ defined for each observation as in Eq. 2.9.

2.3.2 Computational Practicality

This method involves multiple iterative layers which require many Hessian inversions and other matrix operations (matrix-matrix products and determinants) that cost $\mathcal{O}(n^3)$ in run-time complexity and $\mathcal{O}(n^2)$ in memory, where ($\mathbf{x} \in \mathbb{R}^n$). For any significant data size, a straightforward implementation is hopelessly slow. With 1 ms time resolution (or similar), this method would be restricted to spike trains lasting less than a second, and even this problem would be burdensome. Achieving computational improvements is critical, as a naive implementation is, for all practical purposes, intractable. Techniques to improve computational performance are a subject of study in themselves which we detail in the following chapter (Chapter 3). We give a brief outline in the following paragraph.

In the MAP estimation of $\mathbf{x}^*$, since we have analytical forms of all matrices, we avoid explicit representation of any matrix, resulting in linear storage. Hessian inversions are avoided using the matrix inversion lemma and conjugate gradients, leaving matrix vector multiplications as the single costly operation. Multiplication of any vector by Λ can be done in linear time, since Λ is a (block-wise) vector outer product matrix. Since we have evenly spaced resolution of our data $\mathbf{x}$ in time indices t_i, Σ is Toeplitz; thus multiplication by Σ can be done using Fast Fourier Transform (FFT) methods (Silverman, 1982). These techniques allow exact MAP estimation with linear storage and nearly linear run time performance. In practice, for example, this translates to solving MAP estimation problems of 10^3 variables in fractions of a second, with minimal memory load. For the modal hyperparameter scheme (as opposed to approximately integrating over the hyperparameters), gradients of Eq. 2.6 must also be calculated at each step of the model evidence optimization. In addition to using similar techniques as in the MAP estimation, log determinants and their derivatives (associated with the Laplace approximation) can be accurately approximated

by exploiting the eigenstructure of Λ.

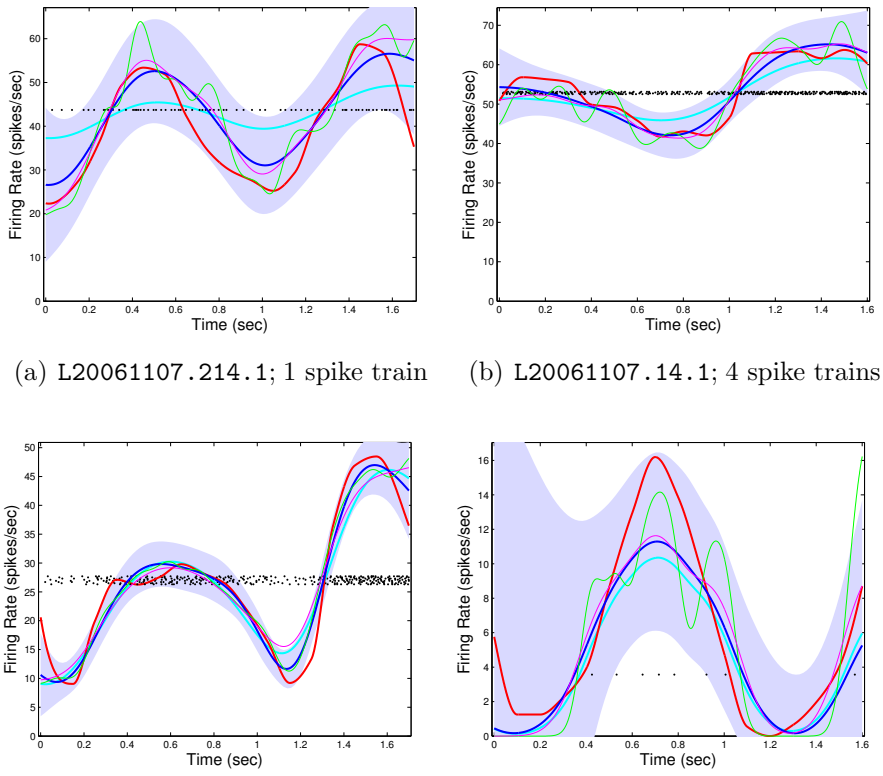
In total, these techniques allow optimal firing rates functions of 10^3 to 10^4 variables to be estimated in seconds or minutes (on a 2006 era workstation). These data sizes translate to seconds of spike data at 1 ms resolution, long enough for most electrophysiological trials. This algorithm achieves a reduction from a naive implementation which would require large amounts of memory and would require many hours or days to complete.

2.4 Results

We tested the methods developed here using both simulated neural data, where the true firing rate was known by construction, and in real neural spike trains, where the true firing rate was estimated by a PSTH that averaged many similar trials. The real data used were recorded from macaque premotor cortex during a reaching task (see Chestek et al. (2007) for experimental method). Roughly 200 repeated trials per neuron were available for the data shown here.

We compared the IGIP-likelihood GP method (hereafter, GP IGIP) to other rate estimators (kernel smoothers, Bayesian Adaptive Regressions Splines or BARS (DiMatteo et al., 2001), and variants of the GP method) using root mean squared difference (RMS) to the true firing rate. PSTH and kernel methods approximate the mean conditional intensity $\overline{\lambda}(t) = E_{H(t)}[\lambda(t|x(t), H(t))]$. For a renewal process, we know (by the time rescaling theorem (Brown et al., 2002; Daley and Vere-Jones, 2002)) that $\overline{\lambda}(t) = x(t)$, and thus we can compare the GP IGIP (which finds $x(t)$) directly to the kernel methods. To confirm that hyperparameter optimization improves performance, we also compared GP IGIP results to maximum likelihood (ML) estimates of $x(t)$ using fixed hyperparameters θ. This result is similar in spirit to previously published likelihood methods with fixed bases or smoothness parameters. To evaluate the importance of an observation model with spike history dependence (the IGIP of Eq. 2.3), we also compared GP IGIP to an inhomogeneous Poisson (GP IP) observation model (again with a GP prior on $x(t)$; simply $\gamma = 1$ in Eq. 2.3).

The hyperparameters θ have prior distributions ($p(\theta)$ in Eq. 2.5). For σ_f, κ, and γ, we set log-normal priors to enforce meaningful values (*i.e.* finite, positive, and greater than 1 in the case of γ). Specifically, we set $\log(\sigma_f^2) \sim \mathcal{N}(5, 2)$, $\log(\kappa) \sim$



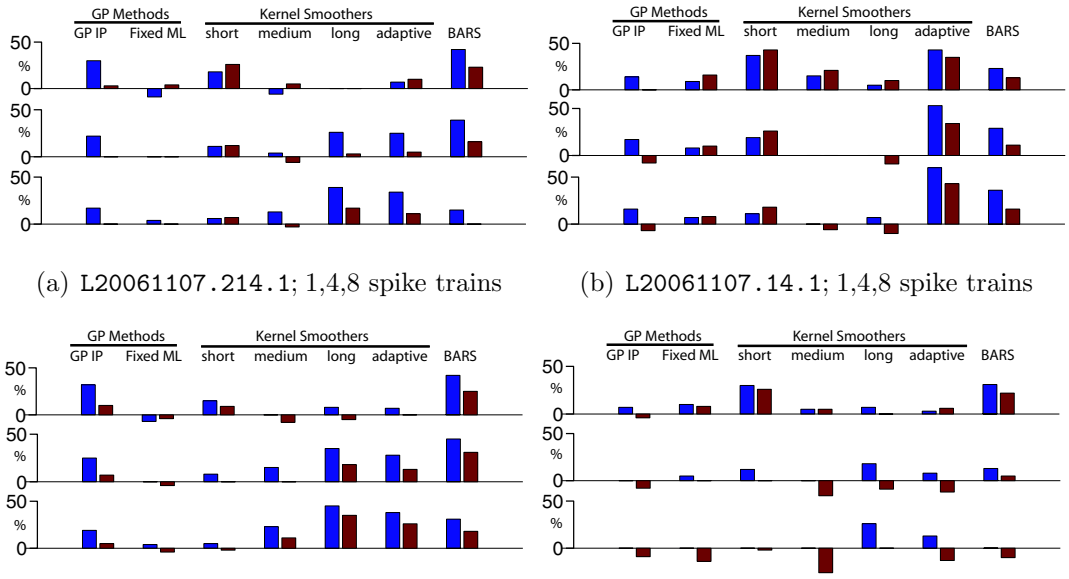
(a) `L20061107.214.1`; 1 spike train (b) `L20061107.14.1`; 4 spike trains

(c) `L20061107.151.5`; 8 spike trains (d) `L20061107.46.3`; 1 spike train

Figure 2.1: Sample GP firing rate estimates. The empirical average firing rate of the spike trains is shown in bold red. The spike trains (trains of black dots) were generated from these rates using an IGIP ($\gamma = 4$). In bold blue, we see $\mathbf{x}^*$, the results of the GP IGIP method, with light blue 95% confidence intervals. See full description in text.

$\mathcal{N}(2, 2)$, and $\log(\gamma - 1) \sim \mathcal{N}(0, 100)$. The variance σ_v can be set arbitrarily small, since the GP IGIP method avoids explicit inversions of Σ with the matrix inversion lemma (see 2.3.2). For the approximate integration, we chose a grid consisting of the empirical mean rate for μ (that is, total spike count N divided by total time T) and $(\gamma, \log(\sigma_f^2), \log(\kappa)) \in [1, 2, 4] \times [4, \ldots, 8] \times [0, \ldots, 7]$. We found this coarse grid (or similar) produced similar results to many other very finely sampled grids.

The four examples in Fig. 2.1 represent experimentally gathered firing rate profiles (according to the methods in Chestek et al. (2007)). In each of the plots, the empirical average firing rate of the spike trains is shown in bold red. For simulated spike trains, the spike trains were generated from each of these empirical average firing rates using an IGIP ($\gamma = 4$, comparable to fits to real neural data). For real neural data, the



(a) `L20061107.214.1`; 1,4,8 spike trains

(b) `L20061107.14.1`; 1,4,8 spike trains

(c) `L20061107.151.5`; 1,4,8 spike trains

(d) `L20061107.46.3`; 1,4,8 spike trains

Figure 2.2: Average percent RMS improvement of GP IGIP method (with model selection) vs. method indicated in the column title. Only significant results are shown (paired t-test, $p < 0.05$). Blue improvement bars are for simulated spike trains; red improvement bars are for real neural spike trains. See full description in text.

spike train(s) were selected as a subset of the roughly 200 experimentally recorded spike trains that were used to construct the firing rate profile. These spike trains are shown as a train of black dots, each dot indicating a spike event time (the y-axis position is not meaningful). This spike train or group of spike trains is the only input given to each of the fitting models. In thin green and magenta, we have two kernel smoothed estimates of firing rates; each represents the spike trains convolved with a normal distribution of a specified standard deviation (50 and 100 ms). We also smoothed these spike trains with adaptive kernel (Richmond et al., 1990), fixed ML (as described above), BARS (DiMatteo et al., 2001), and 150 ms kernel smoothers. We do not show these latter results in Fig. 2.1 for clarity of figures. These standard methods serve as a baseline from which we compare our method. In bold blue, we see $\mathbf{x}^*$, the results of the GP IGIP method. The light blue envelopes around the bold blue GP firing rate estimate represent the 95% confidence intervals. Bold cyan shows the GP IP method. This color scheme holds for all of Fig. 2.1.

We then ran all methods 100 times on each firing rate profile, using (separately) simulated and real neural spike trains. We are interested in the average performance

of GP IGIP vs. other GP methods (a fixed ML or a GP IP) and vs. kernel smoothing and spline (BARS) methods. We show these results in Fig. 2.2. The four panels correspond to the same rate profiles shown in Fig. 2.1. In each panel, the top, middle, and bottom bar graphs correspond to the method on 1, 4, and 8 spike trains, respectively. GP IGIP produces an average RMS error, which is an improvement (or, less often, a deterioration) over a competing method. Fig. 2.2 shows the percent improvement of the GP IGIP method vs. the competing method listed. Only significant results are shown (paired t-test, $p < 0.05$). Blue improvement bars are for simulated spike trains; red improvement bars are for real neural spike trains. The general positive trend indicates improvements, suggesting the utility of this approach. Note that, in the few cases where a kernel smoother performs better (*e.g.* the long bandwidth kernel in panel (b), real spike trains, 4 and 8 spike trains), outperforming the GP IGIP method requires an optimal kernel choice, which can not be judged from the data alone. In particular, the adaptive kernel method generally performed more poorly than GP IGIP. The relatively poor performance of GP IGIP vs. different techniques in panel (d) is considered in the Discussion section. The data sets here are by no means exhaustive, but they indicate how this method performs under different conditions.

2.5 Discussion

We have demonstrated a new method that accurately estimates underlying neural firing rate functions and provides confidence intervals, given one or a few spike trains as input. This approach is not without complication, as the technical complexity and computational effort require special care. Estimating underlying firing rates is especially challenging due to the inherent noise in spike trains. Having only a few spike trains deprives the method of many trials to reduce spiking noise. It is important here to remember why we care about single trial or small number of trial estimates, since we believe that in general the neural processing on repeated trials is not identical. Thus, we expect this signal to be difficult to find with or without trial averaging.

In this study we show both simulated and real neural spike trains. Simulated data provides a good test environment for this method, since the underlying firing rate is known, but it lacks the experimental proof of real neural spike trains (where

spiking does not exactly follow a gamma-interval process). For the real neural spike trains, however, we do not know the true underlying firing rate, and thus we can only make comparisons to a noisy, trial-averaged mean rate, which may or may not accurately reflect the true underlying rate of an individual spike train (due to different cognitive processing on different trials). Taken together, however, we believe the real and simulated data give good evidence of the general improvements offered by this method.

Panels (a), (b), and (c) in Fig. 2.2 show that GP IGIP offers meaningful improvements in many cases and a small loss in performance in a few cases. Panel (d) tells a different story. In simulation, GP IGIP generally outperforms the other smoothers (though, by considerably less than in other panels). In real neural data, however, GP IGIP performs the same or relatively worse than other methods. This may indicate that, in the low firing rate regime, the IGIP is a poor model for real neural spiking. It may also be due to our algorithmic approximations (namely, the Laplace approximation, which allows density outside the nonnegative orthant). We will report on this question in future work.

Furthermore, some neural spike trains may be inherently ill-suited to analysis. A problem with this and any other method is that of very low firing rates, as only occasional insight is given into the underlying generative process. With spike trains of only a few spikes/sec, it will be impossible for any method to find interesting structure in the firing rate. In these cases, only with many trial averaging can this structure be seen.

Several studies have investigated the inhomogeneous gamma and other more general models (*e.g.* Barbieri et al. (2001); Kass and Ventura (2003)), including the inhomogeneous inverse gaussian (IIG) interval and inhomogeneous Markov interval (IMI) processes. The methods of this paper apply immediately to any log-concave inhomogeneous renewal process in which inhomogeneity is generated by time-rescaling (this includes the IIG and several others). The IMI (and other more sophisticated models) will require some changes in implementation details; one possibility is a variational Bayes approach. Another direction for this work is to consider significant nonstationarity in the spike data. The SE kernel is standard, but it is also stationary; the method will have to compromise between areas of categorically different covariance. Nonstationary covariance is an important question in modeling and remains an

area of research (Paciorek and Schervish, 2003). Advances in that field should inform this method as well.

2.6 Acknowledgments

This work was supported by NIH-NINDS-CRCNS-R01, the Michael Flynn SGF, NSF, NDSEGF, Gatsby, CDRF, BWF, ONR, Sloan, and Whitaker. This work was conceived at the UK Spike Train Workshop, Newcastle, UK, 2006; we thank Stuart Baker for helpful discussions during that time. We thank Vikash Gilja, Stephen Ryu, and Mackenzie Risch for experimental, surgical, and animal care assistance. We thank also Araceli Navarro.

Chapter 3

Fast Computational Methods for Rate Estimation

We saw in the previous chapter that algorithmic development can improve our ability to smooth and denoise spike trains, in addition to providing benefits such as automatic smoothness detection and confidence intervals on our estimates. Unfortunately, the price paid for these benefits is the price of many machine learning methods and Gaussian process methods in particular - computational complexity. Naive implementations of these methods will become computationally infeasible in any problem of reasonable size, both in memory and run time requirements. In this chapter, we develop problem specific methods for a class of renewal processes to eliminate the memory burden and reduce the solve-time by orders of magnitude. These methods draw on techniques from large-scale optimization and numerical linear algebra. From the perspective of neural signal processing, having tractable computational methods is critical to enable the use of this method, in a real applied setting, by neuroscience researchers. While we developed this method to facilitate the motor cortical processing effort of Chapter 2, we note importantly that the following computational methods are generic to a class of point process signal processing methods. Thus, this algorithmic development is an example of applied algorithmic research informing a broader technical field. Accordingly, the following chapter is written not specifically for the neuroscientist. This work, which has been published as Cunningham et al. (2008a), was done jointly with Maneesh Sahani and Krishna Shenoy.

3.1 Introduction

Point processes with temporally or spatially varying intensity functions arise naturally in many fields of study. When the intensity function is itself a random process (often a Gaussian Process), the process is called a doubly-stochastic or Cox point process. Application domains including economics and finance (*e.g.* Basu and Dassios (2002)), neuroscience (*e.g.* Cunningham et al. (2008c)), ecology (*e.g.* Moller et al. (1998)), and others. In neuroscience, as we saw in Chapter 2, this underlying intensity function is defined over time and is typically called the firing rate function.

Given observed point process data, one can use a Gaussian Process (GP) framework to infer an optimal estimate of the underlying intensity. In this paper we consider GP prior intensity functions coupled with point process observation models. The problem of intensity estimation then becomes a modification of GP regression and inherits the computational complexity inherent in GP methods (*e.g.* Rasmussen and Williams (2006)). The data size n will grow with the length (*e.g.* total time) of the point process. Naive methods will be $\mathcal{O}(n^2)$ in memory requirements (storing Hessian matrices) and $\mathcal{O}(n^3)$ in run time (matrix inversions and determinants). At one thousand data points (such as one second of millisecond-resolution data), a naive solution to this problem is already quite burdensome on a common workstation. At ten thousand or more, this problem is for all practical purposes intractable.

While applications of doubly-stochastic point processes are numerous, there is little work proposing solutions to the serious computational issues inherent in these methods. Thus, the development of efficient methods for intensity estimation would be of broad appeal. In this paper, we do not address the appropriateness of doubly-stochastic point process models for particular applications, but rather we focus on the significant steps required to make such modeling computationally tractable. We build on previous work from both GP regression and large-scale optimization to create a considerably faster and less memory intensive algorithm for doubly-stochastic point-process intensity estimation.

As part of the GP intensity estimation problem we optimize model hyperparameters using a Laplace approximation to the marginal likelihood or evidence. This requires an iterative approach which divides into two major parts. First, at each iteration we must find a modal (MAP) estimate of the intensity function. Second, we must calculate the approximate model evidence and its gradients with respect to GP

hyperparameters. Both aspects of this problem present computational and memory problems. We develop methods to reduce the costs of both drastically.

We show that for certain classes of renewal process observation models, MAP estimation may be framed as a tractable convex program. To ensure nonnegativity in the intensity function we use a log barrier Newton method (Boyd and Vandenberghe, 2004), which we solve efficiently by deriving decompositions of matrices with known structure. By exploiting a recursion embedded in the algorithm, we avoid many costly matrix inversions. We combine these advances with large scale optimization techniques, such as conjugate gradients (CG, as used by Gibbs and MacKay (1997)) and fast fourier transform (FFT) matrix multiplication methods.

To evaluate the model evidence, as well as its gradients with respect to hyperparameters, we again exploit the structure imposed by the renewal process framework to find an exact but considerably less burdensome representation. We then show that a further approximation loses little in accuracy, but makes the cost of this computation insignificant.

Combining these advances, we are able to reduce a problem that is effectively computationally infeasible to a problem with minimal memory load and very fast solution time. $\mathcal{O}(n^2)$ memory requirements are eliminated, and $\mathcal{O}(n^3)$ computation is reduced to modestly superlinear.

3.2 Problem Overview

Define $\mathbf{x} \in \mathbb{R}^n$ to be the intensity function (the high dimensional signal of interest); $\mathbf{x}$ is indexed by input[1] time points $\mathbf{t} \in \mathbb{R}^n$. Let the observed data $\mathbf{y} = \{y_0, \ldots, y_N\} \in \mathbb{R}^{N+1}$ be a set of $N+1$ time indices into the vector $\mathbf{x}$; that is, the ith point event occurs at time y_i, and the intensity at that time is x_{y_i}. Denote all hyperparameters by θ. In general, the prior and observation models are both functions of θ. The GP framework implies a normal prior on the intensity $p(\mathbf{x} \mid \theta) = \mathcal{N}(\mu\mathbf{1}, \Sigma)$, where the nonzero mean is a sensible choice because the intensity function is constrained to be nonnegative. Thus we treat μ as a hyperparameter ($\mu \in \theta$). The positive definite covariance matrix Σ (also a function of θ) is defined by an appropriate kernel such

[1] In this work we restrict ourselves to a single input dimension (which we call time), as it aligns with the family of renewal processes in one-dimension. Some ideas here can be extended to multiple dimensions (*e.g.* if using a spatial Poisson process).

as a squared exponential or Ornstein-Uhlenbeck kernel (see Rasmussen and Williams (2006), for a discussion of GP kernels). The point-process observation model gives the likelihood $p(\mathbf{y} \mid \mathbf{x}, \theta)$. In this work, we consider renewal processes (*i.e.* one-dimensional point processes with independent event interarrival times), a family of point processes that has both been well-studied theoretically and applied in many domains (Daley and Vere-Jones, 2002).

The GP prior is log concave in $\mathbf{x}$, and the nonnegativity constraint on intensity ($\mathbf{x} \succeq 0$) is convex (constraining $\mathbf{x}$ to be nonnegative is equivalent to solving an unconstrained problem where the prior on the vector $\mathbf{x}$ is a truncated multivariate normal distribution, but this is not the same as truncating the GP prior in the continuous, infinite dimensional function space; see Horrace (2005)). Thus, if the observation model is also log concave in $\mathbf{x}$, the MAP estimate $\mathbf{x}^*$ is unique and can be readily found using a log barrier Newton method (Boyd and Vandenberghe, 2004; Paninski, 2004). Renewal processes are simply defined by their interarrival distribution $f_z(z)$. A common construction for a renewal process with an inhomogeneous underlying intensity is to use the intensity rescaling $m(t_i \mid t_{i-1}) = \int_{t_{i-1}}^{t_i} x(u)du$ (in practice, a discretized sum of $\mathbf{x}$) (Barbieri et al., 2001; Daley and Vere-Jones, 2002). Accordingly, the density for an observation of event times $\mathbf{y}$ can be defined

$$\begin{aligned} p(\mathbf{y}) &= \prod_{i=1}^{N} p(y_i \mid y_{i-1}) \\ &= \prod_{i=1}^{N} |m'(y_i \mid y_{i-1})| \; f_z(m(y_i \mid y_{i-1})) \end{aligned} \tag{3.1}$$

by a change of variables for the interarrival distribution (Papoulis and Pillai, 2002). Since $m(t)$ is a linear transformation of the intensity function (our variables of interest), the observation model obeys log concavity as long as the distribution primitive $f_z(z)$ is log concave. Examples of suitable renewal processes include the inhomogeneous Poisson, gamma interval, Weibull interval, inverse Gaussian (Wald) interval, Rayleigh interval, and other processes (Papoulis and Pillai, 2002). For this paper, we choose one of these distributions and focus on its details. However, for processes of the form above, the implementation details are identical up to the forms of the actual distributions.

To solve the GP intensity estimation, we first find a MAP estimate $\mathbf{x}^*$ given fixed

hyperparameters θ, and then we approximate the model evidence $p(\mathbf{y} \mid \theta)$ (for which we need $\mathbf{x}^*$) and its gradients in θ. Iterating these two steps, we can find the optimal model $\hat{\theta}$ (we do not integrate over hyperparameters). Finally, MAP estimation under these optimal hyperparameters $\hat{\theta}$ gives an optimal estimate of the underlying intensity. This iterative solution for $\hat{\theta}$ can be written:

$$\begin{aligned} \hat{\theta} &= \operatorname*{argmax}_{\theta} p(\theta)p(\mathbf{y} \mid \theta) \\ &\approx \operatorname*{argmax}_{\theta} p(\theta)p(\mathbf{y} \mid \mathbf{x}^*, \theta)p(\mathbf{x}^* \mid \theta)\frac{(2\pi)^{\frac{n}{2}}}{|\Lambda^* + \Sigma^{-1}|^{\frac{1}{2}}}, \end{aligned} \tag{3.2}$$

where the last term is a Laplace approximation to the intractable form of $p(\mathbf{y} \mid \theta)$, $\mathbf{x}^*$ is the mode of $p(\mathbf{y} \mid \mathbf{x})p(\mathbf{x})$ (MAP estimate), and $\Lambda^* = -\nabla^2_{\mathbf{x}} \log p(\mathbf{y} \mid \mathbf{x}, \theta) \mid_{\mathbf{x}=\mathbf{x}^*}$. The log concavity of our problem in $\mathbf{x}$ supports the choice of a Laplace approximation. Each of the two major steps in this algorithm (MAP estimation and model selection) involves computational and memory challenges. We address these challenges in Sections 3.4 and 3.5.

The computational problems inherent in GP methods have been well studied, and much progress has been made in sparsification (*e.g.* Quinonero-Candela and Rasmussen (2005)). Unfortunately, these methods do not apply directly to point process estimation, as there are no distinct training and test sets. The reader might wonder if a coarser grid would be adequate, thereby obviating the detailed methods developed here. We have found in experiments (not shown) that the sacrifice in accuracy required to allow reasonable computational tractability is large, and thus we do not consider the coarse grid a viable option. One could also consider re-expressing the problem in terms of the integrals $m(y_i \mid y_{i-1})$ appearing in Eq. 3.1. While this is possible in certain cases, it requires additional approximation. Finally, we note that the Laplace approximation is often inferior to Expectation Propagation (EP) (Kuss and Rasmussen, 2005) for GP methods. While many of the same techniques used here could also be used with EP, EP requires additional approximations and computational overhead. We find in experiments (not shown) that EP yields similar accuracy to the Laplace approximation in this domain, but EP incurs increased complexity and computational load. We describe the details of the EP implementation for this problem in Appendix B. We also note that this EP implementation pointed to an

additional finding of broad stastical interest, which we detail in depth in Chapter 4.

3.3 Model Construction

To demonstrate our fast method, we choose the specific observation model of an inhomogeneous gamma interval process (Barbieri et al., 2001) (with hyperparameter $\gamma \in \theta$, $\gamma \geq 1$). If time has been discretized with precision Δ, this can be written

$$p(\mathbf{y} \mid \mathbf{x}, \theta) = \prod_{i=1}^{N} \left[\frac{\gamma x_{y_i}}{\Gamma(\gamma)} \left(\gamma \sum_{k=y_{i-1}}^{y_i - 1} x_k \Delta \right)^{\gamma - 1} \exp\left\{ -\gamma \sum_{k=y_{i-1}}^{y_i - 1} x_k \Delta \right\} \right], \tag{3.3}$$

(where we have ignored terms that scale with Δ). Let $f(\mathbf{x}) = -\log p(\mathbf{y} \mid \mathbf{x}, \theta) p(\mathbf{x} \mid \theta)$. Our MAP estimation problem is to minimize $f(\mathbf{x})$ subject to the constraint $\mathbf{x} \succeq 0$ (nonnegativity). In the log barrier method, we consider the above problem as a sequence of convex problems where we seek to minimize, at increasing values of τ, the (unconstrained) objective function

$$f_\tau(\mathbf{x}) = f(\mathbf{x}) - \sum_{k=1}^{n} \left(\frac{1}{\tau} \right) \log(x_k) \tag{3.4}$$

which has Hessian (positive definite by our log concave construction):

$$H = \nabla^2_{\mathbf{x}} f_\tau(\mathbf{x}) = \Sigma^{-1} + \Lambda, \quad \text{where} \quad \Lambda = B + D, \tag{3.5}$$

with $D = \mathbf{diag}(x_{y_0}^{-2}, \ldots, 0, \ldots, x_{y_i}^{-2}, \ldots, 0 \ldots, x_{y_N}^{-2}) \;+\; (\frac{1}{\tau})\mathbf{diag}(x_1^{-2}, \ldots, x_n^{-2})$ being positive definite and diagonal. B is block diagonal with N blocks $\hat{B}_i$:

$$\hat{B}_i = b_i b_i^T \quad \text{where} \quad b_i = \sqrt{(\gamma - 1)} \left(\sum_{k=y_{i-1}}^{y_i - 1} x_k \right)^{-1} \mathbf{1}. \tag{3.6}$$

B is thus block rank 1 (with the positive eigenvalue in each block corresponding to the eigenvector b_i). This matrix is key, as we exploit its structure to achieve improvements in computational performance.

3.4 MAP Estimation Problem

As outlined in Section 3.2, we first find the MAP estimate $\mathbf{x}^*$ for any model defined by hyperparameters θ. The log barrier method has the intensive requirements of calculating the objective Eq. 3.4, its gradient $\mathbf{g}$ (in $\mathbf{x}$), and the Newton step $\mathbf{x}_{nt} = -H^{-1}\mathbf{g}$. Each of these calculations is $\mathcal{O}(n^3)$ in run time and $\mathcal{O}(n^2)$ in memory. We show an approach that alleviates these burdens.

3.4.1 Finding the Newton Step

First we consider the Hessian, $H = \Sigma^{-1} + \Lambda$, which itself contains the costly inverse Σ^{-1}. We would like to avoid this inversion of Σ entirely with the matrix inversion lemma (Sherman-Woodbury-Morrison formula):

$$\begin{aligned} -H^{-1} &= -(\Sigma^{-1} + \Lambda)^{-1} \\ &= -\Sigma + \Sigma R(I + R^T\Sigma R)^{-1}R^T\Sigma \end{aligned} \tag{3.7}$$

where R is any valid factorization such that $RR^T = \Lambda$. This decomposition preserves symmetry in the remaining matrix inverse (required for CG) and has advantageous numerical properties. With this form, instead of calculating $\mathbf{x}_{nt} = -H^{-1}\mathbf{g}$ directly, we need only multiply the rightmost expression in Eq. 3.7 with the gradient $\mathbf{g}$. Doing so requires the inversion $(I+R^T\Sigma R)^{-1}\mathbf{v}$ where $\mathbf{v} = R^T\Sigma\mathbf{g}$. CG allows us to avoid directly calculating matrix inverses and instead achieve the desired inversion by iteratively multiplying $(I + R^T\Sigma R)\mathbf{z}$ for different vectors $\mathbf{z}$ (Gibbs and MacKay, 1997).

It is common to precondition the CG method to reduce the number of iterations required for convergence. However, our experience with preconditioning (using both classic preconditioners and some of our own design) was that it actually degraded run-time performance. Preconditioners typically aim to improve the condition number of the Hessian, which indeed they do in this problem. However, the rapidity of CG convergence here is facilitated more by spectral concentration – many eigenvalues being equal or close to 1 – than by overall conditioning. Thus, we found it more effective to use CG inversion directly on $(I + R^T\Sigma R)$.

In general, however, finding the decomposition $\Lambda = RR^T$ is an $\mathcal{O}(n^3)$ operation, which would remove any computational benefit from this approach. For log concave

renewal processes, we can derive a valid decomposition in closed form and linear computation time. Since Λ is block diagonal, we consider only one block without loss of generality. Calling this block $\hat{\Lambda}$, we know $\hat{\Lambda} = bb^T + \hat{D}$, where $\hat{D}$ is a diagonal block of the larger diagonal matrix D, and b is defined in Eq. 3.6. $\hat{D}$ is positive definite, so $T = \hat{D}^{-\frac{1}{2}}$ satisfies $T\hat{D}T = I$ (a similarity transform). Then, calling $\tilde{b} = Tb$, we have $T\hat{\Lambda}T = \tilde{b}\tilde{b}^T + I$. With this form, we see that the general structure of $T\hat{\Lambda}T$ is preserved under the desired matrix decomposition, up to scaling of the components:

$$(\alpha\tilde{b}\tilde{b}^T + I)(\alpha\tilde{b}\tilde{b}^T + I)^T = (\alpha^2\|\tilde{b}\|^2 + 2\alpha)\tilde{b}\tilde{b}^T + I \tag{3.8}$$

and we want to choose α such that (Eq. 3.8) equals $\tilde{b}\tilde{b}^T + I$. Using the quadratic formula to find this α, we see then that

$$\tilde{R} = \left(\frac{\sqrt{1 + \|\tilde{b}\|^2} - 1}{\|\tilde{b}\|^2}\right)\tilde{b}\tilde{b}^T + I \tag{3.9}$$

satisfies $T\hat{\Lambda}T = \tilde{R}\tilde{R}^T$. Since T is diagonal, it easily inverts to $T^{-1} = \hat{D}^{\frac{1}{2}}$. Then:

$$\hat{\Lambda} = T^{-1}\tilde{R}\tilde{R}^T T^{-1} = (T^{-1}\tilde{R})(T^{-1}\tilde{R})^T = \hat{R}\hat{R}^T. \tag{3.10}$$

To be explicit, we have found that

$$\hat{R} = \left(\frac{\sqrt{1 + \|\hat{D}^{-\frac{1}{2}}b\|^2} - 1}{\|\hat{D}^{-\frac{1}{2}}b\|^2}\right)bb^T\hat{D}^{-\frac{1}{2}} + \hat{D}^{\frac{1}{2}} \tag{3.11}$$

is a valid decomposition $\hat{R}\hat{R}^T = \hat{\Lambda}$. This decomposition can be seen as a partial rank-one (blockwise) update to a Cholesky factorization (Gill et al., 1974), in that $\hat{D}$ can trivially be factorized to $\hat{D}^{\frac{1}{2}}$. The final form is not, however, a Cholesky factorization, since $\hat{R}$ is not triangular (making a triangular factor would require additional computation and the explicit representation of the Cholesky matrix).

Since all of the products needed to construct $\hat{R}$ can be formed in $\mathcal{O}(m)$ time (where m is the size of the block), and since the larger matrix R can be formed by tiling the blocks $\hat{R}$, we have a total complexity for this decomposition of $\mathcal{O}(n)$. We

can then use CG to find the solution to $(I + R^T \Sigma R)^{-1}(R^T \Sigma \mathbf{g})$. With this inversion calculated, we can perform the remaining forward multiplications in Eq. 3.7; this completes calculation of a Newton step.

In fact, we need not form the matrix R in memory. Instead, we retain each of its component elements (in Eq. 3.11), and reduce multiplication of a vector by R to a sequence of inner products and multiplications by diagonal matrices, all of which can be stored and calculated in $\mathcal{O}(n)$ time. Thus, we eliminate the need for $\mathcal{O}(n^2)$ storage, and we perform the relevant matrix multiplications in $\mathcal{O}(n)$ time. Since R can be multiplied in linear time, the complexity of multiplying vectors by $(I + R^T \Sigma R)$ depends on multiplying vectors by the covariance matrix Σ.

Since we have evenly spaced resolution of our data $\mathbf{x}$ in time indices t_i, Σ is Toeplitz. This matrix can be embedded in a larger circulant matrix, multiplication by which is simply a convolution operation of the argument vector with a row of this circulant matrix. Thus, the operation can be quickly done in $\mathcal{O}(n \log n)$ using frequency domain multiplications(Silverman, 1982). Further, we need never represent the matrix Σ; we only store the first row of the circulant matrix. Again we have eliminated $\mathcal{O}(n^2)$ memory needs. Other methods for fast kernel matrix multiplications include Fast Gauss Transforms (FGT) (Raykar et al., 2005) and kd-Trees (Shen et al., 2006; Gray and Moore, 2003). We note that the single input dimension (time) enables this Toeplitz structure, and thus an extension to multiple dimensions should use FGT or similar. The regular structure of the data points in any dimension make Σ multiplications very fast with such a method. Further, these methods avoid explicit representation of Σ. Here, the simple FFT approach for this one-dimensional problem significantly outperforms other (more general) methods in both speed and accuracy.

Finally, we note that the matrix $(I + R^T \Sigma R)$ is particularly well suited to CG. Although $R^T \Sigma R$ is full rank by definition, in practice its spectrum has very few large eigenvalues (typically fewer than N, the number of events). Loosely, the matrix looks like identity plus low rank. In practice, the CG method converges with high accuracy almost always in fewer than 50 steps (very often under 30). This is drastically fewer than the worst case of n steps (n of 10^3 to 10^4).

Instead of decomposing $\Lambda = RR^T$, one might have considered using the matrix inversion lemma to write $(\Sigma^{-1} + \Lambda)^{-1} = \Sigma - \Sigma\Lambda(\Lambda + \Lambda\Sigma\Lambda)^{-1}\Lambda\Sigma$. Indeed this valid form enables all of the CG and fast multiplication methods previously discussed. While it

may seem that this form's ease of derivation (compared to the matrix decomposition in Eq. 3.11) warrants its use in general, the matrix to be inverted is poorly conditioned compared to $(I + R^T \Sigma R)$, and thus the inversion requires more CG steps. We have found in testing that the number of CG steps can roughly double. Thus, the decomposition of Eq. 3.11 is computationally worthwhile.

In this section, we have constructed a fast method for calculating the Newton step that costs $\mathcal{O}(n \log n)$ per CG step and incurs a very small number of CG steps. Also, we have avoided explicit representation of any matrix, so that memory requirements are only linear in the data size n, allowing problem sizes of potentially millions of time steps. These two factors stand in contrast to the cubic run time and quadratic storage needs of a naive method.

3.4.2 Evaluating the Gradient and Objective

Calculating the objective $f_\tau(\mathbf{x})$ (Eq. 3.4) and its gradient (both required for the log barrier method) require finding $\Sigma^{-1}(\mathbf{x} - \mu\mathbf{1})$. Note that the kth iterate $\mathbf{x}^{(k)}$ (of the log barrier method) has the form

$$\begin{aligned}(\mathbf{x}^{(k)} - \mu\mathbf{1}) &= \mathbf{x}^{(k-1)} + t^{(k-1)}\mathbf{x}_{nt}^{(k-1)} - \mu\mathbf{1} \\ &= \sum_{j=1}^{k-1} t^{(j)}\mathbf{x}_{nt}^{(j)} \; + \; (\mathbf{x}^{(0)} - \mu\mathbf{1}) \end{aligned} \tag{3.12}$$

where $t^{(j)}$ and $\mathbf{x}_{nt}^{(j)}$ represent the jth iterates of the Newton step size t and the step $\mathbf{x}_{nt}$, and $\mathbf{x}^{(0)}$ is the algorithm initial point. The most logical starting point $\mathbf{x}^{(0)}$ is $\mu\mathbf{1}$, in which case the rightmost term in Eq. 3.12 drops out. Thus, letting $\mathbf{x}^{(0)} = \mu\mathbf{1}$ and using the form of $\mathbf{x}_{nt} = -H^{-1}\mathbf{g}$ with $-H^{-1}$ defined as in Eq. 3.7, we write:

$$\Sigma^{-1}(\mathbf{x}^{(k)} - \mu\mathbf{1}) = \sum_{j=1}^{k-1} t^{(j)} \Big(-\mathbf{g}^{(j)} + R^{(j)}(I + R^{(j)T}\Sigma R^{(j)})^{-1} R^{(j)T}\Sigma\mathbf{g}^{(j)} \Big). \tag{3.13}$$

In the earlier calculation of $\mathbf{x}_{nt}$ (Section 4.1), both of the right hand side arguments in Eq. 3.13 have already been found. As such, we have a recurrence that obviates the invertion of Σ, with no additional memory demands (Rasmussen and Williams, 2006).

The above steps reduce a naive MAP estimation (of any log concave renewal

process) that requires cubic effort and quadratic storage to an algorithm that is modestly superlinear in run time and linear in memory requirements.

3.5 Model Selection Problem

Having now found $\mathbf{x}^*$ for any hyperparameters θ, the second major part of the problem is to find the negative logarithm of our approximation to the evidence $p(\mathbf{y} \mid \theta)$ in Eq. 3.2, and its gradients with respect to θ. The approximated log evidence can be written as:

$$-\log\ p(\mathbf{y} \mid \theta) \approx -\log\ p(\mathbf{y} \mid \mathbf{x}^*) + \frac{1}{2}(\mathbf{x}^* - \mu\mathbf{1})^T\Sigma^{-1}(\mathbf{x}^* - \mu\mathbf{1}) + \frac{1}{2}\log|I + \Sigma\Lambda^*| \quad (3.14)$$

(ignoring constants). Each of these terms has an explicit and an implicit gradient with respect to θ, where the latter result from the dependence of the MAP estimate $\mathbf{x}^*$ on the hyperparameters (such implicit gradients are typical for the use of Laplace approximation in GP learning; see Rasmussen and Williams (2006), section 5.5.1). The implicit gradients in this problem are extremely computationally burdensome to calculate (requiring the trace of matrix inversions and matrix-matrix products for each element of $\mathbf{x}$). In empirical tests, we find implicit gradients to be quite small relative to the explicit gradients (often by several orders of magnitude). Ignoring these gradients is undesirable but essential to make this problem computationally feasible. Thus we consider only explicit gradients. This is a common approach for GP methods; see Rasmussen and Williams (2006).

Efficient computation of the first two terms of Eq. 3.14, as well as their gradients with respect to θ, can be achieved by the fast multiplication method and the recursion derived in Sec. 3.4. Specifically, the values of the first and second terms of Eq. 3.14 are calculated during the MAP estimation, so no additional memory or computation is necessary for them. The gradient of the first term is nonzero only with respect to γ and is linear in $\mathbf{x}$ (no matrix multiplications are required). Thus it can be quickly calculated with no additional memory demands. Computation of the gradient of the second term (the prior) can exploit the fact that we calculated $\Sigma^{-1}(\mathbf{x}^* - \mu\mathbf{1})$ in the final step of the MAP estimation. The gradient of this term with respect to μ is a

simple inner product $\mathbf{1}^T(\Sigma^{-1}(\mathbf{x}^* - \mu\mathbf{1}))$ (since we have already calculated the right side of this inner product, this computation is $\mathcal{O}(n)$ in run time and requires no additional memory). The gradient of this term with respect to a kernel hyperparameter θ_i (*e.g.* a lengthscale or variance) is:

$$\frac{d}{d\theta_i}\Big[\frac{1}{2}(\mathbf{x}^* - \mu\mathbf{1})^T\Sigma^{-1}(\mathbf{x}^* - \mu\mathbf{1})\Big] = \frac{1}{2}\big(\Sigma^{-1}(\mathbf{x}^* - \mu\mathbf{1})\big)^T\Big(\frac{d\Sigma}{d\theta_i}\Big)\big(\Sigma^{-1}(\mathbf{x}^* - \mu\mathbf{1})\big). \quad (3.15)$$

Since we have $\Sigma^{-1}(\mathbf{x}^* - \mu\mathbf{1})$, this gradient only requires one matrix-vector multiplication. $\frac{d\Sigma}{d\theta_i}$ has the same Toeplitz structure as Σ and can thus be quickly multiplied. Thus, calculating the first two terms of Eq. 3.14 and their gradients adds no complexity to the method developed so far.

Only the term $\frac{1}{2}\log|I + \Sigma\Lambda^*|$ presents difficulty. Determinants in general require $\mathcal{O}(n^2)$ memory and $\mathcal{O}(n^3)$ solve time using a Cholesky or PLU factorization, so we must consider the problem more carefully. We examine the eigenstructure of $(I+\Sigma\Lambda^*)$. Since we are not trying to find a MAP estimate, there is no log barrier term (*i.e.* let $\tau \to \infty$); thus D (from Eq. 3.5) is rank N only. This means that $\Lambda^* = B+D$ (Eq. 3.6) is block outer product plus sub rank diagonal, so it is also rank deficient with block rank 2. Thus, it has $2N$ nonzero eigenvalues (two corresponding to each of the N events, one in each block from B and one in each block from D). Using the eigenvalue decomposition $\Lambda^* = USU^T$, we see

$$\begin{aligned}\log|I + \Sigma\Lambda^*| &= \log|I + \Sigma USU^T| \\ &= \log|U^T||I + \Sigma USU^T||U| \\ &= \log|I + U^T\Sigma US|, \end{aligned} \quad (3.16)$$

since the orthogonal matrix U has determinant 1 and $U^TU = I$ by definition. Since Λ^* has rank $2N$, we know that S is diagonal with zeros on the last $n - 2N$ entries. By construction, the number of events N is much smaller than the total data size n. Since the determinant of a matrix is the product of its eigenvalues, the unit eigenvalue dimensions of $I + U^T\Sigma US$ can be ignored. We define $\overline{S}$ as the $2N \times 2N$ submatrix of S that is made up of the diagonal block with nonzero diagonal entries. Further define $\overline{U}$ as the corresponding $2N$ eigenvectors. Then, since the other dimensions of

$U^T \Sigma U S$ contribute nothing to the determinant, we have

$$\begin{aligned} \log|I + \Sigma\Lambda^*| &= \log|I + U^T \Sigma U S| \\ &= \log|I + \overline{U}^T \Sigma \overline{U}\ \overline{S}| \\ &= \log|I + \overline{\Sigma}\ \overline{S}|, \end{aligned} \tag{3.17}$$

where I is now the $2N \times 2N$ identity, and we have further defined $\overline{\Sigma} = \overline{U}^T \Sigma \overline{U}$. Computationally, $\overline{\Sigma}$ is formed by multiplying Σ with the columns of $\overline{U}$. Since Λ^* is block rank 2, both matrices $\overline{S}$ and $\overline{U}$ can be found in closed form (N rank 2 eigendecompositions, one decomposition per block). This calculation of Eq. 3.17 requires $2N$ matrix multiplications which each have a run time cost of $\mathcal{O}(n \log n)$.

We can make a small approximation that simplifies this problem even further. Typically, N of these $2N$ eigenvalues are substantially larger than the other N. Each block of Λ^* contributes two nonzero eigenvalues. The larger is due to the diagonal entry $x_{y_i}^{-2}$ (from the matrix D) and is nearly axis aligned. The smaller eigenvalue is due to the outer product vector from block $\hat{B}_i$. Examination of the denominators in the definition of $\hat{B}_i$ and D in Eqs. 3.5 and 3.6 explains the difference in magnitude, since $x_{y_i}^2$ is much smaller than the square of sums denominator in $\hat{B}_i$. We approximate the eigenvector as the y_i axis and approximate its eigenvalue as the corresponding value in Λ^*. Then $\overline{S}$ is size $N \times N$. This savings is small, but importantly we can form $\overline{\Sigma} = \overline{U}^T \Sigma \overline{U}$ simply by picking out the N rows and columns of Σ corresponding to the event times y_i.

In this formulation, we are left with matrices of size $N \times N$ only, so we have some modest number of $\mathcal{O}(N^3)$ operations; this approach is considerably faster and scales better than the exact method above. We have also reduced $\mathcal{O}(n^2)$ storage to $\mathcal{O}(N^2)$. The following section elucidates the quality of this approximation.

To calculate the gradients with respect to this log determinant term, we also use the approximation of Eq.3.17. We call our approximate gradient of this term the gradient of the approximation in Eq. 3.17. This approximation can readily be differentiated with respect to the hyperparameters (again, typical for GP; see Rasmussen and Williams (2006)). Since these approximations are matrices in the event space N (not time space n), these gradients are quickly calculated with a handful of $\mathcal{O}(N^3)$ operations and with storage of $\mathcal{O}(N^2)$.

3.6 Results and Discussion

The methods developed here maintain computational accuracy while achieving massive speed-up and the elimination of memory burden. First, we have shown a fast method that achieves an accurate approximation of the MAP estimate $\mathbf{x}^*$ in much less time than a naive method. We have made all matrix multiplications implicit, thereby eliminating the memory burden of representing full matrices. We call this piece the "MAP Estimation." Second, we found the approximate model evidence, as well as its gradients, so as to perform model selection on the hyperparameters θ. These calculations, which involved the calculation of a log determinant and its gradients (Eq. 3.17), were achieved with matrices of significantly reduced dimension, again removing the storage demands of the naive method. We call this piece the "log determinant approximation." These two pieces must be iterated (as described before Eq. 3.2) to find both the optimal model $\hat{\theta}$ and the optimal intensity $\mathbf{x}^*$. We call this iterative method (combining the two pieces above) the "full GP intensity estimation." We show here that each piece is fast and accurate, and finally that they combine to make an overall method that is considerably faster than a standard implementation, with minimal sacrifice to accuracy.

To demonstrate results, we pick six representative intensity functions, consisting of sinusoids of various amplitudes (5-100 events/second), means (15-150 events/second), frequencies (1-2 Hz), and lengths (0.5-10 seconds of millisecond resolution data, implying data sizes n of 500 to 10000). This set is by no means exhaustive, but it does indicate how this method outperforms a naive implementation in a range of scenarios. Our testing over many different intensity functions (including those in Cunningham et al. (2008c)) agrees with the results shown here. We simulate point process data $\mathbf{y}$ from these intensities, and we implement both the naive and the fast method on these process realizations.

All results are given for 2006era Linux (FC4) 64 bit workstations with 2-4GB of RAM running MATLAB (R14sp3, BLAS ATLAS 3.2.1 on AMD processors). The naive method was implemented in MATLAB. The fast method was similarly implemented in MATLAB with some use of the C-MEX interface for linear operations such as multiplication of a vector by the (implicitly represented) matrix R.

First we demonstrate the utility of our fast MAP estimation method on problems of several different sizes and with different $\mathbf{x}$. We compare the fast MAP estimation

Table 3.1: Runtime performance for fast and naive GP inference methods. Results averaged over 10 independent trials.

	Data Set					
	1	2	3	4	5	6
Data Size(n)	500	1000	1000	2000	4000	10000
Num. Events (N)[1]	20-30	30-40	140-160	55-70	55-70	140-160
MAP Estimation						
Fast Solve Time(s)	0.12	0.17	0.46	0.32	7.6	37.9
Naive Solve Time(s)	7.04	40.5	39.5	333	3704	1day[3]
Speed Up	**58**×	**232**×	**86**×	**1043**×	**493**×	**2000**×[3]
MS Error (Fast vs. Naive)[2]	4.3e-4	4.2e-4	2.1e-4	5.2e-6	6.1e-6	-
Avg. CG Iters.	6.4	5.5	16.2	8.1	29.9	49.7
Log Determinant Approximation						
Fast Solve Time(s)	6.5e-4	1.8e-3	1.9e-2	2.8e-3	2.8e-3	2.5e-2
Naive Solve Time(s)	0.24	1.02	0.97	5.7	34.7	540[3]
Speed Up	**375**×	**566**×	**52**×	**2058**×	**1.3e4**×	**2.2e4**×[3]
Avg. Acc. of Fast Approx.	99.1%	98.8%	99.8%	98.9%	99.7%	-
Avg. Model Selection Iters.	54.3	54.6	89.1	68.1	39.4	40.7
Full GP Intensity Estimation (Iterative Model Selection and MAP Estimation)						
Fast Solve Time(s)	4.4	7.1	30.3	18.7	128	423
Naive Solve Time(s)	443	3094	4548	2.4e4	1.5e5	1month[3]
Speed Up	**105**×	**451**×	**150**×	**1512**×	**1166**×	**1e4**×[3]
MS Error (Fast vs. Naive)[2]	0.10	0.03	10.8	0.01	0.01	-

[1] Entries show a range of data used.
[2] Squared norm of $x(t)$ is roughly 10^3 to 10^5, so these errors are insignificant.
[3] Unable to complete naive method; numbers estimated from cubic scaling.

to a naive implementation, demonstrating the average mean squared (MS) error (between the fast and naive estimates) and the average solve time. These results are found in the first part of Table 3.1. The squared norm of $\mathbf{x}$ is roughly 10^3 to 10^5, so the errors shown (the difference between the naive and fast methods) are vanishingly small. Thus, the fast MAP estimation gives an extremely accurate approximation of the naive MAP estimate. For all practical purposes, the fast MAP estimation method is exact.

The naive method scales in run time as the cube of data size n, as expected. The fast method and the speed-up factor do not appear to scale linearly in the data size. Indeed, run time depends heavily on the number of CG iterations required to solve the MAP estimation. This number of CG steps depends on problem size n, number of events N, and hyperparameters such as the lengthscale of the covariance matrix. Even so, major gains are achieved.

Second, we demonstrate our model selection accuracy and speed-up (the log determinant approximation). We run the full iterative fast method with both MAP estimation and evidence model selection. At each iterate of θ, we calculate evidence

and its gradients using both the fast and naive methods. In the second section of Table 3.1, we show average solution times for calculating the log determinant in both naive and fast methods, and we compare their accuracy. For the sake of brevity, we demonstrate only the calculation of $\log|I + \Sigma\Lambda^*|$, not its gradients with respect to the hyperparameters. Those calculations show very similar speed-ups and are as well approximated. Thus, the log determinant is calculated to 99-100% accuracy with the naive method, and we have a highly accurate approximation.

Finally, the full intensity estimation problem requires iterative evidence calculations and MAP estimations, so we must also demonstrate the accuracy of the full fast method versus the full naive method. The last part of Table 3.1 shows this result (Full GP Intensity Estimation). We see that all data sets converge to quite similar results in both the fast and the naive methods, and the fast method enjoys significant speed-up. The MS errors shown compare the result of the fast method to the result of the naive method and are very small compared to the squared norm of $\mathbf{x}$ (10^3 to 10^5).

We have demonstrated a method for inferring optimal intensity estimates from an observation of renewal process data, and we have exploited problem structure to make this method computationally attractive. As an extension, we also developed this fast GP technique for multiple observations $\mathbf{y}^{(i)}$ of the same underlying $\mathbf{x}$. It uses the same approach with comparable performance improvements. As such, we do not report it here.

Since we avoid all explicit representations of $n \times n$ matrices, our memory requirements are very minor for a problem of this size. The major run time improvements in Table 3.1 require effectively no loss of accuracy from an exact naive approach, and thus the additional technical complexity of this approach is well justified. Having fast, scalable methods for point process intensity estimation problems may mean the difference between theoretically interesting approaches and methods that become well used in practice.

3.7 Acknowledgments

This work was supported by NIH-NINDS-CRCNS-R01, the Michael Flynn SGF, NSF, Gatsby, CDRF, BWF, ONR, Sloan, and Whitaker. We thank Stephen Boyd for helpful discussions, Drew Haven for technical support, and Sandy Eisensee for administrative assistance.

Chapter 4

Calculating Multivariate Gaussian Probabilities

In the previous chapter, we saw that careful analysis of the firing rate estimation problem results in orders of magnitude of runtime improvement and the elimination of the associated memory burden. We also saw that the computational methods introduced in Chapter 3 are generally useful in a number of application areas outside of neuroscience. In this vein, this work also pointed to another finding of broad statistical interest. In the description of the rate estimation problem (Section 3.2), we pointed to the implementation specifics of the Expectation Propagation algorithm (Rasmussen and Williams, 2006) detailed in Appendix B. In particular, this implementation looks very much like the important and unsolved problem of calculating Gaussian probabilities. The Gaussian is the most fundamental and widely used probability distribution in existence. Univariate and multivariate Gaussians appear throughout nature, science, and engineering. Calculating Gaussian densities (evaluating the probability density function) is straightforward, but no closed-form formula exists for the cumulative distribution function. Extremely fast and accurate methods have been developed for univariate Gaussians, but a similar algorithm for multivariate Gaussians has not been found. To date, calculating multivariate Gaussian probabilities has required sophisticated numerical integration techniques, and there is a body of literature addressing this important and challenging problem. Here, we turn to the seemingly unrelated field of Bayesian inference - specifically Expectation Propagation - to develop a much simpler, more principled, and computationally faster method for

calculating multivariate probabilities with high accuracy. This method is also the first to produce an analytical solution, which is largely important in many applied settings. This chapter, which at the time of this thesis is being revised for resubmission to a machine learning journal, is sole author work. As a note to the reader, because this chapter is entirely statistical, it can be safely skipped without loss to the overall theme of understanding motor cortical processing. However, this chapter does represent a finding of broad interest that highlights again the broader impact that applied biomedical research can have to more general technical fields.

4.1 Introduction

The Gaussian (or normal) probability distribution is likely the most important probability distribution to science, engineering, and the natural world. Its ubiquity is due largely to the Central Limit Theorem (Papoulis and Pillai, 2002), which proves the tendency of many random events such as thermal noise, repeated flips of a coin, or student examination scores (to name a few) to be well described by a Gaussian distribution. The probability density of a Gaussian at any point can be readily calculated as

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{n}{2}}|K|^{\frac{1}{2}}}\exp\Big\{-\frac{1}{2}(\mathbf{x}-\mathbf{m})^T K^{-1}(\mathbf{x}-\mathbf{m})\Big\}, \tag{4.1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is a vector with n real valued elements, $\mathbf{m} \in \mathbb{R}^n$ is the mean vector, and $K \in \mathbb{R}^{n\times n}$ is the symmetric, positive semidefinite covariance matrix (in the univariate case, $K = \sigma^2$, the familiar variance term)[1]. In addition to the Central Limit Theorem, the Gaussian receives much attention because of its attractive mathematical properties - for example, unlike many probability distributions, adding and scaling Gaussians yields again a Gaussian. The moment generating function (and characteristic function) of a Gaussian is also a convenient closed-form expression. However, despite all its advantages, the Gaussian presents difficulty in that its cumulative distribution function (cdf) has no closed-form expression and is difficult to calculate. More generally, we are interested in Gaussian probabilities; that is, the probability that a random draw falls in a particular region $\mathcal{A} \in \mathbb{R}^n$.

Fig. 4.1 illustrates the probabilitiy of a Gaussian. Fig. 4.1A shows a heatmap of a

[1] Also, Gaussian processes extend the Gaussian to infinitely many random variables (Papoulis and Pillai, 2002), but we will not discuss that here.

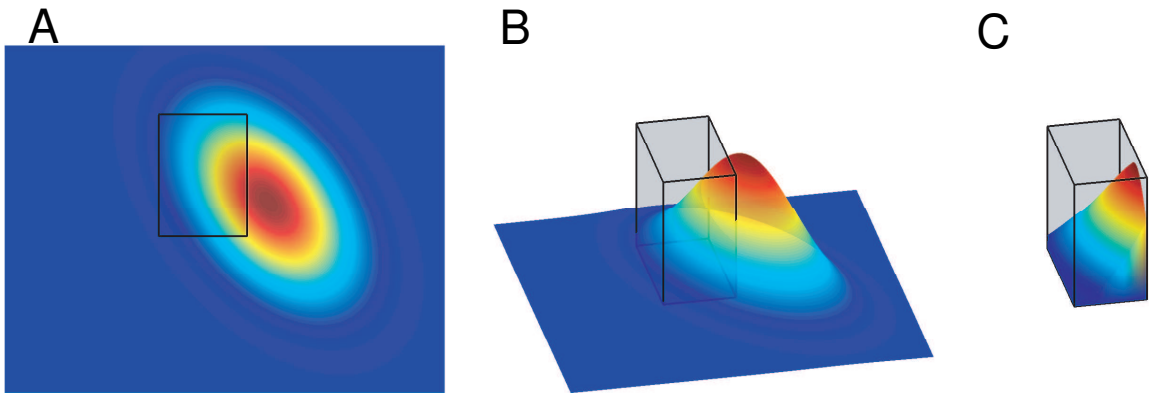


Figure 4.1: Gaussian probabilities. **(A)** heatmap of a two-dimensional Gaussian distribution and a rectangle corresponding to the region $\mathcal{A}$. **(B)** the same Gaussian in three dimensions, where the height of the curve corresponds to the probability density $p(\mathbf{x})$ as in Eq. 4.1. **(C)** The probability $F(\mathcal{A})$ is just the total mass captured above that region $\mathcal{A}$.

two-dimensional Gaussian distribution, and a rectangle corresponding to the region $\mathcal{A}$. Fig. 4.1B shows that same Gaussian in three dimensions, where the height of the curve corresponds to the probability density $p(\mathbf{x})$ as in Eq. 4.1. Isolating just that region $\mathcal{A}$ of probability mass, we see in Fig. 4.1C that the probability is just the total mass captured above that region $\mathcal{A}$. Mathematically, we write

$$F(\mathcal{A}) = \text{Prob}\{\mathbf{x} \in \mathcal{A}\} = \int_{\mathcal{A}} p(\mathbf{x})d\mathbf{x} = \int_{l_1}^{u_1} \dots \int_{l_n}^{u_n} p(\mathbf{x})dx_n \dots dx_1, \tag{4.2}$$

where $l_1, \dots, l_n$ and $u_1, \dots, u_n$ denote the upper and lower bounds of the region $\mathcal{A}$. This probability $F(\mathcal{A})$ generalizes the cdf, as the cdf can be recovered by setting $l_1 = \dots = l_n = -\infty$ (the region $\mathcal{A}$ is unbounded to the left).[2] Gaussian probabilities are as fundamentally important as the distribution itself, and the applications are equally widespread. Applications for multivariate Gaussian probabilities include statistics (Genz, 1992; Joe, 1995; Hothorn et al., 2005), economics (Boyle et al., 2005), mathematics (Hickernell and Hong, 1999), biostatistics (Thiebaut and Jacqmin-Gadda, 2004; Zhao et al., 2005), environmental science (Buccianti et al., 1998), computer science (de Klerk et al., 2000), neuroscience (Pillow et al., 2004),

[2]We note that generally the upper and lower bounds may be functions of each other, thereby defining any region of interest $\mathcal{A}$ (such as an ellipsoid or a polyhedron). In this report, we deal with hyper-rectangular regions (boxes with arbitrary position in high-dimensional space). By this choice, the bounds $l_1, \dots, l_n, u_1, \dots, u_n$ are simply numbers and not functions of each other.

machine learning (Liao et al., 2007), and more.

The vast importance of Gaussian probabilities has compelled a well-labored body of work investigating methods for calculating these quantities. The univariate cdf can be very quickly and accurately calculated using a number of techniques (*e.g.*, see (Cody, 1969)). These methods are so fast and accurate that the univariate cdf (often denoted $\phi(\cdot)$ or a scaled version of the complementary error function $\mathrm{erfc}(\cdot)$) is available with machine-level precision (as precise as a digital computer can represent any number) in many statistical computing packages (*e.g*, `normcdf` in MATLAB, `CDFNORM` in SPSS, `pnorm` in R, to name a few). Unfortunately, no similarly powerful algorithm exists for multivariate Gaussians. Previous research from math and statistics (principally due to Genz; see (Genz, 1992; Drezner and Wesolowsky, 1989; Drezner, 1994; Genz and Kwong, 2000; Genz and Brentz, 1999, 2002; Genz, 2004)) has produced advanced numerical integration methods that calculate probabilities with high accuracy (but not machine-level precision, as in the univariate case). These algorithms make a series of transformations to the Gaussian and to the region $\mathcal{A}$, using the Cholesky factor of the covariance K, the univariate Gaussian cdf and its inverse, and randomly generated points. These transformations restate the probability as a problem that can be well handled by quasi-random or lattice point numerical integration (see Appendix C.2 for more details and references). Though many important studies across many fields (Joe, 1995; Hothorn et al., 2005; Boyle et al., 2005; Hickernell and Hong, 1999; Thiebaut and Jacqmin-Gadda, 2004; Zhao et al., 2005; Buccianti et al., 1998; de Klerk et al., 2000; Pillow et al., 2004; Liao et al., 2007) have been critically enabled by this algorithm, there remains ample need across science and engineering for a simple algorithm that calculates cumulative densities and cdf values quickly and accurately for Gaussians up to large dimension. It is also essential that an algorithm produces an analytical form for $F(\mathcal{A})$, such that derivatives can be taken to optimize the probability with respect to the parameters $(\mathbf{m}, K)$ of the Gaussian. Current numerical methods do not have this important feature. In this paper, we turn to Bayesian inference, specifically the well-known Expectation Propagation (EP) framework (Minka, 2001b,a, 2005), and we develop a simple algorithm that calculates, with high accuracy, cumulative densities of low- or high-dimensional Gaussian probability distributions. This algorithm's simplicity also affords very low

computational overhead (fast run-time) and an analytical expression that can be readily manipulated. The following section (Sec. 4.2) details the algorithm, and then we provide results showing how this algorithm matches the correct probability, and that with little computational effort (Sec. 4.3).

4.2 EP-based Gaussian Probability Algorithm

Here we describe the probability algorithm, both intuitively (Sec. 4.2.1) and mathematically (Sec. 4.2.2). We also provide simple algorithm pseudocode to promote the use of this method across many fields.

4.2.1 Algorithm Intuition

All approximate Bayesian inference methods seek to approximate an intractable distribution $p(\cdot)$ (*e.g.*, a posterior distribution $p(\mathbf{x} \mid \mathbf{y})$ in some inference problem) with a tractable, convenient distribution $q(\cdot)$, where $q(\cdot)$ is, by some definition, a "good" approximation of $p(\cdot)$ (Minka, 2001b; MacKay, 2003; Bishop, 2006; Rasmussen and Williams, 2006). A "good" approximation, for example (and our case of interest in this study), might be a Gaussian distribution $q(\cdot)$ that matches its moments (zeroth - total mass, first - mean, and second - covariance) to the intractable $p(\cdot)$. Though the two distributions may not be the same at every point, their important summary statistics are indeed the same. One recent inference algorithm - Expectation Propagation (EP) (Minka, 2001b,a, 2005) - endeavors to make exactly this approximation. EP has achieved excellent results on a variety of problems and has received much attention in literature (*e.g.*, (Heskes and Zoeter, 2002; Lawrence et al., 2003; Kuss and Rasmussen, 2005; Rasmussen and Williams, 2006), to name but a few).

A key contribution of the present paper is to cast the problem of Gaussian probabilities in terms of the above-mentioned inference framework. To do so, we introduce a related distribution, the truncated multivariate normal $p_{\mathcal{A}}(\mathbf{x})$ (Horrace, 2005). This distribution has the same shape as $p(\mathbf{x})$ in Eq. 4.1 on the region $\mathcal{A}$, but it is zero

elsewhere (*i.e.*, it looks like Fig. 4.1C). Mathematically, we write:

$$p_{\mathcal{A}}(\mathbf{x}) = \begin{cases} p(\mathbf{x}) & \mathbf{x} \in \mathcal{A} \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

Since $p_{\mathcal{A}}(\mathbf{x})$ is a truncated distribution, it normalizes to $\int p_{\mathcal{A}}(\mathbf{x})d\mathbf{x} = \int_{\mathcal{A}} p(\mathbf{x})d\mathbf{x}$ (note that this is an *unnormalized* probability distribution, as it does not normalize to 1). We note that this normalizer is the probability of $p(\mathbf{x})$ over the region $\mathcal{A}$ ($F(\mathcal{A})$ as in Eq. 4.2). This is the zeroth moment (total mass) of $p(\mathbf{x})$ on the region $\mathcal{A}$. Accordingly, we can apply this distribution to the EP moment-matching method described above. EP aims to produce a distribution $q(\mathbf{x})$ with different shape but identical moments and a tractable analytical form. An appropriate choice for $q(\mathbf{x})$ is again a Gaussian (but not truncated to $\mathcal{A}$). To be clear, this method tries to approximate the truncated distribution $p_{\mathcal{A}}(\mathbf{x})$ with a *different* Gaussian $q(\mathbf{x})$ that, though not truncated, shares its zeroth (total mass), first (mean), and second (covariance) moments with $p_{\mathcal{A}}(\mathbf{x})$. Critically, since $q(\mathbf{x})$ has a tractable form, we can calculate its moments easily and exactly. And it is the calculation of this zeroth moment that yields the probability of interest $F(\mathcal{A})$. We call the method solving this moment-matching problem EP-based Gaussian Cumulative Distribution (EPGCD).

EPGCD proceeds iteratively in four simple steps, which we detail in Fig. 4.2 (Sec. 4.2.2 gives more mathematical description, simple pseudocode, and a link to our implementation). To begin ("START" in Fig. 4.2), we have a current Gaussian distribution $q(\mathbf{x})$ ("Current Global" in Fig. 4.2, whose moments we seek to match with the moments of $p_{\mathcal{A}}(\mathbf{x})$). First, recognizing the difficulty of calculating high-dimensional moments, we form *local* one-dimensional distributions, from $q(\mathbf{x})$, whose moments can be calculated. This simple calculation produces univariate Gaussians ("Cavity Locals" in Fig. 4.2, to correspond with EP literature). We want to update these cavity locals and aggregate them into a new global distribution. Ideally, we would truncate the cavity locals with each dimension of region $\mathcal{A}$ (the intervals $[l_i, u_i]$), but this would make global aggregation impossible. Instead of this hard truncation, we use a soft truncation, namely a univariate Gaussian producing the same moments as the hard truncation (univariate truncated moments are exact, using the error function $\text{erf}(\cdot)$ (Jawitz, 2004)). Thus, the second step chooses the univariate Gaussians

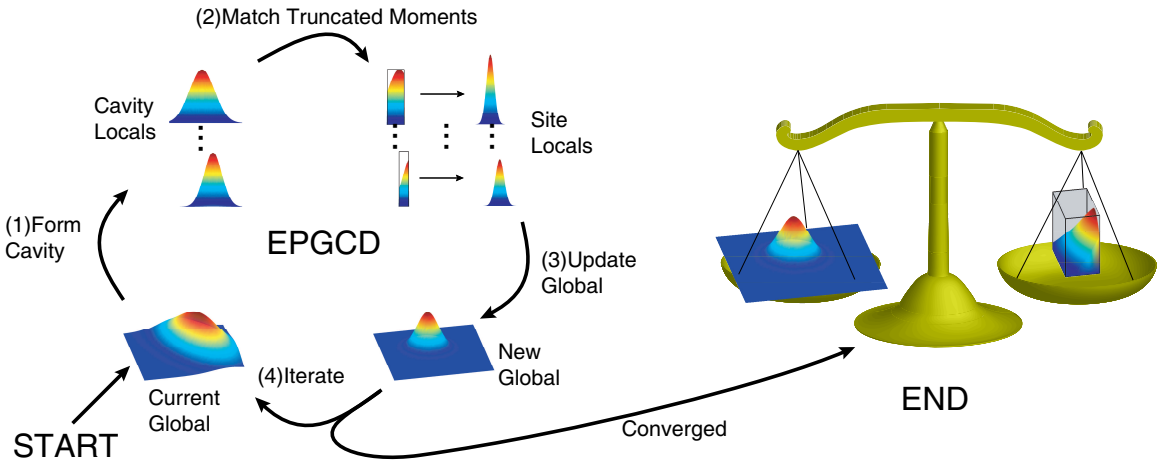


Figure 4.2: Intuition of EPGCD. The algorithm **(1)** projects to single dimensions, **(2)** matches the truncated moments to local Gaussians, **(3)** aggregates these matched moments to multiple dimensions, and **(4)** iterates, resulting in the EPGCD calculation of probability: the total mass of the untruncated EPGCD Gaussian matches $F(\mathcal{A})$, as shown by the balanced scales. See text for full description.

("Site Locals" in Fig. 4.2) such that the product of the site locals with the cavity locals (*i.e.*, this product is the soft-truncated cavity locals) have moments that match the hard truncation of the cavity locals. Third, since we have chosen Gaussian locals, we aggregate this local information into a new global $q(\mathbf{x})$ using standard properties of the Gaussian ("New Global" in Fig. 4.2). Though the cavity and site distributions are local, this step couples all the local updates and makes our new $q(\mathbf{x})$ a global estimate of the high-dimensional moments of $p_{\mathcal{A}}(\mathbf{x})$. The final (fourth) step checks the new global distribution against the previous global. If the two do not match, the algorithm iterates. If the two do match, the algorithm has converged to a global $q(\mathbf{x})$ with the same (or quite similar) moments of $p_{\mathcal{A}}(\mathbf{x})$. Balanced scales in Fig. 4.2 show that EPGCD produces $q(\mathbf{x})$ with its total mass accurately estimating that of the truncated $p_{\mathcal{A}}(\mathbf{x})$. As the total mass of $p_{\mathcal{A}}(\mathbf{x})$ is by definition the probability $F(\mathcal{A})$, finding the mass of $q(\mathbf{x})$ also calculates $F(\mathcal{A})$ with high accuracy. Since $q(\mathbf{x})$ is a regular Gaussian, its mass can calculated easily and exactly. In summary, EPGCD calculates probabilities by iteratively projecting locally, matching moments locally, and aggregating that information globally, resulting in the EPGCD calculation of the desired $F(\mathcal{A})$. The following section describes the algorithm in mathematical detail.

4.2.2 EPGCD Algorithm Details

To detail EPGCD, we first describe Expectation Propagation (EP) (Minka, 2001b,a, 2005) and why it is a sensible choice for calculating probabilities. As noted in the text, EP aims to replace an intractable $p(\cdot)$ with a tractable $q(\cdot)$ that approximates $p(\cdot)$ well. A natural measure for the quality of the approximation is Kullback-Leibler (KL) divergence (Cover and Thomas, 1991). Specifically, if we choose $q(\cdot)$ to be a high-dimensional Gaussian, then the choice of $q(\cdot)$ that minimizes $KL(p \parallel q)$ (*i.e.*, the best approximation) is the $q(\cdot)$ with the same zeroth (total mass), first (mean), and second (covariance) moments as $p(\cdot)$.[3] If we are trying to calculate a probability $F(\mathcal{A})$, then we equivalently seek the zeroth moment of $p_{\mathcal{A}}(\mathbf{x})$ as defined in Eq. 4.3. As such, minimizing this global KL divergence is an appropriate and sensible method to calculate a probability. The proof that minimizing KL divergence results in moment matching (for this problem) can be found in Appendix C.1. Unfortunately, minimizing global KL divergence directly is in many cases intractable. This fact motivated the creation of the EP algorithm, which seeks to do this KL minimization (or at least approximately do it) by iteratively minimizing the KL divergence of local, single dimensions of $q(\cdot)$ with respect to $p(\cdot)$.

In the following, We use standard EP notation (as in (Rasmussen and Williams, 2006)) to describe the EPGCD algorithm. We also use the standard notation $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ to denote a Gaussian (that normalizes to 1) with mean $\boldsymbol{\mu}$ and covariance Σ (in any dimension n, thus $\mathcal{N}(\mu, \sigma^2)$ denotes a univariate Gaussian). First, let us define the region $\mathcal{A}$ as the hyperrectangle bounded at each dimension i by lower and upper bounds l_i and u_i. We are interested in finding the probability on this region $\mathcal{A}$ of the high-dimensional Gaussian $p(\mathbf{x}) = \mathcal{N}(\mathbf{m}, K)$. We can define $\mathbf{m} = 0$ with no loss of generality, as the region $\mathcal{A}$ can be shifted by $\mathbf{m}$ if $\mathbf{m} \neq 0$ (having $\mathbf{m} = 0$ will slightly simplify the presentation below). Then, we define sites $t_i(x_i) = \delta(x_i \in [l_i, u_i])$; that is, a function on the ith dimension that is 1 inside the interval $[l_i, u_i]$, and 0 otherwise. Then, by this definition, we can rewrite the definition of probability (Eq. 4.2) as:

[3] Note that minimizing $KL(p \parallel q)$ is different than minimizing $KL(q \parallel p)$, and the latter is another potential choice which results in variational approximation methods (Rasmussen and Williams, 2006; Bishop, 2006; MacKay, 2003). However, minimizing $KL(q \parallel p)$ does not result in moment matching and is thus inappropriate for probability calculation.

$$F(\mathcal{A}) = \int_{\mathcal{A}} p(\mathbf{x})d\mathbf{x} = \int p_{\mathcal{A}}(\mathbf{x})d\mathbf{x} = \int p(\mathbf{x})\prod_{i=1}^{n} t_i(x_i)d\mathbf{x}, \tag{4.4}$$

and we see that $p_{\mathcal{A}}(\mathbf{x}) = p(\mathbf{x})\prod_{i=1}^{n} t_i(x_i)$. These hard truncations $t_i(x_i)$, which we call the *sites*, make this integral intractable, so we replace the $t_i(x_i)$ with soft truncations $\tilde{t}_i(x_i)$ (*site local approximations*). We choose $\tilde{t}_i(x_i)$ to be scaled univariate Gaussians: $\tilde{t}_i(x_i) = \tilde{Z}_i\mathcal{N}(\tilde{\mu}_i, \tilde{\sigma}_i^2)$ (note these $\tilde{t}_i(\mathbf{x}_i)$ normalize to $\tilde{Z}_i$). Then, our global approximation is $q(\mathbf{x}) = p(\mathbf{x})\prod_{i=1}^{n} \tilde{t}_i(x_i) \approx p_{\mathcal{A}}(\mathbf{x})$. Because $q(\mathbf{x})$ is a Gaussian (the product of Gaussians is another scaled Gaussian (Rasmussen and Williams, 2006)), we can easily find its zeroth, first, and second moments (we are primarily interested in the zeroth moment $F(\mathcal{A})$). Choosing the parameters of these soft truncations $\tilde{t}_i(\mathbf{x}_i)$ to minimize KL divergence is the purpose of EPGCD.

Many of the subsequent EPGCD equations are standard for EP, so we will not rederive them, for the sake of brevity. We assume we already have the Gaussian approximation $q(\mathbf{x}) = Z\mathcal{N}(\boldsymbol{\mu}, \Sigma)$, and now we want to refine our site approximations $\tilde{t}_i(x_i)$. The key idea is to project $q(\mathbf{x})$ onto the ith dimension so we can consider what $\tilde{t}_i(x_i)$ should be included to minimize KL divergence in that dimension. To do this, we form the *cavity* local distribution for the ith dimension (labelled **(1)** in Fig. 4.2):

$$\begin{aligned} q_{-i}(x_i) &= \int p(\mathbf{x}) \prod_{j\neq i} \tilde{t}_j(x_j)d\mathbf{x} = \mathcal{N}(\mu_{-i}, \sigma_{-i}^2), \\ \text{where} \quad \mu_{-i} &= \sigma_{-i}^2(\sigma_i^{-2}\mu_i - \tilde{\sigma}_i^{-2}\tilde{\mu}_i), \quad \text{and} \quad \sigma_{-i}^2 = (\sigma_i^{-2} - \tilde{\sigma}_i^{-2})^{-1}, \end{aligned} \tag{4.5}$$

(as in (Rasmussen and Williams, 2006), Eq. 3.56). This univariate Gaussian is the projection of all of $q(\mathbf{x})$ *except* for the ith site approximation (soft truncation) $\tilde{t}_i(x_i)$, which we want to update. Including this site approximation, the full projection of $q(\mathbf{x})$ onto the ith dimension (the marginal) is $q_{-i}(x_i)\tilde{t}_i(x_i)$. Since we want to match the global moments of $q(\mathbf{x})$ to those of $p_{\mathcal{A}}(\mathbf{x})$, the moments of the projection must also match. We consider the true site (the hard truncation) $t_i(x_i)$, which would result in the projection of the ith dimension $q_{-i}(x_i)t_i(x_i)$. We then minimize this univariate KL divergence resulting in matching the moments of $q_{-i}(x_i)\tilde{t}_i(x_i)$ to the moments of $q_{-i}(x_i)t_i(x_i)$. The product $q_{-i}(x_i)t_i(x_i)$ is simply a univariate truncated Gaussian (see upper right picture in Fig. 4.2), and the moments of this product can be calculated

exactly (Jawitz, 2004) using the error function erf(·):

$$\begin{aligned} q_{-i}(x_i)t_i(x_i) &\approx \hat{q}(x_i) = \hat{Z}_i\mathcal{N}(\hat{\mu}_i, \hat{\sigma}_i^2), \\ \text{where} \quad \hat{Z}_i &= \frac{1}{2}\Big(\text{erf}(\alpha) - \text{erf}(\beta)\Big), \\ \hat{\mu}_i &= \mu_{-i} + \frac{1}{\hat{Z}_i}\Big(\frac{\sigma_{-i}}{\sqrt{2\pi}}\big(\exp\{-\alpha^2\} - \exp\{-\beta^2\}\big)\Big), \quad \text{and} \\ \hat{\sigma}_i^2 &= \mu_{-i}^2 + \sigma_{-i}^2 + \\ &\quad \frac{\sigma_{-i}}{\sqrt{2\pi}}\big((l_i + \mu_{-i})\exp\{-\alpha^2\} - (u_i + \mu_{-i})\exp\{-\beta^2\}\big) - \hat{\mu}_i^2, \end{aligned} \tag{4.6}$$

where we have everywhere above used the shorthand $\alpha = \frac{l_i - \mu_{-i}}{\sqrt{2}\sigma_{-i}}$ and $\beta = \frac{u_i - \mu_{-i}}{\sqrt{2}\sigma_{-i}}$. These moments will match the moments of $q_{-i}(x_i)\tilde{t}_i(x_i)$ (our desired KL minimization) if $\tilde{t}_i(x_i)$ is chosen as:

$$\begin{aligned} \tilde{t}_i(x_i) &= \tilde{Z}_i\mathcal{N}(\tilde{\mu}_i, \tilde{\sigma}_i^2), \\ \text{where} \quad \tilde{\mu}_i &= \tilde{\sigma}_i^2(\hat{\sigma}_i^{-2}\hat{\mu}_i - \sigma_{-i}^{-2}\mu_{-i}), \qquad \tilde{\sigma}_i^2 = (\tilde{\sigma}_i^{-2} - \sigma_{-i}^{-2})^{-1}, \\ \tilde{Z}_i &= \hat{Z}_i\sqrt{2\pi}\sqrt{\sigma_{-i}^2 + \tilde{\sigma}_i^2}\exp\Big\{\frac{1}{2}(\mu_{-i} - \tilde{\mu}_i)^2/(\sigma_{-i}^2 + \tilde{\sigma}_i^2)\Big\}. \end{aligned} \tag{4.7}$$

(as in (Rasmussen and Williams, 2006), Eq. 3.59). While these equations may look complicated, they are all simple outcomes of standard manipulations on Gaussian random variables. This local site update (and the local updates from all other dimensions $j \neq i$) is then aggregated globally (labelled **(3)** in Fig. 4.2) to form a new global Gaussian approximation $q(\mathbf{x}) = p(\mathbf{x})\prod_{i=1}^n \tilde{t}_i(x_i)$, which will have the updated form:

$$\begin{aligned} q(\mathbf{x}) &= Z\mathcal{N}(\boldsymbol{\mu}, \Sigma), \\ \text{where} \quad \boldsymbol{\mu} &= \Sigma\tilde{\Sigma}^{-1}\tilde{\boldsymbol{\mu}}, \qquad \Sigma = (K^{-1} + \tilde{\Sigma}^{-1})^{-1}, \\ \text{and} \quad \log Z &= -\frac{1}{2}\log|K + \tilde{\Sigma}| - \frac{1}{2}\tilde{\boldsymbol{\mu}}(K + \tilde{\Sigma})^{-1}\tilde{\boldsymbol{\mu}} + \sum_{i=1}^n \log\tilde{Z}_i - \frac{n}{2}\log(2\pi) \end{aligned} \tag{4.8}$$

(as in (Rasmussen and Williams, 2006), Eqs. 3.53 and 3.65), where we have used $\tilde{\boldsymbol{\mu}}$ to denote the vector of site means $\tilde{\mu}_i$, and $\tilde{\Sigma}$ to denote the diagonal matrix with diagonal elements of the site variances $\tilde{\sigma}_i^2$. Again, these equations follow from standard normalization when considering a product of Gaussian distributions. Note that Z is the

zeroth moment of $q(\mathbf{x})$, which is the EPGCD result for the probability $F(\mathcal{A})$. Critically, the form of Z can be readily differentiated (we refer the reader to (Rasmussen and Williams, 2006), section 5.5.2, and (Seeger, 2003) for specific details).[4]

Importantly, we now have an updated $q(\mathbf{x}) = Z\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ that has minimized the KL divergence of the ith dimension and has updated the global $q(\mathbf{x})$ to hopefully reduce $KL\big(p_{\mathcal{A}}(\mathbf{x}) \parallel q(\mathbf{x})\big)$ as well. Though there is no proof for when EP methods converge to this global KL minimizer, our results (Sec. 4.3, to follow) suggest that EPGCD does converge very close to this minimizer (99.8% to 100% accuracy). We iterate this update procedure (labelled **(4)** in Fig. 4.2) until the algorithm converges (that is, when the parameters of $q(\mathbf{x})$ stop changing). One can choose to update the global $q(\mathbf{x})$ after every site update, or, as we do, after a full sweep of all n sites. In our experience, EPGCD always converges quickly (less than 10 steps through all n sites typically, even for high n) to the same result, regardless of this choice of when to update the global $q(\mathbf{x})$. Accordingly, we present Fig. 4.2 as doing all the site updates in parallel, and then aggregating this information globally. The steps above lead to the following simple EPGCD pseudocode:

Algorithm 1 EPGCD: Calculate $F(\mathcal{A})$, the probability of $p(\mathbf{x})$ on region $\mathcal{A}$.

Initialize with any $q(\mathbf{x})$ defined by $Z, \boldsymbol{\mu}, \Sigma$.
while $q(\mathbf{x})$ has not converged **do**
  for $i \leftarrow 1 : n$ **do**
    form cavity local $q_{-i}(x_i)$ by Eq. 4.5.
    calculate moments of $q_{-i}(x_i)t_i(x_i)$ by Eq. 4.6.
    choose $\tilde{t}_i(x_i)$ so $q_{-i}(x_i)\tilde{t}_i(x_i)$ matches above moments by Eq. 4.7.
  end for
  update global $Z, \boldsymbol{\mu}, \Sigma$ with new site approximations $\tilde{t}_i(x_i)$ using Eq. 4.8.
end while
return Z, the total mass of $q(\mathbf{x})$.

In practice, researchers have found that it helps to reparameterize the EP steps to ensure numerical stability (for example, as simple as considering $\frac{1}{\sigma^2_{-i}}$ instead of σ^2_{-i}). In our code, we followed the implementation details of (Rasmussen and Williams, 2006), section 3.6.3 (which aligns with several of the equations given above - only the moment calculations $\hat{\mu}, \hat{\sigma}^2, \hat{Z}$ are different; see Eq. 4.6).

[4] It has been proven that the derivatives of the cavity and site parameters (with respect to the parameters of the Gaussian) are 0, so only the explicit dependencies need to be considered (Seeger, 2003; Rasmussen and Williams, 2006).

4.3 Results

For all the studies that use EP and consider its theoretical backing, no formal proof has emerged for when EP will converge and, if it converges, when EP will match all the global moments of the distribution $q(\mathbf{x})$ to those moments of $p(\mathbf{x})$. Experts have conjectured that EP converges (Rasmussen and Williams, 2006) for the class of log-concave probability distributions (our problem is also log-concave). Also, EP is known to converge correctly in some cases (an outcome of its connection to Loopy Belief Propagation - see (Pearl, 1988; Weiss and Freeman, 2001; Minka, 2001b,a; Yedidia et al., 2002; Minka, 2005); interesting connections have also been made to statistical physics and free-energy formulations (Csato et al., 2002; Opper and Winther, 2001, 2000)). There are trivial cases where EP (and, as a result, EPGCD) converges correctly, such as any diagonal covariance K (isotropic or uncorrelated covariance). Indeed, calculating a cumulative density for a diagonal K is simply a product of univariate probabilities, so this case is uninteresting. Here we focus only on the more interesting (and common) case of full (correlated) covariances K. In lieu of a proof of the correctness of EPGCD, we turn to empirical results to demonstrate the quality of this method. Though there are infinite possible Gaussians (choices of n, $\mathbf{m}$, and K) and infinite possible probability regions $\mathcal{A}$, we applied the algorithm to many different distributions across many different regions and many different dimensions (our specific method for selecting these regions and distributions is described in Appendix C.3). In all cases we have tested, EPGCD converges very quickly. Further, EPGCD, when started from many different initializations, always converged to the same solution, strongly suggesting the optimization is unimodal. As the ground truth for any probability test case is unknown (hence the motivation for this algorithm), evaluating the accuracy of EPGCD results is tricky. The aforementioned numerical integration method (we call this the "Genz" method, and we describe this further in Appendix C.2) of (Genz, 1992; Genz and Kwong, 2000; Genz and Brentz, 1999, 2002; Genz, 2004) can be run at different numbers of integration points, which produces different levels of accuracy and very different computational run-times. Running Genz for a very long time (5×10^5 integration points) produces a result that claims high accuracy. Though this estimate is not guaranteed to be correct, we believe it to be highly accurate (it claims over 99.99% accuracy, and indeed we have found it more accurate than much more time-consuming Monte-Carlo (MC) methods (Bishop, 2006)).

Table 4.1: Average performance of probability estimators. We used 1000 cases per dimension n (100 for $n = 500$ and $n = 1000$).

	Avg. Run Times(s)			Avg. % Err. vs. High Acc. Est.[1]			Speedup[3]	
Gaussian		Genz			Genz		EPGCD vs.	
Dimension	EPGCD	Fast[2]	Acc.[3]	EPGCD	Fast	Acc.	Fast	Acc.
$n = 2$	0.001	0.005	0.01	0.02%	0.05%	0.00%	4×	8×
$n = 3$	0.006	0.02	0.06	0.07%	0.20%	0.00%	4×	11×
$n = 4$	0.007	0.03	0.10	0.14%	0.43%	0.00%	5×	14×
$n = 5$	0.005	0.02	0.07	0.17%	0.79%	0.01%	5×	15×
$n = 10$	0.006	0.03	0.11	0.23%	1.40%	0.02%	6×	19×
$n = 20$	0.003	0.04	0.13	0.14%	1.25%	0.02%	16×	46×
$n = 50$	0.011	0.13	0.35	0.05%	0.75%	0.01%	12×	32×
$n = 100$	0.066	0.30	0.75	0.02%	0.54%	0.01%	5×	11×
$n = 500$	8.43	19.0	48.0	0.00%	0.25%	0.00%	2×	6×
$n = 1000$	41.0	28.5	119	0.00%	0.16%	0.00%	1×	3×

[1] A very slow, very accurate Genz method ($5{\times}10^5$ points, 5×10^4 for $n = 500, 1000$) for comparison.
[2] Genz method run with 50 points (very fast, but less accurate); see Appendix C.2.
[3] Genz method run with 5000 points (slower, but accurate); see Appendix C.2.

We call this result in all cases the "high accuracy estimate."

One might also wonder how EPGCD performs compared to the Genz method run at more reasonable numbers of integration points. We compare EPGCD to: (1)"Fast Genz," a very fast but less accurate method (Genz method with 50 integration points), and (2)"Accurate Genz," a slower but accurate method (Genz method with 5000 integration points). Average run-times and errors vs. the high accuracy estimation are shown in Table 4.1.

In Table 4.1, each row of the table corresponds to normal distributions of increasing dimension (number of jointly Gaussian random variables). At a given dimension n, we chose 1000 different regions $\mathcal{A}$ and Gaussians (100 for $n = 500$ and $n = 1000$, see Appendix C.3). The first group of columns shows run-times for EPGCD and the Genz methods (Fast and Accurate). The second group of columns indicates the average error (in %) between each method and the high accuracy estimate. Finally, the last column indicates the speed-up of EPGCD vs. the Fast and Accurate Genz methods. Table 4.1 shows, across a diverse set of Gaussians and probability regions, that EPGCD calculates $F(\mathcal{A})$ with high (between 99.8% and 100%) accuracy. It calculates these quantities much more quickly and accurately than the Fast Genz method. Comparing to the Accurate Genz method, EPGCD is more than an order of magnitude faster, at the price of some decreased accuracy. We note here a potential bias in these results. Since we use the Genz method for our high accuracy estimate

of ground truth, errors inherent in the Genz algorithm may not be shown in the Fast and Accurate Genz methods, artificially penalizing EPGCD. Even still, these results demonstrate that EPGCD calculates probabilities very quickly and with high accuracy.

4.4 Discussion

As EPGCD and Genz both calculate multivariate probabilities quite accurately, some comparison should be made between other features of the algorithms. First, EPGCD involves four simple iterative steps that do nothing more than moment match and exploit properties of the Gaussian distribution, whereas the Genz method requires several problem transformations and sophisticated numerical integration. Accordingly, we claim EPGCD is a simpler algorithm, which in turn leads to simpler implementations in software and generally much faster running time (see final columns of Table 4.1). We show roughly an order of magnitude speed up on average when using EPGCD, depending on the problem size and the number of integration points used in the Genz method. We caution that these speed up numbers can vary depending on the computing platform (we used Linux Fedora Core 4 with 64 bit, 2.2-2.4GHz AMD processors and 2-4GB of RAM), coding language (we used MATLAB R2007), and choices made in implementation. For example, increasing the number of integration points in the Genz method can slow its run-time by orders of magnitude without changing the result (when going from, for example 5×10^3 to 5×10^4 integration points), or will do little for the run time but quite increase the error (when going from 50 to fewer integration points).

In addition to its simplicity, EPGCD is a well-principled approach. We cast the probability problem as a calculation of the zeroth moment of a truncated distribution. EPGCD then logically proceeds by projecting a distribution $q(\cdot)$ to each dimension, matching local moments, and then aggregating that information globally to form a new $q(\cdot)$. The Genz method, on the other hand, makes several intelligent choices in its transformation of the problem, but eventually it resorts to brute-force numerical integration. Because of its principled approach, EPGCD also naturally produces the mean and covariance of the truncated distribution[5], which the Genz method does

[5]our testing indicates that EPGCD finds these values as accurately as the total mass $F(\mathcal{A})$.

not. These quantities are of critical importance in Bayesian inference, and should be valuable to many of the studies across many fields that require calculating probabilities (Joe, 1995; Hothorn et al., 2005; Boyle et al., 2005; Hickernell and Hong, 1999; Thiebaut and Jacqmin-Gadda, 2004; Zhao et al., 2005; Buccianti et al., 1998; de Klerk et al., 2000; Pillow et al., 2004; Liao et al., 2007). Finally, since EPGCD produces an analytical result, the probability result $F(\mathcal{A})$ can be readily differentiated with respect to the Gaussian parameters $(\mathbf{m}, K)$. This feature is hugely important in many circumstances: for example, if $F(\mathcal{A})$ is a probability of classification error that we want to minimize, we may want to optimize the position of the Gaussian with respect to that error (and any other constraints). Thus, the principled foundation for EPGCD allows deeper insight and more opportunities for manipulation than numerical methods.

In some ways also, however, EPGCD is not as developed as numerical methods. Future studies should focus on generalizing the region $\mathcal{A}$ to non-hyperrectangular regions, and future theoretical work should focus on proving the convergence and accuracy of EPGCD and EP-based methods in general. Our tested cases always converged quickly to a highly accurate result. We tried to find any pattern to the discrepancies between EPGCD and the Genz methods to indicate potential strengths or weaknesses of EPGCD. Of all the features we tested, condition number of the covariance matrix K tended to correlate positively with the error between EPGCD and Genz. This suggests that either EPGCD or the Genz method may be less robust to poorly conditioned matrices. We also considered angular offset of the principle covariance axes (*i.e.*, the extent to which the covariance is not axis-aligned), the volume of the region $\mathcal{A}$, and the eccentricity of the region $\mathcal{A}$, but these factors seem to have little effect on the difference between the results produced by these methods.

4.5 Conclusion

To summarize, we have adapted a popular method from Bayesian inference to produce EPGCD: a method for calculating Gaussian probabilities with high accuracy. EPGCD distinguishes itself from previous methods in its simplicity and principled

However, as those calculations are not relevant to the present study, we do not further discuss this aspect of EPGCD.

construction, which allow simple implementation, very fast run-time, and the flexibility for probability to be optimized based on problem parameters. These features should allow a broad adoption of this method to a range of Gaussian probability problems. probabilities of Gaussian distributions are as fundamental as the Gaussian probabilities themselves. Thus, increasing our understanding of and ability to calculate these quantities is of prominent importance to science and engineering as a whole.

4.6 Acknowledgments

The author thanks Krishna Shenoy for valuable discussions, manuscript review, and support. The author thanks Maneesh Sahani for valuable discussions and manuscript review, and Matt Kaufman for artistic assistance. The author also thanks Drew Haven for technical support and Sandy Eisensee for administrative assistance. This work was supported by the Michael Flynn Stanford Graduate Fellowship.

Chapter 5

Low-dimensional Single-trial Analysis of Neural Population Activity

We here return to the central problem of understanding motor cortical processing. In Chapter 2, we developed a method to process a spike train, recorded on a single experimental trial, into a smooth, denoised signal that is more amenable to analytical efforts. However, single spike trains are fundamentally quite noisy and only a single view of a highly complex system, so it is likely that no algorithm will be able to answer all interesting scientific questions with a single neuron alone. In this chapter, we extend this signal processing approach (and use some of the computational developments of Chapters 3 and 4) across many simultaneously-recorded neurons to extract a population-level signature of neural activity. We consider the problem of extracting smooth, low-dimensional *neural trajectories* that summarize the activity recorded simultaneously from many neurons on individual experimental trials. Beyond the benefit of visualizing the high-dimensional, noisy spiking activity in a compact form, such trajectories can offer insight into the dynamics of the neural circuitry underlying the recorded activity. Current methods for extracting neural trajectories involve a two-stage process: the spike trains are first smoothed over time, then a static dimensionality reduction technique is applied. We first describe extensions of the two-stage methods that allow the degree of smoothing to be chosen in a principled way, and that account for spiking variability which may vary both across neurons and across time.

We then present a novel method for extracting neural trajectories, Gaussian-process factor analysis (GPFA), which unifies the smoothing and dimensionality reduction operations in a common probabilistic framework. We applied these methods to the activity of 61 neurons recorded simultaneously in macaque premotor and motor cortices during reach planning and execution. By adopting a goodness-of-fit metric that measures how well the activity of each neuron can be predicted by all other recorded neurons, we found that the proposed extensions improved the predictive ability of the two-stage methods. The predictive ability was further improved by going to GPFA. From the extracted trajectories, we directly observed a convergence in neural state during motor planning, an effect that was shown indirectly by previous studies. We then show how such methods can be a powerful tool for relating the spiking activity across a neural population to the subject's behavior on a single-trial basis. Finally, to assess how well the proposed methods characterize neural population activity when the underlying timecourse is known, we performed simulations which revealed that GPFA performed tens of percent better than the best two-stage method. This work, which has been published as Yu et al. (2009a), was done jointly with Byron Yu, Gopal Santhanam, Stephen Ryu, Krishna Shenoy, and Maneesh Sahani. Byron led this project effort, and I was involved heavily in algorithm design, algorithm implementation, data analysis, and construction of the manuscript.

5.1 Introduction

5.1.1 Motivation for Single-trial Analysis of Neural Population Activity

Neural responses are typically studied by averaging noisy spiking activity across multiple experimental trials to obtain firing rates that vary smoothly over time. However, if the neural responses are more a reflection of internal processing rather than external stimulus drive, the timecourse of the neural responses may differ on nominally identical trials. This is particularly true of behavioral tasks involving perception, decision making, attention, or motor planning. In such settings, it is critical that the neural data not be averaged across trials, but instead be analyzed on a trial-by-trial basis (Arieli et al., 1996; Nawrot et al., 1999; Horwitz and Newsome, 2001; Ventura

et al., 2005; Briggman et al., 2006; Yu et al., 2006; Churchland et al., 2007; Jones et al., 2007; Czanner et al., 2008).

The importance of single-trial analyses can be simply illustrated by considering a classic perceptual decision-making study by Newsome and colleagues (Horwitz and Newsome, 2001). In this study, they trained monkeys to report the direction of coherent motion in a stochastic random-dot display. Especially in low coherence conditions, they observed that neurons in the superior colliculus appeared to jump between low and high firing rate states, suggesting that the subject may have vascillated between the two possible directional choices. For the *same* random-dot stimulus, the times at which the firing rates jumped appeared to differ from one trial to the next. Such vascillations may also underlie other perceptual and decision-making tasks, including binocular rivalry (Leopold and Logothetis, 1996), structure-from-motion (Bradley et al., 1998; Dodd et al., 2001), somatosensory discrimination (de Lafuente and Romo, 2005), and action selection (Cisek and Kalaska, 2005). Most of these studies provide indirect evidence that the timecourse of the subject's percept or decision differed on nominally identical trials. Such trial-to-trial differences cannot be eliminated by additional monkey training, since the stimuli are designed to be ambiguous and/or operate near the subject's perceptual threshold.

In the dot-discrimination (Horwitz and Newsome, 2001) and binocular rivalry (Leopold and Logothetis, 1996) studies, the authors attempted to segment single spike trains based on periods of high and low firing rates. In general, it is very difficult to accurately estimate the time or rate at which the firing rate changes based on a single spike train. If one is able to record from multiple neurons simultaneously and the activity of these neurons all reflect a common neural process (e.g., the subject's percept or choice), then one might be able to more accurately estimate the timecourse of the subject's percept or choice on a single trial. Indeed, developments in multi-electrode (Kipke et al., 2008) and optical imaging (Kerr and Denk, 2008) technologies are making this a real possibility. However, it is currently unclear how to best leverage the statistical power afforded by simultaneously-recorded neurons (Brown et al., 2004) to extract behaviorally-relevant quantities of interest (e.g., the timecourse of the subject's percept or internal decision variable) on a single-trial basis.

In this work, we develop analytical techniques for extracting single-trial neural timecourses by leveraging the simultaneous monitoring of large populations of

neurons. The approach adopted by recent studies is to consider each neuron being recorded as a noisy sensor reflecting the time-evolution of an underlying neural process (Smith and Brown, 2003; Stopfer et al., 2003; Brown et al., 2005; Levi et al., 2005; Mazor and Laurent, 2005; Briggman et al., 2005; Broome et al., 2006; Yu et al., 2006; Sasaki et al., 2007; Carrillo-Reid et al., 2008; Bathellier et al., 2008). The goal is to uncover this underlying process by extracting a smooth, low-dimensional *neural trajectory* from the noisy, high-dimensional recorded activity. The activity of each neuron tends to vary significantly between trials, even when experimental conditions are held constant. Some of this variability is due to processes internal to the neuron, such as channel noise in membranes and biochemical noise at synapses (Faisal et al., 2008). But some portion of the variability reflects trial-to-trial differences in the time-evolution of the network state, which may in turn represent different computational paths and may lead to different behavioral outcomes. As it reflects the network state, we expect this component of the variability to be shared amongst many (or all) of the neurons that make up the network. The techniques that we develop here seek to embody this shared activity in a neural trajectory, which represents our best estimate of the time-evolution of the neural state. The neural trajectory provides a compact representation of the high-dimensional recorded activity as it evolves over time, thereby facilitating data visualization and studies of neural dynamics under different experimental conditions. In principle, relative to the high-dimensional recorded activity, such a parsimonious description should bear clearer and stronger relationships with other experimentally-imposed or measurable quantities (e.g., the presented stimulus or the subject's behavior; see the "bouncing ball analogy" in Yu et al., 2006).

Fig. 5.1 illustrates how such an approach may provide insights into the neural mechanisms underlying perception, decision making, attention, and motor planning. Fig. 5.1*A* considers the perceptual and decision tasks described above, in which there are two possible percepts or choices. Applying the analytical methods developed in this work to the activity of multiple neurons recorded simultaneously may reveal different switching timecourses on different trials. In this example (Fig. 5.1*A*, lower left), on trial 1, the subject's percept switched from one choice to another, then back to the first. On trial 2, the percept began to switch, stopped between the two choices, then completed its switch. These switching timecourses can be viewed in terms of single-neuron firing rates (Fig. 5.1*A*, lower right), where the two neurons are shown to

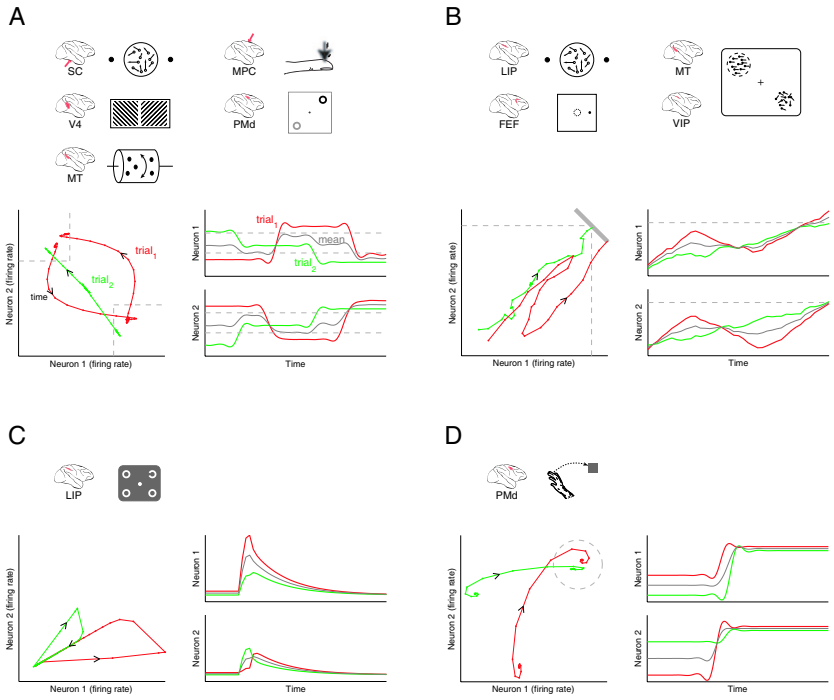


Figure 5.1: Conceptual illustration showing how the analytical methods presented in this work can be applied to different behavioral tasks, including those involving perception, decision making, attention, and motor planning. The neural mechanisms underlying these behavioral tasks may involve *A*: switching between two possible percepts or decisions, *B*: rising to threshold, *C*: decaying along a single slow mode, or *D*: converging to an attractor. Each panel includes icons of the relevant behavioral tasks and brain areas (top), single-trial neural trajectories in the firing rate space of two neurons (lower left), and corresponding firing rate profiles (both single-trial and trial-averaged) for each neuron (lower right).

have anti-correlated firing rates. Note that these firing rate profiles are estimated by leveraging the simultaneously-recorded spike trains across a neural population on a single-trial basis. In this case, the timecourse obtained by averaging neural responses across trials (gray) is not representative of the timecourse on any individual trial (red and green traces). Beyond relating the extracted trajectory (Fig. 5.1*A*, lower left) to the subject's perceptual report or decision on a trial-by-trial basis, such trajectories allow us to ask questions about the dynamics of switching percepts across the neural population. For example, how long does it take to switch between one percept and another? Does it take longer to switch in one direction than the other? Does switching in one direction follow the same path as switching in the other direction? Can regions in firing rate space be defined corresponding to each percept or choice? If so, what is

the shape of these regions?

Fig. 5.1*B* considers a different class of dynamics: rise-to-threshold. Shadlen and colleagues have shown that single neurons in lateral intraparietal (LIP) cortex appear to integrate sensory evidence until a threshold is reached, at which time a decision is made (Roitman and Shadlen, 2002). By grouping trials based on reaction time, they found that the firing rates approached threshold more quickly on trials with short reaction times than on trials with long reaction times. Similar effects were found in frontal eye field prior to saccade initiation (Hanes and Schall, 1996), and middle temporal and ventral intraparietal areas during motion detection (Cook and Maunsell, 2002). In other words, the timecourse of the neural response differed on nominally identical trials in all of these studies. To investigate trial-to-trial differences, correlations were identified between single-trial estimates of firing rate and reaction time (Roitman and Shadlen, 2002; Cook and Maunsell, 2002). However, due to the limited statistical power in a single spike train, most analyses in these previous studies relied on grouping trials with similar reaction times. If one is able to record from multiple neurons simultaneously, one can then leverage the statistical power across the neural population to more accurately estimate single-trial response timecourses. This could potentially uncover even stronger relationships between neural activity and behavioral measurements, such as reaction time. Fig. 5.1*B* shows two trials in which the decision variable crosses threshold at similar times; thus, the subject would be expected to show similar reaction times on these two trials. However, the timecourse of the decision variable was quite different on each trial. On trial 1, the decision variable rose quickly toward threshold, then headed back towards baseline (perhaps due to contrary evidence) before finally rising to threshold. On trial 2, the decision variable rose slowly, but steadily, toward threshold. Such subtle differences between trials would be difficult to see based on single spike trains and would be washed out had the neural activity been averaged across trials (Fig. 5.1*B* lower right, gray trace). By leveraging simultaneous recordings across a neural population, we may be able to uncover such effects on a single-trial basis and gain further insight into the dynamics of decision processes[1].

[1] While the concept of a threshold is well-defined for a single neuron, it is unclear how it generalizes for a population of neurons. Is the decision made when any one of the neurons in the population reaches threshold (i.e., when the neural trajectory first hits a dotted line in Fig. 5.1*B*, lower left)? Or, is it the sum of the activity across the population that matters (i.e., when the neural trajectory first hits the thick gray bar in Fig. 5.1*B*, lower left)? It may be possible to address such questions

Another potential application of the methods developed in this work is to behavioral tasks that involve attention, which is typically not tied to observable quantities in the outside world. Using a GO/NOGO memory saccade task with visual distractors, Goldberg and colleagues showed that neural activity (averaged across trials and neurons) in LIP indicates the attentionally advantaged part of the visual field (Bisley and Goldberg, 2003). For the same stimulus (target, distractor, and probe), the subject showed different behavioral responses (GO or NOGO) on different trials, where the proportion of correct responses depended on the time, location, and contrast of the probe. If multiple neurons could be monitored simultaneously in LIP, the methods developed in this work could be used to track the instantaneous state of the subject's attention on a single-trial basis, which in turn could be related to the subject's behavioral response. For example, it may be that the same distractor kicks the neural state out farther on some trials (Fig. 5.1*C*, lower left, red trace) than others (green trace), thereby conferring a longer-lasting attentional advantage at the distractor (see Fig. 2 in Ganguli et al., 2008, for a detailed explanation of the state space trajectories shown in Fig. 5.1*C*, lower left). Might it be possible to map out the probability of a correct behavioral response (GO or NOGO) for different neural states at the time of the probe? Such an approach could shed light on the dynamics of attentional shifts between different visual locations and how it impacts behavior.

Finally, we consider the arm movement system, which serves as our experimental testbed for exploring analytical methods for extracting single-trial neural timecourses. We have previously shown that the across-trial variability of neural responses in premotor cortex drops during motor preparation (Churchland et al., 2006b). This finding suggested that single-trial neural trajectories might converge during motor preparation, in attractor-like fashion, as illustrated in Fig. 5.1*D* (lower left). Although we have previously hypothesized such a convergence of trajectories (see Fig. 1 in Churchland et al., 2006b), we have not been able to directly view this effect due to a lack of appropriate analytical methods for extracting trajectories on a single-trial basis. By studying how the neural state evolves from an initially variable baseline state toward a consistent planning state, and relating aspects of the trajectory to the subject's behavior, we can gain insight into how movements are prepared and executed. While the analytical methods developed here are potentially applicable to

using the methods developed here.

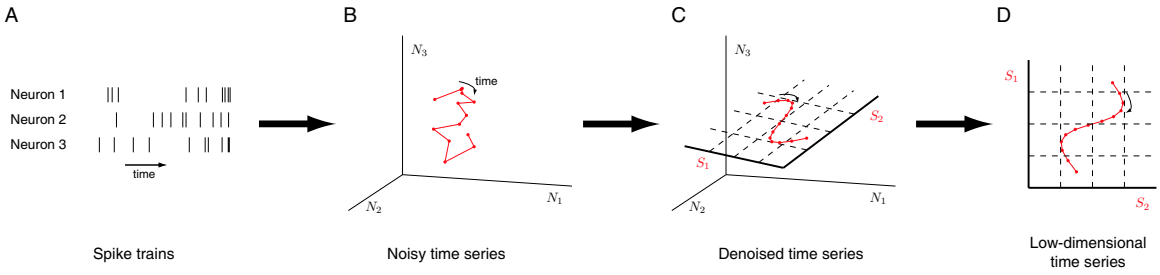


Figure 5.2: Extracting a neural trajectory from multiple spike trains. *A*: Spike trains recorded simultaneously from three neurons. *B*: The time-evolution of the recorded neural activity plotted in a three-dimensional space, where each axis measures the instantaneous firing rate of a neuron (e.g., N_1 refers to neuron 1). *C*: The neural trajectory (a "denoised" version of the trajectory in *B*) is shown to lie within a two-dimensional space with coordinates S_1 and S_2. *D*: The neural trajectory can be directly visualized in the low-dimensional space and be referred to using its low-dimensional coordinates (S_1, S_2).

many different experimental settings, as exemplified in Fig. 5.1, we demonstrate the utility of the developed methods in the context of motor preparation and execution in this work.

For each of the examples shown in Fig. 5.1, there are ways to detect (or indirectly view) trial-to-trial differences in the neural responses, including computing streak indices (Horwitz and Newsome, 2001), estimating firing rates from a single spike train (Roitman and Shadlen, 2002; Cook and Maunsell, 2002), and measuring the across-trial variability of neural responses (Churchland et al., 2006b). In all of these cases, however, what one really wants is a direct view of the time-evolution of the neural response on single trials. In this report, we present analytical methods that can extract such single-trial timecourses from neural population activity.

5.1.2 Existing Methods for Extracting Neural Trajectories

Fig. 5.2 shows conceptually how a neural trajectory relates to a set of simultaneously-recorded spike trains. Suppose that we are recording simultaneously from three neurons, whose spike trains are shown in Fig. 5.2*A*. Although the following ideas hold for larger numbers of neurons, we use only three neurons here for illustrative purposes. We define a high-dimensional space, where each axis measures the instantaneous firing rate of a neuron being monitored (Fig. 5.2*B*). At any given time, the activity of

the neural population is characterized by a single point in this space. As the activity of the neural population evolves over time, a noisy trajectory is traced out. The goal is to extract a corresponding smooth neural trajectory that embodies only the shared fluctuations (termed *shared variability*) in firing rate across the neural population (Fig. 5.2*C*). What is discarded in this process are fluctuations particular to individual neurons (termed *independent variability*), which presumably reflect noise processes involved in spike generation that are internal to the neuron[2]. Due to the correlated activity across the neural population, the neural trajectory may not explore the entire high-dimensional space; in other words, the neural system may be using fewer degrees of freedom than the number of neurons at play. If this is true, then we would like to identify a lower dimensional space (shown as a two-dimensional plane denoted by grid lines in Fig. 5.2*C*) within which the neural trajectory lies. The neural trajectory can then be directly visualized in the low-dimensional space, and be referred to equivalently using its high-dimensional (N_1, N_2, N_3) or low-dimensional (S_1, S_2) coordinates (Fig. 5.2*D*).

A simple way to extract neural trajectories is to first estimate a smooth firing rate profile for each neuron on a single trial (e.g., by convolving each spike train with a Gaussian kernel), then apply a static dimensionality reduction technique (e.g., principal components analysis, PCA) (Nicolelis et al., 1995; Levi et al., 2005). The signal flow diagram for these so-called *two-stage methods* is shown in Fig. 5.3*A*. Smooth firing rate profiles may also be obtained by averaging across a small number of trials (if the neural timecourses are believed to be similar on different trials) (Stopfer et al., 2003; Brown et al., 2005; Mazor and Laurent, 2005; Broome et al., 2006), or by applying more advanced statistical methods for estimating firing rate profiles from single spike trains (DiMatteo et al., 2001; Ventura et al., 2005; Cunningham et al., 2008c). Numerous linear and non-linear dimensionality reduction techniques exist, but to our knowledge only PCA (Levi et al., 2005; Mazor and Laurent, 2005; Nicolelis et al., 1995) and locally linear embedding (LLE) (Stopfer et al., 2003; Brown et al., 2005; Broome et al., 2006; Roweis and Saul, 2000) have been used to extract neural trajectories. Smoothed firing rate trajectories based on pairs of simultaneously-recorded neurons without dimensionality reduction have also been studied (Aksay et al., 2003).

[2]While the independent variability indeed feeds back into the network and can affect the aggregate network state, we assume that such effects are small.

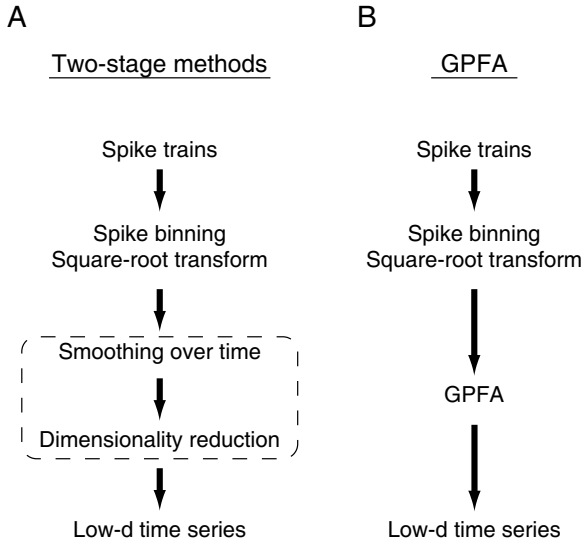


Figure 5.3: Signal flow diagram of how neural trajectories are extracted from spike trains using *A*: two-stage methods and *B*: Gaussian-process factor analysis (GPFA). Whereas the smoothing and dimensionality reduction operations are performed sequentially with the two-stage methods (dotted box), they are performed simultaneously using GPFA. For the two-stage methods, smoothing can either be performed directly on spike trains or on binned spike counts.

The two-stage methods have been fruitfully applied in studies of the olfactory system, where the presentation of an odor sets off a timecourse of neural activity across the recorded population. To understand how the population response varies under different experimental conditions (e.g., different presented odors), one could attempt to directly compare the recorded spike trains. However, this quickly becomes unmanageable as the number of neurons and the number of experimental conditions increases. Instead, a neural trajectory can be extracted for each trial condition (typically averaged across a small number of trials) and compared in a low-dimensional space. This approach has been adopted to study the population response across different odor identities (e.g., Brown et al., 2005), concentrations (Stopfer et al., 2003), durations (Mazor and Laurent, 2005), and sequences (Broome et al., 2006). Dynamical behaviors resembling fixed points (Mazor and Laurent, 2005) and limit cycles (Bathellier et al., 2008) have also been identified. In these studies, hypotheses were generated based on the visualized trajectories, then tested using the high-dimensional recorded activity. Without the low-dimensional visualizations, many of these hypotheses would have remained unposed and, therefore, untested.

5.1.3 Methodological Advances Proposed Here

While two-stage methods have provided informative low-dimensional views of neural population activity, there are several aspects that can be improved. (i) Because the smoothing and dimensionality reduction are performed sequentially, there is no way for the dimensionality reduction algorithm to influence the degree or form of smoothing used. This is relevant both to the identification of the low-dimensional space, as well as to the extraction of single-trial neural trajectories. (ii) PCA and LLE have no explicit noise model and, therefore, have difficulty distinguishing between changes in the underlying neural state (i.e., shared variability) and spiking noise (i.e., independent variability). (iii) For kernel smoothing, the degree of smoothness is often arbitrarily chosen. We would instead like to learn the appropriate degree of smoothness from the data. With probabilistic methods, a principled approach would be to ask what is the degree of smoothness that maximizes the probability of having observed the data at hand. Unfortunately, kernel smoothing, PCA, and LLE are all non-probabilistic methods, so such standard parameter-learning techniques are not applicable. One may try to get around this problem by applying kernel smoothing, followed by a probabilistic dimensionality reduction technique (e.g., probabilistic PCA; Roweis and Ghahramani, 1999; Tipping and Bishop, 1999), which does assign probabilities to data. However, the problem with this scheme is that these probabilities correspond to the smoothed data (the input to the probabilistic dimensionality reduction technique), rather than the unsmoothed data (the input to the kernel smoother). Because the smoothed data change depending on the degree of smoothness chosen, the resulting probabilities are not comparable. (iv) The same kernel width is typically used for all spike trains across the neural population, which implicitly assumes that the population activity evolves with a single timescale. Because we don't know *a priori* how many timescales are needed to best characterize the data at hand, we would like to allow for multiple timescales.

In this work, we first propose extensions of the two-stage methods that can help to address issues (ii) and (iii) above. We summarize these extensions here; details can be found in *Methods*. For (ii), we explore dimensionality reduction algorithms possessing different explicit noise models and consider the implications of the different noise assumptions. We find that an effective way to combat spiking noise (whose

variance may vary both across neurons and across time) is to employ the square-root transform (Kihlberg et al., 1972) in tandem with factor analysis (FA) (Everitt, 1984). Taking the square root of the spike counts serves to approximately stabilize the spiking noise variance. FA is a dimensionality reduction technique related to PCA that, importantly, allows different neurons to have different noise variances. Although non-linear dimensionality reduction techniques with explicit noise models have been developed (for a probabilistic LLE-inspired algorithm, see Teh and Roweis, 2003), we consider only linear mappings between the low-dimensional neural state space and the high-dimensional space of recorded activity in this work for mathematical tractability. For (iii), we adopt a goodness-of-fit metric that measures how well the activity of each neuron can be predicted by the activity of all other recorded neurons, based on data not used for model fitting. This metric can be used to compare different smoothing kernels and allows for the degree of smoothness to be chosen in a principled way. An advantage of this metric is that it can be applied in both probabilistic and non-probabilistic settings. In *Results*, we will use this as a common metric by which different methods for extracting neural trajectories are compared.

Next, we develop Gaussian-process factor analysis (GPFA), which unifies the smoothing and dimensionality reduction operations in a common probabilistic framework. GPFA takes steps toward addressing all of the issues (i)–(iv) described above, and is shown in *Results* to provide a better characterization of the recorded population activity than the two-stage methods. Because GPFA performs the smoothing and dimensionality reduction operations simultaneously (Fig. 5.3*B*) rather than sequentially (Fig. 5.3*A*), the degree of smoothness and the relationship between the low-dimensional neural trajectory and the high-dimensional recorded activity can be jointly optimized. GPFA allows for multiple timescales, whose optimal values can be found automatically by fitting the GPFA model to the recorded activity. Unlike the two-stage methods, GPFA assigns probabilities to the *unsmoothed* data, which allows the timescale parameters to be optimized using standard maximum likelihood techniques. As with FA, GPFA specifies an explicit noise model that allows different neurons to have different noise variances. The time series model involves Gaussian processes (GP), which only require the specification of a parameterized correlation structure of the neural state over time.

A critical assumption when attempting to extract a low-dimensional neural trajectory is that the recorded activity evolves within a low-dimensional manifold. Previous studies have typically assumed that the neural trajectories lie in a three-dimensional space for ease of visualization. In this work, we will investigate whether this low-dimensional assumption is justified in the context of reach planning and execution. If so, we will attempt to identify the appropriate dimensionality. Furthermore, we will systematically compare different analytical methods for extracting neural trajectories.

We first detail the two-stage methods, GPFA, and dynamical system approaches to extracting neural trajectories. Next, the behavioral task and the neural recordings in premotor and motor cortices are described. We then apply the different extraction techniques to study the dimensionality and timecourse of the recorded activity during reach planning and execution.

Preliminary versions of this work have previously appeared (Yu et al., 2008, 2009b).

5.2 Methods

5.2.1 Two-stage Methods

The two-stage methods involve first estimating a smooth firing rate profile for each neuron on a single trial, then applying a static dimensionality reduction technique. For the simplest two-stage method, the firing rate estimates are obtained by convolving each spike train with a Gaussian kernel. These firing rate estimates, taken across all simultaneously-recorded neurons, defines a trajectory in the high-dimensional space of recorded activity (Fig. 5.2*B*). This trajectory is represented by a series of datapoints (red dots in Fig. 5.2*B*). In the simplest case, the datapoints of many such trajectories are then passed to PCA, which identifies the directions of greatest variance in the high-dimensional space. The high-dimensional datapoints are then projected into the low-dimensional space defined by the principal component axes (conceptualized by the S_1S_2 plane shown in Fig. 5.2*C*). The projected datapoints can then be strung back together over time to obtain a low-dimensional neural trajectory (Fig. 5.2*D*).

Although PCA is widely used and simple to apply, it is problematic when applied to neural data because neurons with higher firing rates are known to show higher count

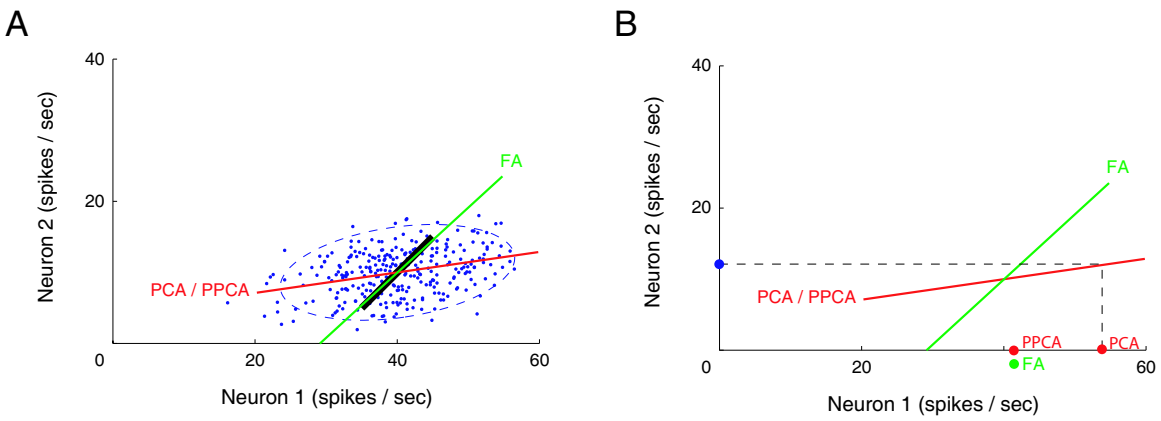


Figure 5.4: Simulation comparing PCA, PPCA, and FA in the two-neuron case. *A*: FA (green line) is better able to uncover the true firing rate relationship (black line) between the two neurons than PCA / PPCA (red line). The noise-corrupted observations (blue dots) and two SD covariance ellipse (dashed blue) are shown. *B*: Leave-neuron-out model prediction for PCA (red dot labeled 'PCA'), PPCA (red dot labeled 'PPCA'), and FA (green dot). Each model predicts the activity of neuron 1, given the activity of neuron 2 (blue dot).

variability (i.e., their Poisson-like behavior) (Dayan and Abbott, 2001). Because PCA finds directions in the high-dimensional space of greatest variance, these directions tend to be dominated by the neurons with the highest firing rates. This is illustrated in Fig. 5.4*A* using two neurons. In this simulation, the underlying firing rates of the two neurons are perfectly correlated (black line), representing the ground truth. What we are able to observe, however, are noise-corrupted versions (blue dots) of the underlying firing rates, where the noise is assumed to be independent for each neuron. These blue dots are analogous to the red dots in Fig. 5.2*B*. The goal is to recover the true relationship between the activity of the two neurons (black line) using only the noise-corrupted datapoints (blue dots). Once this relationship is identified (i.e., an estimate of the black line), the datapoints are then projected onto the estimated line, yielding "denoised" firing rate estimates for the two neurons. In this case, identifying the true one-dimensional relationship between the two neurons provides a succinct account of the noisy recorded activity.

Compared with neuron 2, neuron 1 has a higher mean firing rate and correspondingly higher firing rate variability in Fig. 5.4*A*. The higher variability leads to an elongation of the covariance ellipse (dashed blue) along the horizontal direction. When

PCA is applied to the datapoints, the direction of highest variance (red line) is identified. Comparing the red and black lines, it is apparent that PCA provides a poor estimate of the true firing rate relationship between the two neurons. The reason for the mismatch is that PCA implicitly assumes that the noise variance is isotropic (i.e., the same for all neurons regardless of mean firing rate). PCA erroneously identifies a direction that is biased in the direction of high noise variance, in this case along horizontal axis corresponding to neuron 1. The same incorrect direction would be found by probabilistic PCA (PPCA) (Roweis and Ghahramani, 1999; Tipping and Bishop, 1999), which explicitly assumes isotropic noise[3].

Here we propose to relax the assumption of isotropic noise by applying factor analysis (FA) (Everitt, 1984) instead of PCA / PPCA. The only difference between FA and PPCA is that FA allows for each neuron to have a different noise variance that is learned from the data. Fig. 5.4*A* shows that the direction recovered by FA (green line) is much closer to the true relationship (black line) than that recovered by PCA / PPCA (red line). The reason is that FA does not simply seek directions of greatest variance; rather, it seeks directions of greatest *covariance* while allowing for different noise variances along the different observed dimensions.

Although FA better estimates the true firing rate relationship compared to PCA / PPCA, there remains a problem that is common to all three techniques. In Fig. 5.4*A*, the underlying firing rates (black bar) of the two neurons lie within a relatively small range of 10 spikes per second. However, neurons can change their firing rate by many tens of spikes per second, for example, in response to a stimulus or during movement preparation and execution. As described above, the noise variance can therefore also change drastically over time for a given neuron. This is problematic because PCA, PPCA, and FA all assume that the noise variance of each neuron is fixed over time, regardless of how the neuron's underlying firing rate fluctuates. A possible solution is to replace the Gaussian observation noise model of PPCA and FA with a point-process (Smith and Brown, 2003; Truccolo et al., 2005; Yu et al., 2006) likelihood model. Such an extension is challenging due to issues of mathematical tractability and computational complexity (Cunningham et al., 2008c). In this work, we consider a simpler approach based on discrete time steps. The square-root transform is known to stabilize the variance of Poisson-distributed counts (Kihlberg et al., 1972). By stabilizing

[3]PCA is the limiting case of PPCA as the noise variance goes to zero.

the noise variance, dimensionality-reduction techniques that assume stationary noise variance (such as PCA, PPCA, and FA) can then be applied. Because the square-root transform only operates on count data, we propose performing the following sequence of preprocessing operations in lieu of kernel-smoothing the spike trains directly: (i) spike counts are taken in non-overlapping time bins, (ii) the counts are square-root tranformed[4], and (iii) the transformed counts are kernel-smoothed over time. The resulting datapoints are then passed to PCA, PPCA, or FA.

If the spike counts were indeed Poisson-distributed and if the square-root transform were able to perfectly stabilize the variance of Poisson-distributed counts, then the use of PCA / PPCA would be justified, since the spiking noise (in the space of smoothed, square-rooted counts) would be isotropic across different neurons and timepoints. However, spike counts of real neurons are known to deviate from a Poisson distribution (e.g., Tolhurst et al., 1983; Churchland et al., 2006b), and the square-root transform only approximately stabilizes the variance of Poisson-distributed counts (Kihlberg et al., 1972). This is the case both across neurons and across timepoints. To compensate for unequal variances across neurons, we apply FA rather than PCA / PPCA. In *Results*, we show that two-stage methods based on FA outperform those based on PCA / PPCA.

5.2.2 Leave-neuron-out Prediction Error

There are several modeling choices to be made when extracting neural trajectories. First, for the two-stage methods involving kernel smoothing, we would like to find the appropriate degree of smoothness by comparing different smoothing kernel widths. Second, for the two-stage methods, we would like to compare different dimensionality reduction techniques (in this work, PCA, PPCA, and FA). Third, we would like to compare the two-stage methods with GPFA. Fourth, for all two-stage methods and GPFA, we would like to compare different dimensionalities of the low-dimensional state space. Such a comparison would help to determine whether the high-dimensional recorded activity can indeed be succinctly summarized by a low-dimensional neural

[4]Our datasets include multi-unit activity, comprising the activity of single neurons whose spike waveforms could not be discriminated through spike sorting. We consider a multi-unit spike count to be the summed spike counts of its constituent single neurons. Because the sum of Poisson random variables is Poisson, we apply the square-root transform to both single- and multi-units in our datasets.

trajectory, and help to select the appropriate dimensionality of the low-dimensional space.

Such modeling choices are typically made by comparing cross-validated prediction errors (Hastie et al., 2001) or likelihoods, or by comparing Bayesian marginal likelihoods (MacKay, 2003), which are often approximated using the Akaike information criterion (AIC) or the Bayesian information criterion (BIC). There are two reasons why the likelihood approaches are not applicable here. First, most of the two-stage methods are partially or entirely non-probabilistic. In particular, kernel smoothing and PCA are non-probabilistic operations. Second, even if a probabilistic dimensionality reduction technique (e.g., PPCA or FA) is used, the likelihoods obtained are based on the smoothed data. When the data are altered by different pre-smoothing operations (or not, in the case of GPFA), the likelihoods are no longer comparable. Here, we introduce a goodness-of-fit metric by which all of the comparisons listed above can be made.

We describe the basic idea of the metric in this section; the mathematical details are given in Appendix D. First, we select a particular method for extracting neural trajectories for which we want to evaluate goodness-of-fit. For the two-stage methods using kernel smoothing, this involves specifying the smoothing kernel width and the dimensionality reduction technique (e.g., PCA, PPCA, or FA) to be used. Next, the model parameters are fit to the training data. For example, for the PCA-based two-stage method, the model parameters are the principal directions and data mean found by applying PCA to the smoothed square-rooted spike counts. Then, based on data not used for model fitting, we leave out one neuron at a time and ask how well the fitted model is able to predict the activity of that neuron, given the activity of all other recorded neurons.

This leave-neuron-out model prediction is illustrated in Fig. 5.4*B*. Consider the same situation with two neurons as in Fig. 5.4*A*. Here, we leave out neuron 1 and ask each dimensionality reduction technique (PCA, PPCA, FA) to predict the activity of neuron 1 based only on the activity of neuron 2 (blue dot). For PCA, this is a simple geometric projection, yielding the red dot labeled 'PCA'. Although PPCA finds the same principal direction as PCA (as shown in Fig. 5.4*A*), it yields a different model prediction (red dot labeled 'PPCA'). The reason is that PPCA has an explicit noise model, which allows deviations of neuron 2's activity away from its mean to be

attributed partially to noise, rather than entirely to changes in the low-dimensional state (i.e., movement along the red line). Thus, the PPCA model prediction is more robust to observation noise than the PCA model prediction. One can use the PPCA intuition to understand the FA model prediction (green dot). The only difference is that FA allows different neurons to have different noise variances. Although PPCA and FA are shown to give nearly identical model predictions in the two-neuron case in Fig. 5.4*B*, their model predictions are generally different for larger numbers of neurons (cf. Fig. 5.5*A*). The same ideas can be applied to compute the model prediction for GPFA, which incorporates the concept of time.

For all methods considered in this work, the model prediction can be computed analytically because all variables involved are jointly Gaussian, as detailed in Appendix D. We compute a prediction error, defined as the sum-of-squared differences between the model prediction and the observed square-rooted spike counts across all neurons and timepoints. This prediction error can be computed for the two stage methods (using various smoothing kernel widths and dimensionality reduction techniques) and GPFA, across different choices of the state space dimensionality. The comparisons listed at the beginning of this section can then be made by comparing the prediction errors.

5.3 Gaussian-process Factor Analysis

In this section, we provide motivation for GPFA before describing it mathematically. Then, we detail how to fit the GPFA model to neural data. Finally, we show how the extracted trajectories can be intuitively visualized using an orthonormalization procedure, and describe how this gives rise to a "reduced" GPFA model with fewer state dimensions than timescales.

Motivation for GPFA. PCA, PPCA, and FA are all static dimensionality reduction techniques. In other words, none of them take into account time labels when applied to time series data; the measurements are simply treated as a collection of datapoints. In the two-stage methods, the temporal relationships among the datapoints are taken into account during kernel smoothing. There is then no explicit use of time label information during dimensionality reduction.

Here we propose an extension of FA that performs smoothing and dimensionality reduction in a common probabilistic framework, which we term *Gaussian-process factor analysis* (GPFA). Unlike FA, GPFA leverages the time label information to provide more powerful dimensionality reduction for time series data. The GPFA model is simply a set of factor analyzers (one per timepoint, each with identical parameters) that are linked together in the low-dimensional state space by a Gaussian process (GP) (Rasmussen and Williams, 2006) prior. Introducing the GP allows for the specification of a correlation structure across the low-dimensional states at different timepoints. For example, if the system underlying the time series data is believed to evolve smoothly over time, we can specify that the system's state should be more similar between nearby timepoints than between faraway timepoints. Extracting a smooth low-dimensional neural trajectory can therefore be viewed as a compromise between the low-dimensional projection of each datapoint found by FA and the desire to string them together using a smooth function over time.

Mathematical Description of GPFA. As with the two-stage methods, spike counts are first taken in non-overlapping time bins and square-rooted. However, unlike the two-stage methods, there is no pre-smoothing of the square-rooted spike counts for GPFA, since the smoothing and dimensionality reduction are performed together. Let $\mathbf{y}_{:,t} \in \mathbb{R}^{q \times 1}$ be the high-dimensional vector of square-rooted spike counts recorded at timepoint $t = 1, \ldots, T$, where q is the number of neurons being recorded simultaneously. We seek to extract a corresponding low-dimensional latent *neural state* $\mathbf{x}_{:,t} \in \mathbb{R}^{p \times 1}$ at each timepoint, where p is the dimensionality of the state space ($p < q$). For notational convenience[5], we group the neural states from all timepoints into a *neural trajectory* denoted by the matrix $X = [\mathbf{x}_{:,1}, \ldots, \mathbf{x}_{:,T}] \in \mathbb{R}^{p \times T}$. Similarly, the observations can be grouped into a matrix $Y = [\mathbf{y}_{:,1}, \ldots, \mathbf{y}_{:,T}] \in \mathbb{R}^{q \times T}$. We define a linear-Gaussian relationship between the observations $\mathbf{y}_{:,t}$ and neural states $\mathbf{x}_{:,t}$

$$\mathbf{y}_{:,t} \mid \mathbf{x}_{:,t} \sim \mathcal{N}\left(C\mathbf{x}_{:,t} + \mathbf{d}, R\right), \tag{5.1}$$

where $C \in \mathbb{R}^{q \times p}$, $\mathbf{d} \in \mathbb{R}^{q \times 1}$, and $R \in \mathbb{R}^{q \times q}$ are model parameters to be learned. As in FA, we constrain the covariance matrix R to be diagonal, where the diagonal elements

[5] A colon in the subscript denotes all elements in a particular row or column. For example, $\mathbf{x}_{:,t}$ specifies all elements in the tth column of X, whereas $\mathbf{x}_{i,:}$ specifies all elements in the ith row of X.

are the independent noise variances of each neuron. In general, different neurons can have different independent noise variances. Although a Gaussian is not strictly a distribution on square-rooted counts, its use in Eq. 5.1 preserves computational tractability (e.g., Wu et al., 2006).

The neural states $\mathbf{x}_{:,t}$ at different timepoints are related through Gaussian processes, which embody the notion that the neural trajectories should be smooth. We define a separate GP for each dimension of the state space indexed by $i = 1, \ldots, p$

$$\mathbf{x}_{i,:} \sim \mathcal{N}(\mathbf{0}, K_i), \tag{5.2}$$

where $\mathbf{x}_{i,:} \in \mathbb{R}^{1 \times T}$ is the ith row of X and $K_i \in \mathbb{R}^{T \times T}$ is the covariance matrix for the ith GP. The form of the GP covariance can be chosen to provide different smoothing properties on the neural trajectories. In this work, we chose the commonly-used squared exponential (SE) covariance function

$$K_i(t_1, t_2) = \sigma_{f,i}^2 \cdot \exp\left(-\frac{(t_1 - t_2)^2}{2 \cdot \tau_i^2}\right) + \sigma_{n,i}^2 \cdot \delta_{t_1,t_2}, \tag{5.3}$$

where $K_i(t_1, t_2)$ denotes the (t_1, t_2)th entry of K_i and $t_1, t_2 = 1, \ldots, T$. The SE covariance is defined by its signal variance $\sigma_{f,i}^2 \in \mathbb{R}_+$, characteristic timescale $\tau_i \in \mathbb{R}_+$, and GP noise variance $\sigma_{n,i}^2 \in \mathbb{R}_+$. The Kroneker delta δ_{t_1,t_2} equals 1 if $t_1 = t_2$ and 0 otherwise. The SE is an example of a stationary covariance; other stationary and non-stationary GP covariances (Rasmussen and Williams, 2006) can be applied in a seamless way.

Because the neural trajectories X are hidden and must be inferred from the recorded activity Y, the scale of X (defined by the K_i in Eq. 5.2) is arbitrary. In other words, any scaling of X can be compensated by appropriately scaling C (which maps the neural trajectory into the space of recorded activity) such that the scale of Y remains unchanged[6]. To remove this model redundancy without changing the expressive power of the model, we fix the scale of X and allow C to be learned without constraints. By direct analogy to FA, we set the prior distribution of the neural state $\mathbf{x}_{:,t}$ at each timepoint t to be $\mathcal{N}(\mathbf{0}, I)$ by fixing $K_i(t, t) = 1$ (however, note that the $\mathbf{x}_{:,t}$

[6]This can be seen mathematically in Eq. D.7, where $\bar{K}$ defines the scale of X. The scale of Y depends on the product $\bar{C}\bar{K}\bar{C}'$, not on $\bar{K}$ and $\bar{C}$ individually. Thus, any scaling on $\bar{K}$ can be compensated by appropriately scaling $\bar{C}$.

are still correlated across different t). This can be achieved by setting $\sigma^2_{f,i} = 1 - \sigma^2_{n,i}$, where $0 < \sigma^2_{n,i} \leq 1$[7]. Because we seek to extract smooth neural trajectories, we fixed $\sigma^2_{n,i}$ to a small value (10^{-3}), as is often done for GPs (Rasmussen and Williams, 2006). In Appendix D, we consider learning $\sigma^2_{n,i}$ from the data. For all analyses described in *Results*, the timescale τ_i is the only parameter of the SE covariance that is learned.

Fitting the GPFA Model. The parameters of the GPFA model (Eqs. 5.1 and 5.2) can be fit using the commonly-used expectation-maximization (EM) algorithm (Dempster et al., 1977). The EM algorithm seeks to fit the model parameters $\theta = \{C, \mathbf{d}, R, \tau_1, \ldots, \tau_p\}$ to maximize the probability of the observed data Y. In Appendix D, we derive the EM update equations for the GPFA model. Because the neural trajectories and model parameters are both unknown, the EM algorithm iteratively updates the neural trajectories (in the E-step) and the model parameters (in the M-step) while the other remains fixed. This algorithm is guaranteed to converge to a local optimum. The E-step involves using the most recent parameter updates to evaluate the relative probabilites of all possible neural trajectories given the observed spikes. This Gaussian posterior distribution $P(X \mid Y)$ can be computed exactly because the $\mathbf{x}_{:,t}$ and $\mathbf{y}_{:,t}$ across all timepoints are jointly Gaussian, by definition. In the M-step, the model parameters are updated using the distribution $P(X \mid Y)$ over neural trajectories found in the E-step. The updates for C, $\mathbf{d}$, and R can be expressed in closed form and are analogous to the parameter updates in FA. The characteristic timescales τ_i can be updated using any gradient optimization technique. Note that the degree of smoothness (defined by the timescales) and the relationship between the low-dimensional neural trajectory and the high-dimensional recorded activity (defined by C) are jointly optimized. Furthermore, a different timescale is learned for each state dimension indexed by i.

Visualizing Trajectories via Orthonormalization. Once the GPFA model is learned, we can use it to extract neural trajectories $E[X \mid Y]$ (Eq. D.6) from the observed activity Y. These low-dimensional neural trajectories can be related to the high-dimensional observed activity using Eq. 5.1, which defines a linear mapping C between the two spaces. The following is one way to understand this mapping. Each

[7]The GP noise variance $\sigma^2_{n,i}$ must be non-zero to ensure that K_i is invertible. If $\sigma^2_{n,i} = 1$, there is no correlation across time and GPFA becomes FA.

column of C defines an axis in the high-dimensional space. The ith element of $\mathbf{x}_{:,t}$ ($i = 1, \ldots, p$) specifies "how far to go" along the axis defined by the ith column of C. The location in the high-dimensional space corresponding to the neural state $\mathbf{x}_{:,t}$ is given by the summed contributions along each of the p aforementioned axes, plus a constant offset $\mathbf{d}$.

While the relationship between the low- and high-dimensional spaces is mathematically well-defined, it is difficult to picture this relationship without knowing the direction and scale of the axes defined by the columns of C. For example, any point in two-dimensional space can be represented as a linear combination of arbitrary two-dimensional vectors $\mathbf{w}_1$ and $\mathbf{w}_2$, provided that the two vectors are not scaled versions of each other. If $\mathbf{w}_1$ and $\mathbf{w}_2$ are neither orthogonal nor of unit length, understanding how the two vectors are linearly combined to form different points can be non-intuitive. Hence, points in two-dimensional space are typically referred to using their (x, y) Cartesian coordinates. These coordinates specify how unit vectors pointing along the x and y axes (i.e., orthonormal vectors) should be linearly combined to obtain different points in the two-dimensional space. This provides an intuitive specification of points in the two-dimensional space. In GPFA, the columns of C are not orthonormal, akin to the arbitrary $\mathbf{w}_1$ and $\mathbf{w}_2$. The following paragraphs describe how to orthonormalize the columns of C in GPFA to make visualization more intuitive. This is akin to expressing two-dimensional points in terms of their (x, y) Cartesian coordinates.

In the case of PCA, the identified principal directions are orthonormal, by definition. It is this orthonormal property that yields the intuitive low-dimensional visualization – i.e., the intuitive mapping between the low-dimensional principal components space and high-dimensional data space. Although the columns of C are not constrained to be orthonormal for GPFA, we can still obtain an intuitive "PCA-like" mapping between the two spaces for ease of visualization. The basic idea is to find a set of orthonormal basis vectors spanning the same space as the columns of C. This is akin to finding the unit vectors that point along the two Cartesian axes from the arbitrary $\mathbf{w}_1$ and $\mathbf{w}_2$, in the example above. This orthonormalization procedure does not alter the GPFA model-fitting procedure, nor the extracted neural trajectories; it simply offers a more intuitive way of visualizing the extracted trajectories.

The orthornormalization procedure involves applying the singular value decomposition (Strang, 1988) to the learned C. This yields $C = UDV'$, where $U \in \mathbb{R}^{q \times p}$ and $V \in \mathbb{R}^{p \times p}$ each have orthonormal columns and $D \in \mathbb{R}^{p \times p}$ is diagonal. Thus, we can write $C\mathbf{x}_{:,t} = U\left(DV'\mathbf{x}_{:,t}\right) = U\tilde{\mathbf{x}}_{:,t}$, where $\tilde{\mathbf{x}}_{:,t} = DV'\mathbf{x}_{:,t} \in \mathbb{R}^{p \times 1}$ is the *orthonormalized neural state* at timepoint t. Note that $\tilde{\mathbf{x}}_{:,t}$ is a linear transformation of $\mathbf{x}_{:,t}$. The *orthonormalized neural trajectory* extracted from the observed activity Y is, therefore, $DV'\ E[X \mid Y]$. Since U has orthonormal columns, we can now intuitively visualize the trajectories extracted by GPFA, in much the same spirit as for PCA.

There is one other important advantage of the orthonormalization procedure. Whereas the elements of $\mathbf{x}_{:,t}$ (and the corresponding columns of C) have no particular order, the elements of $\tilde{\mathbf{x}}_{:,t}$ (and the corresponding columns of U) are ordered by the amount of data covariance explained, analogous to PCA. Especially when the number of state dimensions p is large, the ordering facilitates the identification and visualization of the dimensions of the orthonormalized neural trajectory that are most important for explaining the recorded activity. The ordering is made possible by the singular value decomposition, which specifies the scaling of each of the columns of U in the diagonal entries of D (i.e., the singular values). If these diagonal entries are arranged in decreasing order, then the columns of U specify directions in the high-dimensional space in order of decreasing data covariance explained. Overall, the orthonormalization procedure allows us to view the neural trajectories extracted by GPFA using PCA-like intuition[8]. In particular, the low-dimensional axes are ordered and can be easily pictured in the high-dimensional space. These concepts are illustrated in Fig. 5.2*C* and *D*, where S_1 and S_2 correspond to the first two dimensions of the orthonormalized neural state $\tilde{\mathbf{x}}_{:,t}$.

Reduced GPFA. According to Eq. 5.2, each neural state dimension indexed by i has its own characteristic timescale τ_i. This implies that a GPFA model with a p-dimensional neural state possesses a total of p timescales. However, there may be cases where the number of timescales needed to describe the data exceeds the number of state dimensions. For example, it may be that a system only utilizes two degrees

[8]This orthonormalization procedure is also applicable to PPCA and FA. In fact, it is through this orthnormalization procedure that the principal directions found by PPCA are equated with those found by PCA. In general, the solutions found by PPCA and FA are unique up to an arbitrary rotation of the low-dimensional space. The orthonormalization procedure resolves this ambiguity.

of freedom (i.e., two state dimensions), but evolves over time with a wide range of different speeds that cannot be well-captured using only two timescales. Here, we describe a way to obtain a GPFA model whose number of timescales p exceeds the effective state dimensionality $\tilde{p}$. First, a GPFA model with state dimensionality p is fit using the EM algorithm. Next, the orthonormalization procedure described above is applied, yielding the orthonormalized neural state $\tilde{\mathbf{x}}_{:,t} \in \mathbb{R}^{p \times 1}$. Note that, while each dimension of $\mathbf{x}_{:,t}$ possesses a single characteristic timescale, each dimension of $\tilde{\mathbf{x}}_{:,t}$ represents a mixture of p timescales. Because the dimensions of $\tilde{\mathbf{x}}_{:,t}$ are ordered by the amount of data covariance explained, we can choose to retain only the top $\tilde{p}$ dimensions of $\tilde{\mathbf{x}}_{:,t}$ ($\tilde{p} = 1, \ldots, p$) and to discard the remaining lowest dimensions. This yields a $\tilde{p}$-dimensional neural trajectory for the *reduced GPFA* model.

5.3.1 Dynamical Systems Approaches

Another way to extract neural trajectories is by defining a parametric dynamical model that describes how the low-dimensional neural state evolves over time. A hidden Markov model (HMM) is a dynamical model in which the state jumps among a set of discrete values. HMMs have been applied fruitfully to study single-trial neural population activity in monkey frontal cortex (Seidemann et al., 1996; Abeles et al., 1995; Gat et al., 1997), rat gustatory cortex (Jones et al., 2007), monkey premotor cortex (Kemere et al., 2008), and songbird premotor areas (Danóczy and Hahnloser, 2006; Weber and Hahnloser, 2007). In many experimental contexts, it is desirable to allow for a continuous-valued state, rather than one that jumps among a set of discrete values. Even in settings where the experimental paradigm defines discrete states (e.g., Fig. 5.1*A*, one state per percept or decision), there are advantages to using continuous-valued states. Whereas a HMM would indicate *when* the switches occur in Fig. 5.1*A*, a dynamical model with continuous-valued states would allow one to study the details of *how* the switching is carried out; in particular, along what path and how quickly. While it is always possible to define a HMM with a larger number of discrete states to approximate a model with continuous-valued states, such an approach is prone to overfitting and requires appropriate regularization (Beal et al., 2002).

A commonly-used dynamical model with continuous-valued state is a first-order linear auto-regressive (AR) model (Smith and Brown, 2003; Kulkarni and Paninski,

2007), which captures linear Markovian dynamics. Such a model can be expressed as a Gaussian process, since the state variables are jointly Gaussian. This can be shown by defining a separate first-order AR model for each state dimension indexed by $i \in \{1, \ldots, p\}$

$$x_{i,t+1} \mid x_{i,t} \sim \mathcal{N}\left(a_i x_{i,t}, \sigma_i^2\right). \tag{5.4}$$

Given enough time ($t \to \infty$) and $|a_i| < 1$, the model will settle into a stationary state that is equivalent to Eq. 5.2 with

$$K_i(t_1, t_2) = \frac{\sigma_i^2}{1 - a_i^2} a_i^{|t_1 - t_2|}, \tag{5.5}$$

as derived elsewhere (Turner and Sahani, 2007). The first-order AR model described by Eqs. 5.2 and 5.5, coupled with the linear-Gaussian observation model Eq. 5.1, will be henceforth be referred to as "LDS" (linear dynamical system). Different covariance structures K_i can be obtained by going from a first-order to an nth-order AR model. One drawback of this approach is that it is usually not easy to construct an nth-order AR model with a specified covariance structure. In contrast, the GP approach requires only the specification of the covariance structure, thus allowing different smoothing properties to be applied in a seamless way. AR models are generally less computationally demanding than those based on GP, but this advantage shrinks as the order of the AR model grows. Another difference is that Eq. 5.5 does not contain an independent noise term $\sigma_{n,i}^2 \cdot \delta_{t_1,t_2}$ as in Eq. 5.3. The innovations noise σ_i^2 in Eq. 5.4 is involved in setting the smoothness of the time series, as shown in Eq. 5.5. Thus, Eq. 5.4 would need to be augmented to explicitly capture departures from the AR model.

One may also consider defining a non-linear dynamical model (Yu et al., 2006), which typically has a richer set of dynamical behaviors than linear models. The identification of the model parameters provides insight into the dynamical rules governing the time-evolution of the system under study. However, especially in exploratory data analyses, it may be unclear what form this model should take. Even if an appropriate non-linear model can be identified, using it to extract neural trajectories may require computationally-intensive approximations and yield unstable model-fitting algorithms (Yu et al., 2006). In contrast, the model-fitting algorithm for GPFA is

stable, approximation-free, and straightforward to implement. The use of GPFA can be viewed as a practical way of going beyond a first-order linear AR model without having to commit to a particular non-linear system, while retaining computational tractability.

Relative to previously proposed models in the machine learning literature, GPFA is most similar to the semiparametric latent factor model (Teh et al., 2005), where the GPFA model can be obtained by letting time indices play the role of inputs. Although GPFA involves Gaussian processes and latent variables, it is quite different from the Gaussian process latent variable model (GP-LVM) (Lawrence, 2005). The GP-LVM uses Gaussian processes to define a non-linear relationship between the latent and the observed variables. In that case, the GP smoothing is defined by how close two points are in the latent space. In contrast, GPFA defines a *linear* mapping between the latent and observed variables; the GP smoothing is defined by how close two points are in *time* (Lawrence and Moore, 2007). GPFA is also quite different from Gaussian process dynamical models (GPDM) (Wang et al., 2006). Whereas GPDM extends Markovian linear AR models to the non-linear regime (while remaining Markovian), GPFA extends to the non-Markovian regime (while remaining linear) with arbitrary temporal covariance structures. As with GP-LVM, GPDM defines a non-linear relationship between the latent and observed variables.

5.3.2 Delayed-reach Task and Neural Recordings

Animal protocols were approved by the Stanford University Institutional Animal Care and Use Committee. We trained an adult male monkey (*Macaca mulatta*, monkey G) to perform delayed center-out reaches for juice rewards. Visual targets were back-projected onto a fronto-parallel screen ∼30 cm in front of the monkey. The monkey touched a central target and fixated his eyes on a crosshair at the upper right corner of the central target. After a center hold period of 1000 ms, a pseudo-randomly-chosen peripheral reach target was presented at one of 14 possible locations (directions: 0, 45, 90, 135, 180, 225, 315°; distances: 60, 100 mm)[9]. After a randomly-chosen instructed delay period, the "go" cue (signaled by both the enlargement of the reach target and the disappearance of the central target) was given and the monkey reached to the

[9]Reach targets were not presented directly below the central target (i.e., direction: 270°) since they would be occluded by the monkey's hand while he is touching the central target.

target. In the present work, we analyzed data from two experiments which differ only in the distribution of delay periods used. Whereas experiment `G20040123` used delay periods in the range 200–700 ms, experiment `G20040124` used three discrete delay periods of 30, 130, and 230 ms. Eye fixation at the crosshair was enforced throughout the delay period. After a hold time of 200 ms at the reach target, the monkey received a liquid reward.

During experiments, monkeys sat in a custom chair (Crist Instruments, Hagerstown, MD) with the head braced and the non-reaching arm strapped to the chair. The presentation of the visual targets was controlled using the Tempo software package (Reflective Computing, St. Louis, MO). A custom photo-detector recorded the timing of the video frames with 1 ms resolution. The position of the hand was measured in three dimensions using the Polaris optical tracking system (Northern Digital, Waterloo, Ontario, Canada; 60 Hz, 0.35 mm accuracy), whereby a passive marker taped to the monkey's fingertip reflected infrared light back to the position sensor. Eye position was tracked using an overhead infrared camera (Iscan, Burlington, MA; 240 Hz, estimated accuracy of 1°).

A 96-channel silicon electrode array (Cyberkinetics, Foxborough, MA) was implanted straddling dorsal premotor (PMd) and motor (M1) cortex in the right hemisphere, contralateral to the reaching arm. Surgical procedures have been described previously (Churchland et al., 2006b). An intra-operative photo showing the exact location of array implantation can be found in (Batista et al., 2007). We manually discriminated spike waveforms at the start of each session using two time-amplitude window discriminators on each channel. Isolations were tagged as either single-unit or multi-unit based on visual inspection of their quality during the experiment. On this particular electrode array, we found several groups of electrodes that yielded nearly identical (or highly similar) spike trains. While the source of this electrode "crosstalk" is currently unclear, we speculate that it may be due to faulty electrical isolation among the channels either in the pedestal connectorization, in the wire bundle leading out of the array, or in the array iself. It is unlikely that two adjacent electrodes recorded from the same neuron(s), given the distance between adjacent electrodes (400 μm). We have observed such crosstalk on a few different electrode arrays. For prosthetic decoding (e.g., of arm trajectories), this is typically not a major concern, since it simply gives the repeated unit(s) a greater influence on the

decoded result. For extracting neural trajectories, this *is* a major problem, since the goal is to identify structure in the correlated activity across the neural population. If two units have identical (or near identical) activity, one of the dimensions of the neural trajectory is likely to be dedicated to describing this spurious, strong correlation between the pair of units. Thus, before analyzing the data, we checked all pairs of the 96 electrodes for crosstalk by computing the percentage of coincident spikes (allowing for ± 1 ms jitter) for each pair. Across all pairs, this yielded a clear, bimodal distribution. On this electrode array, we found crosstalk in three electrode pairs, two triplets, and one quadruplet. We, therefore, removed 10 of the 96 channels before any of the analyses described in this paper were performed.

The analyses in the present work are concerned primarily with the neural activity during the delay period. However, many of our isolations showed the strongest modulation during the movement period and/or showed weakly-modulated delay-period activity. A single- or multi-unit was therefore only included in our analyses if (i) it possessed tuned ($p < 0.1$) delay-period activity with reasonable modulation (at least five spikes per second difference between the most and least responsive conditions), and (ii) the delay-period firing rate averaged across all conditions was at least 20% of the movement-period firing rate averaged across all conditions. For these assessments, the delay-period firing rate was computed in a 200 ms window starting 150 ms after reach target presentation, whereas the movement-period firing rate was computed in a 300 ms window starting 100 ms before movement onset.

In total, we analyzed 784 (910) trials for experiment `G20040123` (`G20040124`), comprising 18 (18) single-units and 43 (44) multi-units. The distribution of reaction times, defined as the time between the go cue and movement onset, had mean$\pm$SD of 293$\pm$48 ms (329$\pm$54 ms). The arm movement durations were 269$\pm$40 ms (280$\pm$44 ms). Both datasets have previously appeared (Churchland et al., 2006b). We have explicitly chosen to analyze the same datasets here to uncover the single-trial substrates of the trial-averaged effects reported in our previous studies.

5.4 Results

We considered neural data for one reach target at a time, ranging from 200 ms before reach target onset to movement end. This period comprised the randomly-chosen delay period following reach target onset, the monkey's reaction time, and the duration of the arm reach. Spike counts were taken in non-overlapping 20 ms bins, then square-rooted[10]. For the two-stage methods, these square-rooted counts were first smoothed over time using a Gaussian kernel before being passed to a static dimensionality reduction technique: PCA, PPCA, or FA. LDS and GPFA were given the square-rooted spike counts with no kernel pre-smoothing.

Using the goodness-of-fit metric described in *Methods*, we can compare different degrees of smoothness, dimensionality reduction techniques (PCA, PPCA, and FA), and state dimensionalities for the two-stage methods. Fig. 5.5*A* shows the prediction error for PCA (dashed red), PPCA (solid red), and FA (green) across different state dimensionalities (p =3, 5, 10, 15) with a kernel width of 50 ms. Alternatively, we can fix the state dimensionality ($p = 10$ in Fig. 5.5*B*, $p = 15$ in Fig. 5.5*C*) and vary the kernel width. There are two primary findings for the two-stage methods. First, PCA, PPCA, and FA yielded progressively lower prediction error (Wilcoxon paired-sample test, $P < 0.001$). Statistical significance was assessed by looking across the 14 reach targets; for each reach target, we obtained the prediction error for each method at its optimal state dimensionality and kernel width. FA outperforms PCA and PPCA because it allows different neurons to have different noise variances. Recall that prediction errors were evaluated based on data not used for model-fitting (i.e., cross-validated), so this result cannot simply be due to FA having more parameters. PCA has the worst performance because it has no explicit noise model and is, therefore, unable to distinguish between changes in the underlying neural state and spiking noise. Second, for these data, the optimal smoothing kernel width was approximately 40 ms for both PPCA and FA, as indicated by Fig. 5.5*B* and *C*.

The same metric can be used to compare the two-stage methods with LDS and GPFA. As indicated in Fig. 5.5, LDS (blue) yielded lower prediction error than the two-stage methods (Wilcoxon paired-sample test, $P < 0.001$). Furthermore, GPFA

[10] All major trends in Fig. 5.5 were preserved without the square-root transform. We also considered smoothing spike trains directly (i.e., without binning) for the two-stage methods, which yielded nearly identical results as smoothing (non-square-rooted) spike counts. In the present work, the spikes are binned because it allows the square-root transform to be used, as described in *Methods*.

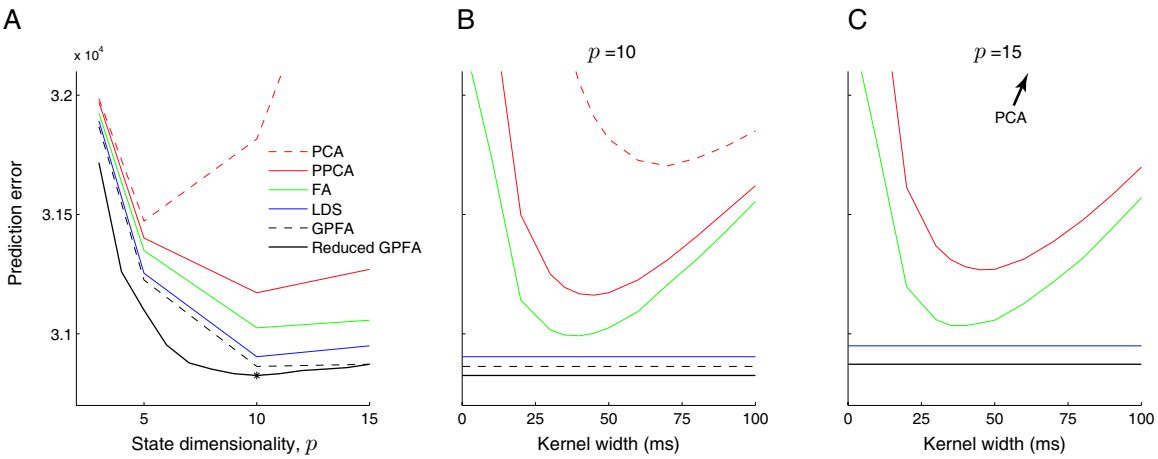


Figure 5.5: Prediction errors of two-stage methods (PCA, dotted red; PPCA, solid red; FA, green), LDS (blue), GPFA (dashed black), and reduced GPFA (solid black), computed using 4-fold cross-validation. *A*: Prediction errors for different state dimensionalities. For two-stage methods, prediction errors shown for 50 ms kernel width (s.d. of Gaussian kernel). For reduced GPFA, the horizontal axis corresponds to $\tilde{p}$ rather than p, where the prediction error is computed using only the top $\tilde{p}$ orthonormalized dimensions of a GPFA model fit with $p = 15$. Star indicates minimum of solid black curve. Denser sampling of kernel widths shown for *B*: $p = 10$ and *C*: $p = 15$. Note that the dashed and solid black lines are overlaid in *C*, by definition. Analyses in this figure are based on 56 trials and $q = 61$ units for the reach target at distance 100 mm and direction 45°. Experiment `G20040123`.

(dashed black) outperformed LDS and the two-stage methods (Wilcoxon paired-sample test, $P < 0.001$). As above, statistical significance was assessed by looking across the 14 reach targets; for each reach target, we obtained minimum prediction error for each method (LDS and GPFA) at its optimal state dimensionality. The prediction error was further reduced by taking only the top $\tilde{p}$ orthonormalized state dimensions of a GPFA model fit with $p = 15$ (reduced GPFA, solid black). Among the methods for extracting neural trajectories compared in this work[11], reduced GPFA produced the lowest prediction error (Wilcoxon paired-sample test, $P < 0.001$). Further insight regarding the performance of the reduced GPFA model is provided below.

Based only on Fig. 5.5, it is difficult to assess the benefit of GPFA relative to

[11]For comparison, one may also consider computing the prediction error using the trial-averaged neural responses from the training data. However, trial-averaging is only possible if the experimental timing is identical on different trials. For the data being analyzed here, across-trial averaging is not possible because different trials have different delay periods, reaction times, and arm movement durations.

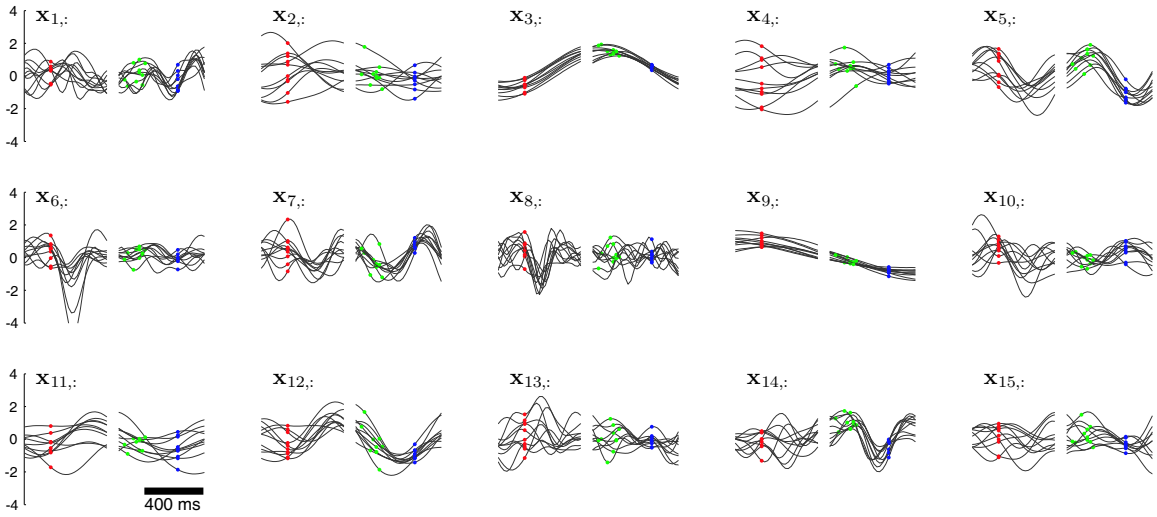


Figure 5.6: Neural trajectories for GPFA with $p = 15$. Each panel corresponds to one of the 15 dimensions of the neural state, which is plotted versus time. The neural trajectory for one trial comprises one black trace from each panel. Dots indicate time of reach target onset (red), go cue (green), and movement onset (blue). Due to differing trial lengths, the traces on the left/right half of each panel are aligned on target/movement onset for clarity. However, the GPFA model was fit using entire trials with no gaps. Note that the polarity of these traces is arbitrary, as long as it is consistent with the polarity of C. Each trajectory corresponds to planning and executing a reach to the target at distance 100 mm and direction 45°. For clarity, only 10 randomly-chosen trials with delay periods longer than 400 ms are plotted. Experiment `G20040123`, $q = 61$ units.

competing methods in terms of percent improvement in prediction error. The reason is that we don't know what the theoretical lower limit on the prediction error is for real neural data. It would be incorrect to compute the percent improvement in terms of distance from zero error. Thus, we performed a simulation (described in Appendix D and Fig. D.2) in which the error floor can be computed. Based on this error floor (which was far above zero), we found that GPFA provided tens of percent improvement in prediction error relative to the best two-stage method. This suggests that GPFA may have a similar percent improvement for the real neural data shown in Fig. 5.5.

Fig. 5.6 shows the neural trajectories $E[X \mid Y]$ (Eq. D.5) extracted by GPFA with $p = 15$. Each panel corresponds to a different neural state dimension, which evolves over time according to its own characteristic timescale τ_i that is learned from the data. For example, the timescales for the first five dimensions in Fig. 5.6 are

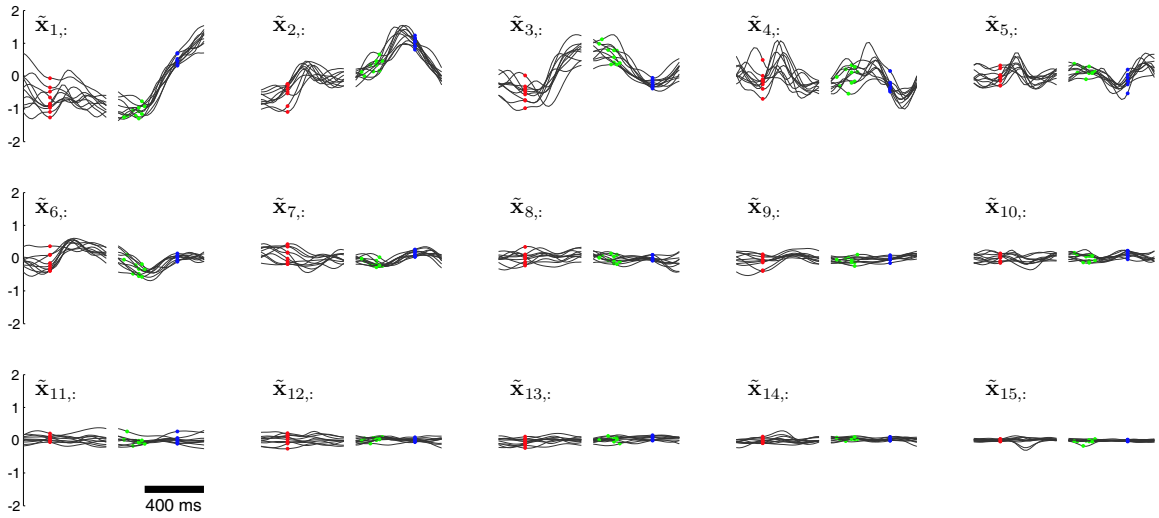


Figure 5.7: Orthonormalized neural trajectories for GPFA with $p = 15$. These are the same 10 trials shown in Fig. 5.6. Each panel corresponds to one of the 15 dimensions of the orthonormalized neural state, which is plotted versus time. The orthonormalized neural trajectory for one trial comprises one black trace from each panel. Note that the polarity of these traces is arbitrary, as long as it is consistent with the polarity of U. Figure conventions identical to those in Fig. 5.6.

54, 160, 293, 173, and 111 ms. Although we've obtained a substantial reduction in dimensionality in going from the 61-dimensional recorded neural responses to the 15-dimensional neural trajectories, it is still difficult to gain intuition about how the neural responses are evolving over time based solely on Fig. 5.6, for two reasons. First, the dimensions of the neural state are not ordered; thus, we don't know whether certain state dimensions are more important than others for explaining the activity across the neural population. Second, although each state dimension corresponds to a column of C, one cannot readily picture how the low-dimensional neural trajectories would appear if mapped out into the high-dimensional space using Eq. 5.1. The reason is that the columns of the learned C may have different scalings and are not guaranteed to be mutually orthogonal.

These difficulties can be overcome by applying the orthonormalization procedure described in *Methods* based on the singular value decomposition of C. The resulting orthonormalized neural trajectories are shown in Fig. 5.7, where each panel corresponds to an orthonormalized state dimension and involves a mixture of timescales. Importantly, the panels are arranged in decreasing order of data covariance explained. This ordering is apparent in Fig. 5.7 if one considers range of values explored by the

orthonormalized neural trajectory along each of its dimensions. The top orthonormalized dimensions indicate fluctuations in the recorded population activity shortly after target onset (red dots) and again after the go cue (green dots). Furthermore, the neural trajectories around the time of the arm movement are well-aligned on movement onset. These observations are consistent with previous analyses of the same data (Churchland et al., 2006b), as well as other studies of neural activity collected during similar tasks in the same cortical areas. Note that the neural trajectories in Fig. 5.7 are remarkably similar (but not identical) on different trials, even though (i) the spike timing differs across repeated trials, and (ii) there is no constraint built into GPFA requiring that neural trajectories should trace out similar paths on different trials. The orthonormalized dimensions $\tilde{\mathbf{x}}_{1,:}$ and $\tilde{\mathbf{x}}_{2,:}$ are analogous to S_1 and S_2, respectively, in Fig. 5.2. Unlike in Fig. 5.2*D* where the trajectory is plotted in the space of S_1 versus S_2, each orthonormalized dimension is plotted versus time in Fig. 5.7 to show more than just the top two (or three) dimensions.

The range of values explored by the trajectories in each orthonormalized dimension is analogous to the variance explained by each principal component in PCA. A common way to estimate the data dimensionality with PCA is to look for an "elbow" in the residual variance curve. Such an "elbow", if it exists, is typically considered to separate the signal dimensions from the noise dimensions. Similarly, we can obtain a rough estimate of the data dimensionality with GPFA by counting the number of top orthonormalized dimensions showing "meaningful" time-varying structure in Fig. 5.7. While the top six dimensions show strong temporal structure, it is unclear by eye whether the lower dimensions are needed to describe the population response. The number of "meaningful" dimensions can be rigorously quantified by computing the prediction error based only on the top $\tilde{p}$ orthonormalized dimensions (reduced GPFA), as described in *Methods*. In Fig. 5.5*A* (solid black), we found that the prediction error continued to decrease as more orthonormalized dimensions ($\tilde{\mathbf{x}}_{1,:}$, $\tilde{\mathbf{x}}_{2,:}$, ...) were included, up to $\tilde{\mathbf{x}}_{10,:}$. This indicates that dimensions $\tilde{\mathbf{x}}_{1,:}$ to $\tilde{\mathbf{x}}_{10,:}$ contain meaningful structure for explaining the population reponse. Beyond $\tilde{\mathbf{x}}_{10,:}$, adding additional dimensions increased the prediction error, indicating that the weak temporal structure seen in these lowest orthonormalized dimensions is primarily "noise". Thus, the solid black line reaches its minimum at $\tilde{p} = 10$ (referred to as p^*). By definition, the solid and dashed black lines coincide at $\tilde{p} = 15$.

Fig. 5.5*A* also shows that prediction error using only the top 10 orthonormalized dimensions (solid black, $\tilde{p} = 10$) is lower than that obtained by directly fitting a GPFA model with a 10-dimensional neural state (dashed black, $p = 10$). This can be understood by recalling that each panel in Fig. 5.7 represents a mixture of 15 characteristic timescales. Thus, the top 10 orthonormalized dimensions can make use of up to 15 timescales. In contrast, a GPFA model fit with $p = 10$ can have at most 10 timescales. By fitting a GPFA model with a large number of state dimensions p (each with its own timescale) and taking only the top $\tilde{p} = p^*$ orthonormalized dimensions, we can obtain neural trajectories whose effective dimensionality is smaller than the number of timescales at play.

Based on the solid black line in Fig. 5.5*A* we consider the effective dimensionality of the recorded population activity to be $p^* = 10$. In other words, the linear subspace within which the recorded activity evolved during reach planning and execution for this particular target was 10-dimensional. Across the 14 reach targets, each considered separately, the effective dimensionality ranged from 8 to 12, with a mode of 10. All major trends seen in Fig. 5.5 were preserved across all reach targets.

Having developed a method for extracting low-dimensional neural trajectories that yields lower prediction error than existing methods, we would like to apply it to study neural population activity on a trial-by-trial basis. We previously showed that the across-trial neural variability decreased during reach planning (Churchland et al., 2006b), which led to the conception that the underlying neural trajectories (indexing the process of motor planning) may be converging over time. However, this effect could only be inferred indirectly by collapsing over many neurons and trials. Using the methods described in the present work, we can now track the progress of motor planning on single trials and directly view their convergence over time. Fig. 5.8 shows neural trajectories plotted in the space of the top three orthonormalized state dimensions (corresponding to the first three panels of Fig. 5.7). The extent to which these trajectories converged during reach planning can be quantified by comparing the spread of neural states at target onset (red dots) to that at the go cue (green dots). These spreads are described by the covariance ellipsoids about the scatter of neural states at each of these timepoints, shown as shaded ellipses in Fig. 5.8. Formally, we computed the volume of the covariance ellipsoid, defined by the square root of the determinant of the covariance matrix. To compare the spreads at two

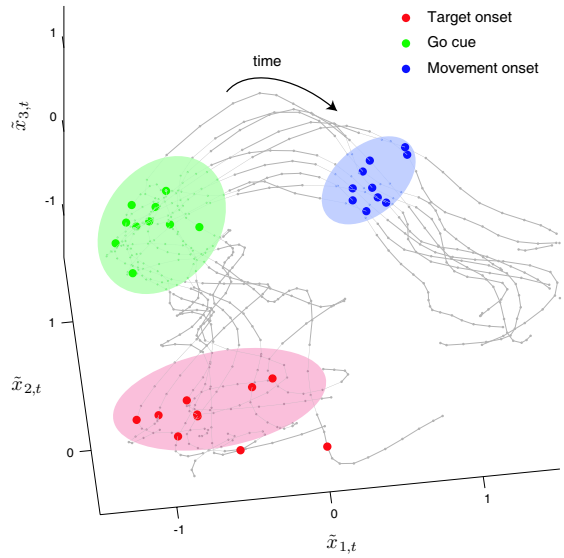


Figure 5.8: Top three dimensions of orthonormalized neural trajectories for GPFA with $p = 15$. Each gray trace corresponds to a single trial (same 10 trials as in Figs. 5.6 and 5.7). Small gray dots are timepoints separated by 20 ms. Larger dots indicate time of reach target onset (red), go cue (green), and movement onset (blue). Ellipses (two SD around mean) indicate the across-trial variability of neural state at reach target onset (red shading), go cue (green shading), and movement onset (blue shading). These ellipses can be obtained equivalently in two ways. One can either first project the neural states from the optimal 10-dimensional space into the three-dimensional space shown, then compute the covariance ellipsoids in the three-dimensional space; or, one can first compute the covariance ellipsoids in the 10-dimensional space, then project the ellipsoids into the three-dimensional space. The covariance ellipsoids were computed based on all 45 trials with delay periods longer than 400 ms for this reach target.

different timepoints, we took the ratio of volumes of the two covariance ellipsoids. We computed the ratio of volumes using the top p^* orthonormalized dimensions (in this case, 10), rather than just the top three orthonormalized dimensions shown in Fig. 5.8. It is essential to compute volumes (and perform other analyses) in the space of optimal dimensionality p^*, since important features of the trajectories can be lost by using only a subset of its dimensions. To compare this result across different reach targets which may have different p^*, we then took the p^*th root of this ratio to obtain a "ratio per dimension". Only trials with delay periods longer than 400 ms, for which there is enough time for the motor planning process to come to completion, were included in this analysis.

For the reach target considered in Figs. 5.7 and 5.8, the ratio per dimension from

target onset to the go cue was 1.34[12]. Across the 14 reach targets, the ratio per dimension was 1.31±0.13 (mean±SD). The mean of this distribution was larger than unity (one-sided t-test, $P < 0.001$), indicating that the neural state converged during reach planning. From the go cue (green dots) to movement onset (blue dots), the neural state further converged (one-sided t-test, $P < 0.001$), a finding that is also consistent with (Churchland et al., 2006b). In this case, the ratio per dimension was 1.17±0.14 (mean±SD) across the 14 reach targets. Since the columns of U are orthonormal, the same volumes can be obtained by first mapping the neural trajectories into the high-dimensional space using U (yielding denoised high-dimensional data) and computing the volumes there. Because firing rates and the associated spiking noise variances tend to rise after target onset (Churchland et al., 2006b), the spread of raw spike counts (with no smoothing or dimensionality reduction) in the high-dimensional space at the time of the go cue would be larger than that at the time of target onset.

Previous reports have shown that reaction times tend to be shorter on trials with longer delay periods, suggesting that some time-consuming motor preparatory process is given a head-start during the delay (Riehle and Requin, 1989; Crammond and Kalaska, 2000; Churchland et al., 2006b). In these studies, evidence is provided by the trial-averaged response of single neurons, or a one-dimensional timecourse (e.g., the average firing rate or the Fano factor) collapsed across the neural population . The methods presented in this work allow us view such effects in a multi-dimensional neural state space on single trials. We applied GPFA to a dataset with three discrete delay periods of 30, 130, and 230 ms. With these short delay periods, we can visualize the effect of the go cue arriving at different times during the early stages of motor preparation. The GPFA model with $p = 15$ was fit to trials of all delays periods together. Fig. 5.9 shows the extracted orthonormalized neural trajectories with trials grouped by delay period. Recall that the orthonormalized dimensions are ordered; within each row, the panels are arranged in decreasing order of data covariance explained. The panels in the first column ($\tilde{\mathbf{x}}_{1,:}$) appear to be largely capturing the movement-related neural activity (the ramp to the right of the green dots). The panels in the second column ($\tilde{\mathbf{x}}_{2,:}$, left dotted box) suggest that, prior to movement

[12]If computed improperly using only the top three orthonormalized dimensions rather than the top p^* orthonormalized dimensions, the ratio per dimension would be 1.01. In other words, there is nearly no decrease in volume between the spread of the red dots and that of the green dots in the top three orthonormalized dimensions shown in Fig. 5.8.

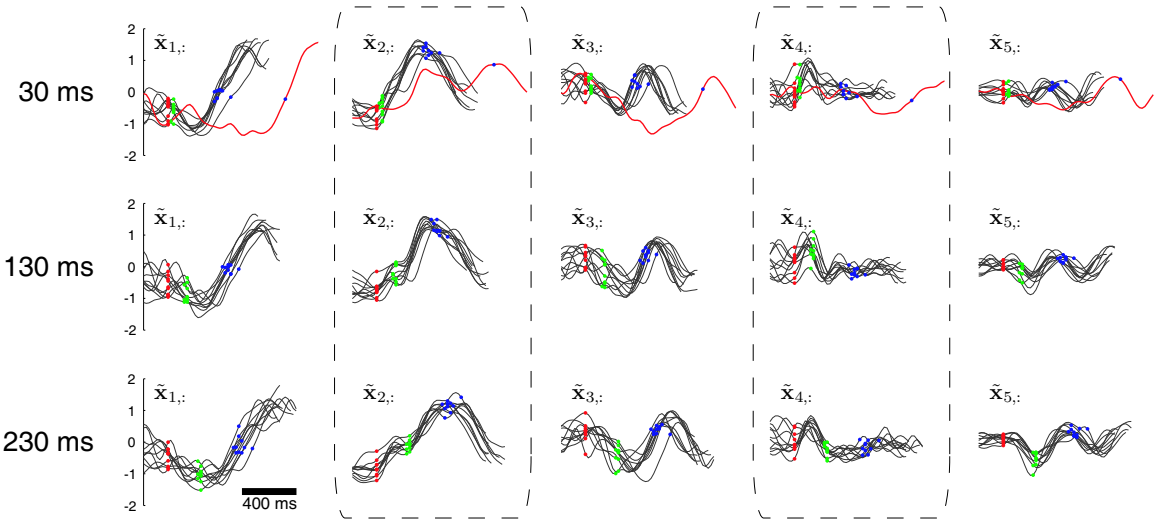


Figure 5.9: Orthonormalized neural trajectories for trials with discrete delay periods of 30 ms (top row), 130 ms (middle row), and 230 ms (bottom row). The red traces in the top row correspond to a single trial with an outlying reaction time (reaction time: 844 ms, trial ID 68). The top five orthonormalized dimensions of a GPFA model fit with $p = 15$ are shown for each delay period; the remaining orthonormalized dimensions are qualitatively similar to dimensions 6 to 15 in Fig. 5.7. Dotted boxes highlight the second and fourth orthonormalized dimensions, which are referred to in *Results*. For clarity, only ten randomly-chosen trials of each delay period are plotted, aligned on target onset. All trials shown correspond to the reach target located at distance 100 mm and direction 45°. Figure conventions are otherwise identical to those in Fig. 5.6. There is a small amount of temporal jitter in the green points due to the refresh rate of the visual display projector. Experiment `G20040124`, $q = 62$ units.

onset, the orthonormalized neural state must move from a baseline state (red dots) to a state appropriate for movement (blue dots) along this dimension. With a 30 ms delay period, nearly the entire traversal from baseline state to movement state occurs after the go cue (green dots). In contrast, with a 230 ms delay, the neural state performs part of the traversal during the delay period and appears to hold while waiting for the go cue. When the go cue arrives, the remainder of the traversal is carried out. If there is a limit on how quickly firing rates (and therefore the neural state) can change over time, then one would expect the reaction times (i.e., the time between the green and blue dots) to be longer for the 30 ms delays than the 230 ms delays. Indeed, we found that the reaction times for the 30 ms delays were greater than those for the 230 ms delays ($p < 0.01$, t test) (Churchland et al., 2006b). Comparing the

panels in the fourth column ($\tilde{\mathbf{x}}_{4,:}$, right dotted box), the neural state appears to trace out a similar path along that orthonormalized dimension following target onset, regardless of when the go cue arrives. Kalaska and colleagues (Crammond and Kalaska, 2000) have previously reported single PMd neurons with similar response properties, whereby a phasic response was only emitted after the first signal with instructional value in reaction-time (analogous to 30 ms delay) and instructed-delay (analogous to 230 ms delay) reach trials. The phasic response was interpreted as information processing that would not need to occur after the go cue if given enough time to be carried out during the delay period. Although we cannot rule out that the "phasic response" seen in the fourth column of Fig. 5.9 is primarily a sensory response (to the appearance of the reach target) rather than motor processing, such visualizations provide invaluable intuition for the recorded activity and suggest tantalizing hypotheses that can be further investigated in future studies.

The methods developed here provide a concise summary of the activity recorded across a neural population on a single trial. By extracting such a summary (i.e., the neural trajectory) for each trial, we can readily compare how the neural activity observed on one trial differs from that observed on other trials, and possibly link such differences to the subject's behavior. Such a comparison would be onerous based solely on the raw spike trains recorded simultaneously from tens to hundreds of neurons. The power of this approach is illustrated in Fig. 5.9. Among the trials with 30 ms delay, one particular trial was readily identified as an outlier, whose whose neural trajectory (red traces) appeared very different from the trajectories on other trials. Note that the visualization is extracted from neural activity alone, with no experimental timing or behavioral information provided. Can we relate this outlying trajectory to the subject's behavior? Indeed, when we labeled the neural trajectories with experimental timing markers (red, green, and blue dots), it became clear that the reaction time (i.e., the time between the green and blue dots) on the outlying trial was much longer than on the other trials. However, the neural activity around the time of the arm movement on the outlying trial matched well with that on the other trials (seen by aligning the trajectories in time based on the blue dots). This neural activity is presumably related to generating the arm movement and it is, therefore, sensible that it is time-locked to movement onset (blue dots). Such visualizations are invaluable when screening large volumes of neural data and during exploratory data

analyses. While this outlying trial provides a particularly illustrative example of how differences in the neural trajectories can be indicative of differences in single-trial behavior, we hope to relate more subtle properties of the neural trajectories to the subject's behavior in future studies.

5.5 Discussion

In this work, we have extended existing two-stage methods and developed a new method (GPFA) for extracting single-trial neural trajectories from neural population activity. For the two-stage methods, we introduced (i) dimensionality reduction techniques PPCA and FA, which explicitly account for spiking noise, (ii) the square-root transform, which approximately stabilizes the spiking noise variance across neurons and across time, and (iii) a goodness-of-fit metric, which allows for the degree of smoothing to be chosen in a principled way and for different extraction methods to be objectively compared. We then presented GPFA, which unifies the smoothing and dimensionality reduction operations in a common probabilistic framework without any loss in predictive power compared to the best two-stage method. We applied these methods to neural activity recorded during a delayed-reach task in premotor and motor cortices. We found that (i) the 61-dimensional recorded activity could be succinctly captured by neural trajectories that evolve within a far lower dimensional (8 to 12-dimensional) space, (ii) the single-trial trajectories converged over time during motor planning, an effect which was shown indirectly by previous studies, and (iii) properties of the trajectories could be related to the subject's behavior on a single-trial basis.

One of the advantages of GPFA over the two-stage methods is that the degree of smoothing (defined by the characteristic timescales τ_i) and the relationship between the low-dimensional neural trajectory and the high-dimensional recorded activity (defined by C in Eq. 5.1) can be jointly optimized. For the two-stage methods, the relationship between the low- and high-dimensional spaces is optimized *given* that the neural data have already been pre-smoothed in some way (e.g., using a Gaussian kernel with a pre-determined kernel width). This suggests a "brute force" approach to joint optimization by pre-smoothing the neural data in different ways, then optimizing the relationship between the two spaces in each case. However, the brute

force approach can only be carried out if the search space of different ways to pre-smooth the neural data is not too large. For example, allowing each neuron to have its own smoothing kernel width would only be tractable for small numbers of neurons. In Fig. 5.5*B* and *C* (red and green lines), we were able to carry out this brute force search for the two-stage methods by assuming that all neurons have the same smoothing kernel width, effectively collapsing 61 parameters down to one parameter. Despite this brute force approach to joint optimization and restricting all neurons to the same smoothing kernel width, the best two-stage method (FA with a 40 ms smoothing kernel width) was able to extract neural trajectories that look qualitatively similar to those extracted by GPFA, shown in Figs. 5.7–5.9. It is reassuring that different methods produce similar trajectories when applied to the same data. However, when compared quantitatively, the best two-stage method still yielded higher prediction error than GPFA (Fig. 5.5). It remains to be seen how important this difference in prediction error is in terms of one's ability to relate features of the neural trajectories to the subject's behavior. The leave-neuron-out prediction error is a general and fundamental criterion for measuring how well a neural trajectory captures the correlated firing rates across a neural population. Depending on the goals of the visualization and scientific questions being asked, there may be other reasonable criteria for comparing different methods for extracting neural trajectories.

It is tempting to try to relate the smoothing kernel width (40 ms) of the best two-stage method to the timescales τ_i learned by GPFA, since the SE covariance has the same shape as the Gaussian smoothing kernel. However, as shown in Fig. D.1, nearly all of the timescales learned by GPFA are greater than 40 ms. This apparent mismatch can be understood by considering the *equivalent kernel* of the SE covariance (Sollich and Williams, 2005), which takes on a sinc-like[13] shape whose main lobe is generally far narrower than a Gaussian kernel with the same width parameter. It is therefore reasonable that the timescales learned by GPFA are larger than the optimal smoothing kernel width.

Because only the GP covariance structure needs to be specified, GPFA is particularly attractive for exploratory data analyses, where the rules governing the dynamics of the system under study are unknown. Based on the trajectories obtained by GPFA, one can then attempt to define an appropriate dynamical model that describes how

[13]The sinc function is defined as $\mathrm{sinc}(x) = \sin(x)/x$.

the neural state evolves over time. Such an approach will allow us to reexamine, and potentially advance, the dynamical systems approach we have previously proposed (Yu et al., 2006). Compared with the two-stage methods, the choice of GP covariance allows for more explicit specification of the smoothing properties of the low-dimensional trajectories. This is important when investigating (possibly subtle) properties of the system dynamics. For example, one may wish to ask whether the system exhibits second-order dynamics by examining the extracted trajectories. In this case, it is critical that second-order effects not be built-in by the smoothness assumptions used to extract the trajectories. With GPFA, it is possible to select a triangular GP covariance that assumes smoothness in position, but not in velocity. In contrast, it is unclear how to choose the shape of the smoothing kernel to achieve this in the two-stage methods.

Whether a two-stage method or GPFA is used to extract neural trajectories, one should critically evaluate the assumptions made by the extraction method before using it to answer scientific questions. No method is assumption-free and one must verify that the assumptions made by the method are not trivially producing the observed effect (e.g., when studying second-order dynamics). This often requires looking at the same data with related methods that apply different assumptions to see if the observed effect holds up. Even with the same data, different scientific questions may call for the use of different methods. Examples of such assumptions include the choice of smoothing kernel or GP covariance, the use of the square-root transform, the observation noise model, the linear mapping between the low- and high-dimensional spaces[14], and edge effects when estimating finite-duration neural trajectories. To avoid possible artifacts introduced by the extraction method, one may consider first generating hypotheses by visualizing the low-dimensional neural trajectories, then testing the hypotheses using the raw high-dimensional recorded activity (e.g., Mazor and Laurent, 2005). While this approach is in principle "safer", the high-dimensional recorded activity is noisy and may mask subtle relationships that are only revealed in the (denoised) low-dimensional neural trajectories.

While being mindful of these caveats, based on our findings described in this

[14]PCA, PPCA, FA, and GPFA all assume a linear relationship between the low-dimensional state space and the high-dimensional space of square-rooted spike counts. However, because the square-root transform is a non-linear operation, the identified manifold is *non-linear* in the original high-dimensional space of firing rates (or raw spike counts).

report, we believe that the GPFA framework offers better single-trial characterization of population activity and greater flexibility for testing different scientific hypotheses relative to competing methods. Given a new dataset with neural activity recorded simultaneously across a neural population, we suggest taking the following steps to extract and visualize single-trial neural trajectories:

1. Signal conditioning: identify and remove electrode channels with crosstalk (see *Methods*), then spike sort remaining channels.

2. Apply square-root transform to binned spike counts.

3. Fit the parameters of the GPFA model using the EM algorithm, as detailed in Appendix D.

4. Using these parameters, extract neural trajectories $E[X \mid Y]$ (Eq. D.6) from the observed activity Y.

5. Apply the orthonormalization procedure described in *Methods* to the neural trajectories. This step is critical for visualization, as it orders the dimensions of the low-dimensional trajectory by the amount of data covariance explained.

6. Plot each dimension of the orthonormalized neural trajectory versus time, as in Fig. 5.7. These timecourses should be inspected for qualitative agreement with prior analyses of the same or related datasets. For example, one may expect firing rates, and therefore the neural state, to change shortly after stimulus presentation. A rough estimate of the data dimensionality can be obtained by counting the number of orthonormalized dimensions showing time-varying structure; the data dimensionality can be formally computed using the leave-neuron-out prediction error described in *Methods*.

7. Plot the top three (or any three) dimensions of the orthonormalized neural trajectories in a three-dimensional state space, as in Fig. 5.8.

Taken together, we consider steps 2 through 7 to be part of the GPFA framework for extracting and visualizing neural trajectories. Step 1 is necessary "best practices" when asking scientific questions about electrode array data.

For visualization in three dimensions, Fig. 5.5 shows that it is still better to fit a GPFA model with a large number of state dimensions (in this case, $p = 15$) and take

the top three orthonormalized dimensions, rather than to fit a GPFA model directly with $p = 3$. This allows the neural trajectories to make use of a large number of timescales, rather than just three timescales. While such a visualization is intuitively appealing, it is only able to show three selected dimensions and, therefore, may be missing important structure contained in the dimensions not plotted. This can be partially overcome by plotting different sets of three dimensions, but we are seeking better ways to visualize higher-dimensional trajectories.

The ability of the methods developed here to concisely summarize the neural events on a single trial offers a powerful tool for studying the timecourse of neural population activity. We intend to apply these methods to data recorded during other behavioral tasks and in other brain areas, as schematized in Fig. 5.1. This is enabled by the development and increasing adoption of large-scale neural recording technologies, including multi-electrode arrays and optical imaging techniques. Such analyses should provide insights into the neural mechanisms underlying cognitive processes (such as perception, decision making, attention, and motor planning), which are not directly yoked to observable quantities in the outside world and whose timecourse may differ substantially from trial to trial. More generally, these methods can be applied in experimental settings with no trial structure, for example in freely behaving animals (Jackson et al., 2007; Santhanam et al., 2007; Eliades and Wang, 2008; Chestek et al., 2009). In such settings, traditional data analysis methods relying on trial-averaging are not applicable. Instead, if large-scale neural recordings are available, the methods presented here can be applied to track the subject's instantaneous neural state during a recording session[15]. Another potential application of the developed methods is in studies of learning. While analyzing the activity of single neurons can detect the presence of learning in neural activity, it is often unclear how the activity across a neural population is changing during the learning process and why such changes might be advantageous. By tracking the subject's instantaneous neural state using the methods developed here, we may be able to further our understanding of the neural mechanisms underlying learning.

[15]Due to computational considerations (cf. Appendix D), it may be desirable to segment a recording session into multiple non-overlapping intervals before applying GPFA. The methods presented here can then be applied unchanged, even though the segments are not multiple realizations of an experimental trial.

Several extensions to the GPFA methodology can be envisaged. It may be possible to (i) couple the covariance structure of the one-dimensional GPs, which would provide for a richer description of the multidimensional neural state $\mathbf{x}_{:,t}$ evolving over time, (ii) apply non-stationary GP covariances, since the neural activity can be non-stationary, (iii) allow for non-linear relationships between the low- and high-dimensional spaces, and (iv) incorporate point-process likelihood models (Truccolo et al., 2005; Cunningham et al., 2008c) with appropriate stimulus and spike history dependence.

5.6 Acknowledgments

We thank Sandra Eisensee for administrative assistance, Melissa Howard for surgical assistance and veterinary care, Drew Haven for computing support, and Dr. Mark Churchland for experimental guidance and discussions regarding analyses.

5.7 Grants

This work was supported by NIH-NINDS-CRCNS 5-R01-NS054283, NDSEG Fellowships, NSF Graduate Research Fellowships, Gatsby Charitable Foundation, Michael Flynn Stanford Graduate Fellowship, Christopher and Dana Reeve Foundation, Burroughs Wellcome Fund Career Award in the Biomedical Sciences, Stanford Center for Integrated Systems, NSF Center for Neuromorphic Systems Engineering at Caltech, Office of Naval Research, Sloan Foundation and Whitaker Foundation.

End of Part I

Connecting Part I to Part II

At a high level, the reader can take away three messages from this first part. First, from Chapter 2 we saw that algorithmic developments can improve neural signal processing both in terms of performance and in terms of offering features unavailable in classic methods. We applied this method to motor cortical data. Second, we saw in Chapters 3 and 4 that this algorithmic research can compel computational research, and further this computational research can lead to nonneuroscientific findings of significant interest. Thirdly, Chapter 5 advanced from these developments to create a new method, GPFA, that allows visualization and analysis of populations of motor cortical neurons on single experimental trials. We used this method to uncover features of motor cortical processing that had only been seen indirectly in previous studies (convergence of across-trial variability during motor planning), and we showed that these single trial trajectories can be related to physical behavior on single trials.

There remains an important question about what new features of the brain we have discovered based on these methods. While this question is of critical scientific importance, I defer that discussion to the final concluding chapter - Chapter 9. What has certainly been accomplished in this part is that algorithms have been carefully and tractably designed that allow better processing of neural signals and the ability to look more deeply into population-level neural signals on single experimental trials.

In the coming part - Part II - we turn to the applied question of neural prosthetic system design. As described in Chapter 1, neural prosthetic systems are engineered to extract signals from the brain and process those signals to provide useful control signals for a prosthetic device. Preliminary work has shown exciting proofs of concept, but the field has not yet delivered a clinically viable system. Thus, it is of critical biomedical importance to investigate opportunities for improving the ability of these systems to decode useful control signals from cortex. Chapter 6 introduces

an algorithm that delivers significant performance improvement in a communications prosthesis. Chapter 7 then turns back to the question of estimating neural firing rates and asks whether or not these signal processing methods are relevant for neural prosthetic system design. Several methods are reviewed and then tested on experimental data to show that firing rate estimation is likely not a source of major performance improvement for prosthetic system design. This somewhat surprising finding naturally raises the question: what is important for prosthetic system design? Chapter 8 discusses several features of prosthetic system design and argues that additional research in some features will, and in some features will not, produce meaningful future performance improvements.

Taken together, Part II develops an algorithm that improves prosthetic performance, finds an aspect of system design that, despite a preponderance of research in this area, does not offer significant improvements, and finally points to features of system design that may result in future performance improvements. Following this part, Chapter 9 concludes and points to exciting opportunities for the future directions of this work.

Part II

Neural Prosthetic Systems

Chapter 6

Toward Optimal Target Placement for Neural Prosthetic Devices

Moving prosthetic systems towards clinical viability requires performance improvements in a number of domains. We first describe here an algorithm that offers improvement in design of particular types of neural prosthetic systems. Neural prostheses have been designed to estimate continuous reach trajectories (motor prostheses) and to predict discrete reach targets (communication prostheses). In the latter case, reach targets are typically decoded from neural spiking activity during an instructed delay period, before the reach begins. Such systems use targets placed in radially symmetric geometries, independent of the tuning properties of the neurons available. Here we seek to automate the target placement process and increase decode accuracy in communication prostheses by selecting target locations based on the neural population at hand. Motor prostheses that incorporate intended target information could also benefit from this consideration. We present an optimal target placement algorithm that approximately maximizes decode accuracy with respect to target locations. In simulated neural spiking data fit from two monkeys, the optimal target placement algorithm yielded statistically significant improvements up to 8 and 9% for two and sixteen targets, respectively. For four and eight targets, gains were more modest, as the target layouts found by the algorithm closely resembled the canonical layouts. We trained a monkey in this paradigm and tested the algorithm with experimental neural data to confirm some of the results found in simulation. In all, the algorithm can serve not only to create new target layouts that outperform canonical

layouts, but it can also confirm or help select among multiple canonical layouts. The optimal target placement algorithm developed here is the first algorithm of its kind, and it should both improve decode accuracy and help automate target placement for neural prostheses. This work, which has been published as Cunningham et al. (2008b), was done jointly with Byron Yu, Vikash Gilja, Stephen Ryu, and Krishna Shenoy.

6.1 Introduction

Most neural prostheses (motor prostheses) decode neural activity into commands which guide a smoothly moving on-screen cursor or robotic arm (Serruya et al., 2002; Taylor et al., 2002; Carmena et al., 2003; Hochberg et al., 2006; Srinivasan et al., 2007; Velliste et al., 2008). Some neural prostheses (communication prostheses) estimate just the intended reach target. These communications prostheses could allow severely diabled patients to communicate messages or perform simple tasks by making a series of discrete choices such as selecting keys on a keyboard (Shenoy et al., 2003; Hatsopoulos et al., 2004; Musallam et al., 2004; Santhanam et al., 2006). Motor prostheses can also incorporate neural information about the reach target into their models (Kemere et al., 2004; Yu et al., 2007; Srinivasan et al., 2007). For communication prostheses or motor prostheses with discrete reach targets, it is critical to decode the intended target accurately. There is a great deal of interest in improving the decode performance of these prosthetic systems, as increased performance will enhance usability and therefore clinical viability. There are many factors which should be considered for improving prosthetic performance, including decoding algorithms (Georgopoulos et al., 1986; Brown et al., 1998; Wu et al., 2004; Brockwell et al., 2004; Wu et al., 2006), incorporating multiple signal modalities (*e.g.*, EEG, ECoG, LFP, and spiking activity), improving recording technology, and improving design of prosthetic end effectors, be that a robotic arm or computer cursor (Lebedev and Nicolelis, 2006; Schwartz, 2004). Here we address the problem of target placement in a communication prosthetic system (or a motor prosthesis using reach target information) that uses intracortical neural spiking activity.

In the behavioral paradigm employed in communication prosthesis studies, a monkey is trained to make center-out, delayed reaches to one of a discrete number of visual

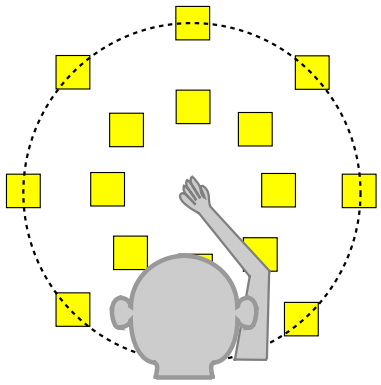


Figure 6.1: A communication prosthesis paradigm with sixteen targets. Yellow squares show placement of targets in a canonical ring topology, evenly spaced around two rings of eight targets each. Dotted line indicates the workspace bound.

targets presented on a fronto-parallel screen (Fig. 6.1). Using neural spiking activity recorded from dorsal pre-motor (PMd) cortex before the onset of movement, during the *instructed delay period*, maximum likelihood (ML) decoding algorithms can predict the intended reach target with high speed and accuracy (Santhanam et al., 2006). Since a neural prosthesis often consists of a keyboard or some other user interface, the key or target layout can be physically configured as the system designer sees fit. These prostheses (Kennedy and Bakay, 1998; Kennedy et al., 2000; Wolpaw and McFarland, 2004; Musallam et al., 2004; Santhanam et al., 2006; Hochberg et al., 2006) commonly place a number of targets (typically two to sixteen) evenly spaced around one or two rings, the radius of which is determined by the subject's maximum reach extent (Fig. 6.1, see also Fig. 2, panel B of (Santhanam et al., 2006) and Fig. 5, panel C of (Hochberg et al., 2006)). This canonical target layout, known as the *ring topology*, reflects the observation that neural activity is more strongly modulated by reach direction than reach extent (Riehle and Requin, 1989; Fu et al., 1993; Moran and Schwartz, 1999a; Messier and Kalaska, 2000; Churchland et al., 2006a). *Ad hoc* attempts at improving decode performance by altering target configurations were made previously (see target configurations in Fig. 2, panel B of (Santhanam et al., 2006)). However, if we understand the tuning properties of the particular neurons from which we are recording, we can quantitatively exploit this prior knowledge to place targets in a configuration that will yield lower decode error. Thus, our goal here is both to increase decode accuracy by placing targets optimally, and to do so in an automated fashion.

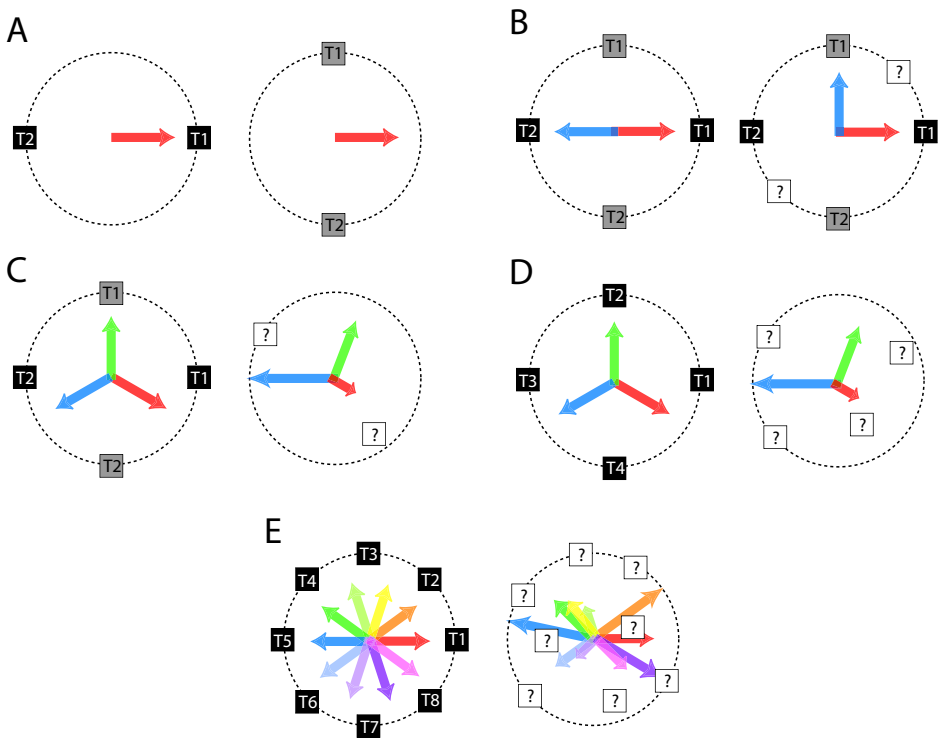


Figure 6.2: Intuition of optimal target placement problem, where we consider progressively (panels A through E) more neurons (each neuron's tuning direction and strength being represented by one arrow) and targets (black, grey, or white squares). See Introduction (Problem Intuition) for a full description.

6.1.1 Problem Intuition

To motivate our approach, we provide here an illustration of the target placement problem. We first consider a hypothetical case where we record from only one neuron, and further we suppose that this neuron's firing rate is cosine tuned with a rightwards preferred direction (Georgopoulos et al., 1982). We show this case in Fig. 6.2, panel A, where we represent this neuron with an arrow pointing right (the preferred direction). Let the length of the arrow correspond to the depth of tuning. As in Fig. 6.1, we have a dotted line corresponding to the workspace bound, which may be the monkey's reach extent or the extent of the visual field (targets must be placed within this workspace). Given this one neuron, where should we place two targets, T1 and T2, to maximize our decode accuracy? In Fig. 6.2, panel A, we show two possible target configurations. In the left subpanel, we place the targets T1 and T2 at the far right and at the far left of the workspace (targets shown in black). In this configuration, a reach to

target T1 will elicit maximal neural spiking activity, while a reach to T2 will elicit minimal activity, thereby maximizing our decode accuracy (the neural responses are most distinguishable). In contrast, we consider the right subpanel of Fig. 6.2, panel A, where we have placed the targets (shown in grey) at the top and bottom of the workspace. While this configuration is geometrically similar to the left subpanel (the targets are maximally separated on the workspace, to exploit distance tuning), we can see that this cosine-tuned neuron will fire at the same rate (on average) to both targets, and the decoder will perform at chance accuracy. Thus, we see that target placement is important, and it should also consider the neural population at hand. In other words, symmetric geometries alone are inadequate.

In Fig. 6.2, panel B, we add one neuron with identical tuning strength but different preferred direction (shown in blue). In the left subpanel, with the neurons preferring left and right (blue and red arrows), we intuit again that the horizontal target layout (T1 and T2 in black) will have optimal decode accuracy, and the vertical target layout (T1 and T2 in grey) will perform at chance. However, if we instead record from the two neurons shown in the right subpanel of panel B (where the blue neuron has an upwards preferred direction), the placement problem becomes more complicated. It seems both the black and grey pairs of targets will decode reasonably, but is there a better configuration? Perhaps, by a symmetry argument, the optimal layout is a pair of diagonally oriented targets (white targets marked with a '?'), but this intuition can not be verified without simulation or experimental testing.

Let us complicate the situation further. In Fig. 6.2, panel C, we add a third neuron (green arrow). In the left subpanel, we again see that, if these neurons had symmetric preferred directions of even tuning strength, either the pair of black targets or grey targets should decode well. In the right subpanel, however, we now change the various tuning strengths (as represented by the length of the arrows) and allow the preferred directions to be less regular. In this case, our intuition breaks down. It is unclear where to put a pair of targets to maximize decode accuracy. This loss of intuition worsens in panel D, where we now consider the same neurons, but instead consider the problem of placing four targets (T1 through T4), not the two target cases in panels A-C. Again, in the left subpanel, ideal neurons should perhaps suggest a symmetrical layout as are often used in experiments. A more realistic neural population, shown at right, significantly increases the difficulty of the target placement problem.

Finally, in Fig. 6.2, panel E, we show a case of placing eight targets when recording from ten neurons. At left, an idealized, symmetric neural population might imply a symmetric target configuration. However, the more realistic neural population (right subpanel) makes impossible any reasonable guesses about target placements. In prosthetic systems with more targets and more neurons (as in (Shenoy et al., 2003; Hatsopoulos et al., 2004; Musallam et al., 2004; Santhanam et al., 2006)), the problem of target placement only gets more difficult.

One might consider a few strategies for optimal target placement. First, as is conventional, one might lay out targets in symmetrical geometric patterns. Indeed, we see in Fig. 6.2, panel A, why this strategy can fail. Thus, the characteristics of the neural population should be considered. One might then imagine a brute force approach, choosing some two-dimensional grid (or three-dimensional, in the most general case) of possible target locations, and then picking the best choices among all target configurations on that grid. Each possible configuration has a decode accuracy that must be found via simulating many reach trials, which takes a reasonable amount of computational effort (depending on the number of targets and the number of simulated trials). Even with a coarse grid of 16 or 32 possible target locations, choosing a layout of 8 targets and simulating decode accuracy would be computationally intractable: there are over 10^5 ($_{16}C_8$, the number of combinations of 8 distinct items chosen from 16 possible items, *i.e.* $\frac{16!}{8!8!}$) and 10^8 ($_{32}C_8$ or $\frac{32!}{24!8!}$) choices for these layouts with grids of size 16 and 32, respectively. These difficulties with initial approaches led us to consider the problem from a communications theory perspective.

To our knowledge, this problem has not yet been investigated. We present the optimal target placement algorithm (OTP), which uses Kullback-Leibler divergence to provide a constellation of optimal target placements. We describe the method and then compare the decode performance of the optimal constellation with canonical ring topologies, using both simulated and experimental neural data.

An introduction to this algorithm was previously published in preliminary form (Cunningham et al., 2006).

6.2 Methods

6.2.1 Overview

We want to construct an algorithm that places reach targets such that they are maximally distinguishable (to achieve optimal performance) in terms of the neural signals we record. To do so, we must first define a model that relates the reach target position to neural spiking during the delay period. We then consider a rule for decoding a particular target, given an observation of spike data. These steps are detailed in "Spike Count Model and Decoding" below. This rule implies a decode error (our measure of prosthetic performance) that is a function of the target locations. Ideally, we could then minimize decode error by moving reach targets appropriately. This general problem is intractable. However, by making standard, reasonable approximations to put this error function (a function of the target locations) into a solvable form, we can optimize the function to produce a set of target placements that approximately minimizes decode error. These steps are detailed in "Optimal Target Placement Algorithm" below. Finally, we test this method with data from two monkeys trained to perform reaches to canonically placed targets. For both monkeys, we fit a neural population (using real reaches) and evaluate decode performance on simulated neural data generated from canonically placed and optimally placed targets (hereafter, simulated data). The second monkey also performed real reaches to both canonically and optimally placed targets, and we compare decode accuracy (hereafter, experimental data). These steps are detailed in "Reach Task and Neural Recordings" and "Evaluating Decode Performance" below.

6.2.2 Spike Count Model and Decoding

We must first consider how target position is reflected in neural spiking. As described above, we present a reach target on the screen during an instructed delay period. We call this time period Δ (*e.g.*, $\Delta = 200$ ms, the window beginning 150ms after target presentation and before the subject is given a movement cue, as used in (Santhanam et al., 2006)). We collect spike counts from K neural units, and the frequency of each unit's spiking (that is, the number of spikes) is indicative of the intended reach target to an extent that allows target location to be predicted (Santhanam et al., 2006;

Hatsopoulos et al., 2004; Musallam et al., 2004; Shenoy et al., 2003). We choose a simple firing rate model (as in (Smith and Brown, 2003; Yu et al., 2007)) and a simple spiking model (as in (Zhang et al., 1998; Yu et al., 2007)). We will later discuss more advanced models, but even these basic models help to simply illustrate the conceptual advance that this method offers.

Let us consider M reach targets placed on a screen as in Fig. 6.1 (where M=16). We define each target by its Cartesian position on the screen $\mathbf{x}_m \in \Re^2$ (for all M targets $m \in \{1, \ldots, M\}$). We define the center of the screen as the origin, but the Optimal Target Placement algorithm will be invariant to that choice. We call the collection of all M targets a *constellation* of targets $\boldsymbol{\chi} \in \Re^{2M}$, that is $\boldsymbol{\chi} = [\mathbf{x}_1^T, \ldots, \mathbf{x}_M^T]^T$.

Having defined the target constellation, we must define a model that maps target position to neural spiking. Let us assume we record from K neural units. Then we map position $\mathbf{x}_m$ to a neural firing rate for the kth neural unit (as in (Smith and Brown, 2003; Yu et al., 2007)) using:

$$f_k(\mathbf{x}_m) = e^{\mathbf{c}_k^T \mathbf{x}_m + d_k}, \tag{6.1}$$

where d_k specifies a baseline firing rate, and $\mathbf{c}_k$ specifies both the preferred direction (Georgopoulos et al., 1982) and the depth of tuning modulation for unit k. The linear mapping $\mathbf{c}_k^T \mathbf{x}_m + d_k$ implies a cosine tuning model (Georgopoulos et al., 1982; Moran and Schwartz, 1999a,b). We group these parameters $\mathbf{c}_k$,d_k (over all K neural units) into $C \in \Re^{2\times K}$ (the matrix with columns $\mathbf{c}_k$) and $\mathbf{d} \in \Re^{K\times 1}$ (the vector of elements d_k). Thus, $f_k(\mathbf{x}_m)$ calculates the delay period firing rate underlying the spiking of unit k, when the target m is presented at position $\mathbf{x}_m$.

To relate this firing rate to spike counts, we use a simple Poisson count model (Zhang et al., 1998; Yu et al., 2007). Specifically, we assume the delay period spiking activity for one neural unit, when conditioned on the target m (at position $\mathbf{x}_m$), is independent of other neural units and of its own spiking history. The probability of all observed spike counts $\mathbf{y}$ (the vector $\mathbf{y} \in \Re^{K\times 1}$ is a vector of non-negative integer spike counts), during the delay period Δ, is then

$$p(\mathbf{y} \mid m) = \prod_{k=1}^{K} \text{Poisson}(y_k; f_k(\mathbf{x}_m)\Delta) \quad = \prod_{k=1}^{K} \frac{(f_k(\mathbf{x}_m)\Delta)^{y_k}}{y_k!} e^{-(f_k(\mathbf{x}_m)\Delta)} \tag{6.2}$$

According to this model, on a given trial, the presented target m^* at position $\mathbf{x}_{m^*}$ is chosen by the experimenter, where $m^* \in \{1, \ldots, M\}$. The observed spike counts $\mathbf{y}$, conditioned on m^*, are assumed to be distributed according to Eq. 6.2. We record $\mathbf{y}$ and want to decode the identity of the presented target m^* from among the M possible choices. We note that we only consider spike counts from the delay period Δ, during which we assume the reach target is fixed and the firing rate (Eq. 6.1) is constant. Thus, all decodes are made from that time period alone (and accordingly, error rates and all other values are calculated during that fixed window). To decode a reach target, we use maximum *a posteriori* (MAP) decoding (Zhang et al., 1998):

$$\widehat{m} = \underset{m}{\text{argmax}}\, p(m \mid \mathbf{y}) \tag{6.3}$$

$$= \underset{m}{\text{argmax}}\, \frac{p(\mathbf{y} \mid m)p(m)}{p(\mathbf{y})} \tag{6.4}$$

$$= \underset{m}{\text{argmax}}\, p(\mathbf{y} \mid m) \tag{6.5}$$

where $\widehat{m}$ is the index of the estimated reach target (at position $\mathbf{x}_{\widehat{m}}$). Eq. 6.3 states that we choose the decoded target as the most likely target, given the neural data. Eq. 6.4 is obtained using Bayes' rule; Eq. 6.5 is a result of all reach directions being equally likely (since in our experiments all targets are presented an equal number of times)[1] and $p(\mathbf{y})$ not being dependent on m. Thus, the decode rule (Eq. 6.3) reduces to a maximum likelihood (ML) estimator (Eq. 6.5) (Papoulis and Pillai, 2002). If the assumptions of the model are satisfied (*i.e.*, Poisson spiking statistics, cosine tuning, etc.), this decode rule will minimize the total error probability (at a given target constellation $\boldsymbol{\chi}$) (Cover and Thomas, 1991):

[1] If the targets were not presented with equal frequency (for example, in a keyboard application, one might know that certain targets/keys will be used more often than others), then Eq.6.5 would still have $p(m)$ (MAP estimation). The OTP algorithm can be extended to incorporate this change; see Future Work in DISCUSSION.

$$P_{\text{error}} = \sum_{m=1}^{M} P(\{\widehat{m} \neq m\} \mid \{m^* = m\}) \ , \tag{6.6}$$

In words, Eq. 6.6 is the probability that, when any target m is presented (the presented target $m^* = m$), some other target is erroneously decoded (the decoded target $\widehat{m} \neq m$). Thus, the goal of our optimal target placement algorithm is to choose the constellation $\boldsymbol{\chi}$ that will minimize the total probability of decode error of Eq. 6.6.

6.2.3 Optimal Target Placement Algorithm (OTP)

This general problem of minimizing total error probability of Eq. 6.6 (over the constellation $\boldsymbol{\chi}$), well known in communications literature (see *e.g.*, (Proakis and Salehi, 1994)), is often analytically intractable (*i.e.*, there is no closed form solution, which will be required so we can calculate how changes in target position effect decode error). Indeed, minimizing Eq. 6.6 is similarly difficult in our case. As a result, it is common to instead minimize the worst pairwise error probability (we denote pairwise probabilities P_{pair})(Gockenbach and Kearsley, 1999). Pairwise error probability is simpler to calculate than total error probability because pairwise error does not consider the influence of other targets. For example, a pair of targets might have a certain error rate in isolation, but that may change with the presence of a third target, since the correct target can now be mistaken for this third target as well. Minimizing the worst pairwise error probability is equivalent to minimizing an upper bound to Eq. 6.6[2]. That is, instead of considering all the targets jointly, we consider all pairs of targets. We then select the least distinguishable ("worst") pair of targets (that is, the pair with the highest error rate when trying to decode which of these two targets is the intended reach goal), and we will try to minimize this error rate (make these two targets more distinguishable). Doing this procedure jointly across all pairs of targets should yield a lower global decode error (Eq. 6.6). Mathematically, we define

[2]This upper bound can be seen by expanding each term in the sum of Eq. 6.6 using the union of events bound (Boole's inequality), such that $\sum_{m=1}^{M} P(\{\widehat{m} \neq m\} \mid \{m^* = m\}) \leq \sum_{m=1}^{M} \sum_{m' \neq m} P_{pair}(\{\widehat{m} = m'\} \mid \{m^* = m\})$. This sum of pairwise errors is upper bounded by $M(M-1)$ times the worst pair, and thus minimizing the worst pair is equivalent to minimizing an upper bound on total error probability.

the solution to this problem $\boldsymbol{\chi}_{otp}$ (the optimal constellation) as:

$$\boldsymbol{\chi}_{otp} = \underset{\boldsymbol{\chi}}{\operatorname{argmin}}\left(\max_{m' \neq m} P_{pair}\big(\{\widehat{m} = m'\} \mid \{m^* = m\}\big)\right). \tag{6.7}$$

The inner expression $P_{pair}(\cdot)$ is the pairwise probability of error between two targets m (the correct, presented target m^*) and m' (the erroneously decoded target $\widehat{m}$). For M targets, there are $M(M-1)$ such probabilities of error (all target pairs). The maximum of these probabilities of error is the worst pair in that it has highest decode error. Finally, the outermost expression (argmin$(\cdot)$) finds the constellation $\boldsymbol{\chi}$ (the collection of target positions) which minimizes this worst pairwise error. Thus, Eq. 6.7 provides a constellation of targets that minimizes the worst pairwise error over all targets.

To calculate the probability of decode error between any pair of targets, we must consider the spiking noise introduced by the Poisson output distributions (Eq. 6.2). Owing to the noisy Poisson model, particular spike counts will erroneously decode a target m' when in fact the presented target was m. There is no closed-form expression for the probability of decode error between two Poisson noise distributions (Verdu, 1986). Kullback-Leibler (KL) divergence is often used as a close proxy to pairwise error probability (Gockenbach and Kearsley, 1999; Johnson et al., 2001; Johnson and Orsak, 1993). KL divergence measures how different two probability distributions are. Pairwise error probability also measures how different two distributions are, in that it quantifies how often a draw from one distribution will be incorrectly classified as having been drawn from another distribution. The use of KL as a proxy to error probability is intuitively sound, and our simulations have shown that increasing KL divergence (making the two distributions more different) corresponds well to decreasing error probability. KL is commonly used when error probability is not closed form (Gockenbach and Kearsley, 1999; Johnson et al., 2001; Johnson and Orsak, 1993), with the understanding that making distributions more distinguishable (increasing the KL divergence) will generally reduce probability of error also. The relationship between KL and error probability can be motivated mathematically by returning to

the two-target case of the ML decode rule (Eq. 6.5) and writing it as

$$\widehat{m} = \begin{cases} m & \text{if } \frac{p(\mathbf{y}|m)}{p(\mathbf{y}|m')} \geq 1, \\ m' & \text{otherwise,} \end{cases} \tag{6.8}$$

(*i.e.*, the decoder predicts m if $p(\mathbf{y} \mid m)$ is larger than $p(\mathbf{y} \mid m')$, and m' otherwise). Assuming a trial has reach target m presented, we want to maximize the likelihood ratio in Eq. 6.8 over all possible instances of $\mathbf{y}$. Doing so will provide the maximum distinguishability between the distributions ($p(\mathbf{y} \mid m)$ and $p(\mathbf{y} \mid m')$) and will miminize the chance that the likelihood ratio will be less than 1 (which implies an error). We can equivalently maximize the logarithm of this likelihood ratio, and, taking the expectation to consider all possible $\mathbf{y}$, we have the KL divergence

$$KL(\mathbf{x}_m \parallel \mathbf{x}_{m'}) = E_{\mathbf{y}|m}\left[\log\frac{p(\mathbf{y} \mid m)}{p(\mathbf{y} \mid m')}\right] \tag{6.9}$$

where the expectation is taken with respect to $\mathbf{y}$ given m. We write KL as a function of the target positions $\mathbf{x}_m$ and $\mathbf{x}_{m'}$ to emphasize that it calculates our proxy to error probability *in terms of the target positions*. Thus, KL in Eq. 6.9 is a measure of distinguishability between targets m and m'; to minimize the probability of decode error, we want to maximize Eq. 6.9 by changing target locations $\mathbf{x}_m$ and $\mathbf{x}_{m'}$. Under the Poisson output distribution we introduced in Eq. 6.2, KL divergence can be calculated exactly (substitute Eq. 6.2 into Eq. 6.9; see Appendix E.1 for details):

$$KL(\mathbf{x}_m \parallel \mathbf{x}_{m'}) \;=\; \Delta\sum_{k=1}^{K}\left(f_k(\mathbf{x}_{m'}) - f_k(\mathbf{x}_m) + f_k(\mathbf{x}_m)\log\frac{f_k(\mathbf{x}_m)}{f_k(\mathbf{x}_{m'})}\right). \tag{6.10}$$

Note that this form is not constrained by the form of the firing rate $f_k(\mathbf{x}_m)$ in Eq. 6.1, allowing OTP to easily generalize for other, more complex firing rate models (on the other hand, changing the spiking model - Eq. 6.2 - will change the form of the KL; see Future Work in DISCUSSION).

In summary, we have replaced the analytically intractable probability of error between two Poisson distributions with the tractable form of Eq. 6.10. We have done so with the understanding that finding a pair of target positions ($\mathbf{x}_m$, $\mathbf{x}_{m'}$) that maximizes the KL divergence from $\mathbf{x}_m$ to $\mathbf{x}_{m'}$ is nearly equivalent to finding that which

minimizes the probability of decoding m' when m was presented. Mathematically, we write

$$\underset{\mathbf{x}_m,\mathbf{x}_{m'}}{\operatorname{argmax}} KL(\mathbf{x}_m \parallel \mathbf{x}_{m'}) \;\approx\; \underset{\mathbf{x}_m,\mathbf{x}_{m'}}{\operatorname{argmin}} P_{pair}\big(\{\widehat{m} = m'\} \mid \{m^* = m\}\big). \tag{6.11}$$

For the Poisson distributions used here, as we have noted, our simulations show that the relationship between error probability and KL divergence is nearly monotonic. Thus, we believe maximizing KL divergence is a valuable proxy to minimizing probability of error in this problem. One might also consider using the Chernoff bound (Cover and Thomas, 1991), which proves an upper bound on error probability with respect to KL divergence in hypothesis testing. However, this bound has been found to be loose (Johnson et al., 2001). Though not a provable bound (upper or lower) on error probability, KL divergence does provide a very close proxy; further supporting arguments can be found in (Johnson et al., 2001; Johnson and Orsak, 1993).

Having made this approximation, we can return to our problem of interest, namely finding the optimal target placement $\boldsymbol{\chi}_{otp}$ as in Eq. 6.7. Using KL divergence, Eq. 6.7 becomes

$$\boldsymbol{\chi}_{otp} = \underset{\boldsymbol{\chi}}{\operatorname{argmax}} \left(\min_{m \neq m'} KL(\mathbf{x}_m \parallel \mathbf{x}_{m'}) \right) \tag{6.12}$$

Note that the two targets with the smallest KL divergence (the inner expression in Eq. 6.12) are the least distinguishable and thus should have the highest probability of error (the worst pair, as in Eq. 6.7). Accordingly, improving the worst pair in Eq. 6.7 (minimizing the maximum error probability) is the same as improving the worst pair in Eq. 6.12 (maximizing the minimum KL divergence). An algorithm solving this problem will push the target positions $\mathbf{x}_m$ as far apart as possible from each other in terms of KL divergence. We impose a workspace limitation such as how far a subject's arm can reach, the extent of the subject's visual field, or the bounds imposed by the prosthesis (such as a computer screen). We capture this limitation with a constraint γ on the Euclidean distance of $\mathbf{x}_m$ from the center of the workspace screen (other constraints, such as a rectalinear workspace, could be readily included as well). With this contraint, our optimal target placement $\boldsymbol{\chi}_{otp}$ is the solution to:

$$\begin{aligned} &\underset{\boldsymbol{\chi}}{\text{maximize}} \left(\min_{m \neq m'} KL(\mathbf{x}_m \parallel \mathbf{x}_{m'}) \right) \\ &\text{subject to} \quad \|\mathbf{x}_m\| \leq \gamma \quad \forall\ m = 1 \ldots M \end{aligned} \tag{6.13}$$

We call the algorithm that solves Eq. 6.13 the optimal target placement algorithm (OTP). We applied sequential quadratic programming (SQP) (Gockenbach and Kearsley, 1999; Boggs and Tolle, 1996) to solve Eq. 6.13. It is important to note here that SQP is an established technology for optimizing nonlinear, constrained objectives such as Eq. 6.13. For example, the MATLAB (The MathWorks, Natick, MA) function `fmincon` (nonlinear, contrained optimization solver) uses SQP. SQP finds an optimum to this problem (Eq. 6.13 is not convex in $\boldsymbol{\chi}$), and this optimum depends on the choice of seed constellation $\boldsymbol{\chi}_0$. To find the global optimum, we solved the SQP multiple times (eight to thirty-two, depending on the number of targets in the constellation) starting at randomly chosen $\boldsymbol{\chi}_0$. After these iterations, a "best" optima (best in terms of having the largest objective, *i.e.* the minimum worst pair KL divergence, as in Eq. 6.12) typically appeared several times, giving confidence that we had indeed found the global optimum. This solution was designated the optimal constellation $\boldsymbol{\chi}_{otp}$. We include notes on our use of SQP in Appendix E.2.

6.2.4 Reach Task and Neural Recordings

Animal protocols were approved by the Stanford University Institutional Animal Care and Use Committee. We trained two adult male monkeys (*Macaca mulatta*, monkey H and monkey L) to perform delayed center-out reaches for juice rewards. As illustrated in Fig. 6.3, visual targets were back-projected onto a fronto-parallel screen 30 cm in front of the monkey. The monkey touched a central target and fixated his eyes on a crosshair adjacent to the central target. After a center hold period of 300 to 500 ms for monkey L and 400 to 600 ms for monkey H, a pseudo-randomly chosen target was presented at one of the target locations. For the canonical reach data sets, the sixteen targets were placed in two rings of eight, as shown in Fig. 6.1, of radius 7 and 12 cm for monkey H (4 and 8 cm for monkey L). After a pseudo-randomly chosen instructed delay period (monkey H: uniformly distributed between 200 and 500 ms; monkey L: exponentially distributed with a mean of 750, 850, or 950 ms, shifted to be no less

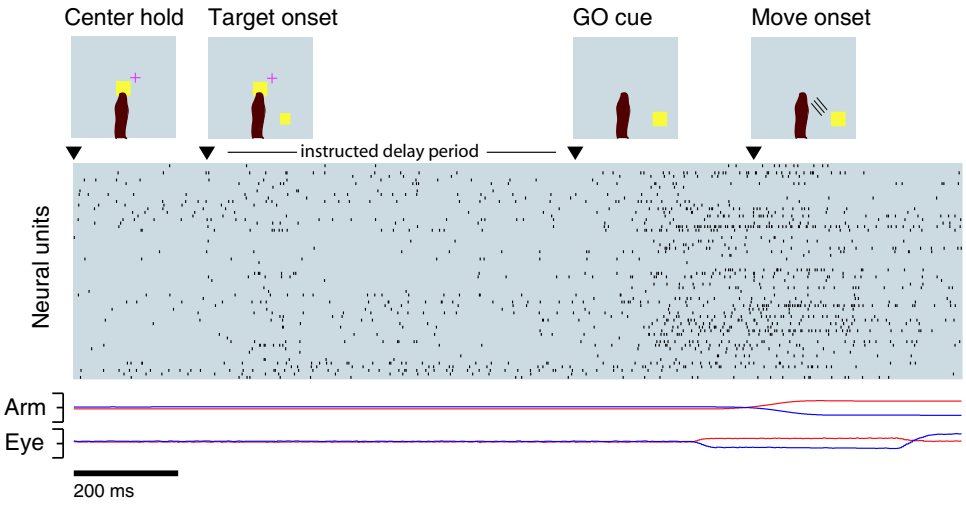


Figure 6.3: Task timeline (top), simultaneously-recorded spike trains (middle), and arm and eye position traces (bottom) are shown for a *single trial*. Red and blue lines correspond to horizontal and vertical position, respectively. The range of movement for the arm and eye position (on the screen) is ±15 cm from the center target. Neural unit activity and physical behavior were taken from trial from experiment `H20041106.1`.

than 50, 100, or 150 ms), the "go" cue (signaled by both the enlargement of the target and the disappearance of the central target) was given, and the monkey reached to the target. After a hold time of 200 or 300 ms at the reach target (depending on the experimental day), the monkey received a liquid reward. The next trial started 100-400 ms later (depending on the experimental day). Eye fixation at the crosshair was enforced throughout the delay period. Reaction times (defined as the time between the "go" cue and movement onset) were enforced to be greater than 80 or 100 ms and less than 400 or 425 ms (depending on the experimental day).

During experiments, the monkey sat in a custom chair (Crist Instruments, Hagerstown, MD) with the head braced. The presentation of the visual targets was controlled using the Tempo software package (Reflective Computing, St. Louis, MO). A custom photo-detector recorded the timing of the video frames with 1 ms resolution. The position of the hand was measured in three dimensions using the Polaris optical tracking system (Northern Digital, Waterloo, Ontario, Canada; 60 Hz, 0.35 mm accuracy), whereby a passive marker taped to the monkey's fingertip reflected infrared light back to the position sensor. Eye position was tracked using an overhead infrared camera (Iscan, Burlington, MA; 240 Hz, estimated accuracy of 1°).

A 96-channel silicon electrode array (Cyberkinetics, Foxborough, MA) was implanted straddling dorsal pre-motor (PMd) and motor (M1) cortex (left hemisphere for both monkey H and monkey L), as estimated visually from local landmarks, contralateral to the reaching arm. Surgical procedures have been described previously (Churchland et al., 2006b; Santhanam et al., 2006; Hatsopoulos et al., 2004). Spike sorting was performed offline using techniques described in detail elsewhere (Sahani, 1999; Santhanam et al., 2004; Zumsteg et al., 2005). Briefly, neural signals were monitored on each channel during a two minute period at the start of each recording session while the monkey performed the behavioral task. Data were high-pass filtered, and a threshold level of three times the RMS voltage was established for each channel. The portions of the signals that did not exceed threshold were used to characterize the noise on each channel. During experiments, snippets of the voltage waveform containing threshold crossings (0.3 ms pre-crossing to 1.3 ms post-crossing) were saved with 30 kHz sampling. After each experiment, the snippets were clustered as follows. First, they were noise-whitened using the noise estimate made at the start of the experiment. Second, the snippets were trough-aligned and projected into a four-dimensional space using a modified principal components analysis. Next, unsupervised techniques determined the optimal number and locations of the clusters in the principal components space. Events assigned to each cluster are considered spikes for a given neural unit.

Fig. 6.3 shows the delayed reach task timeline, along with neural and behavioral data for a single trial with a lower-right reach target. We refer to the time between reach target onset and the "go" cue as the delay period. It is the neural activity during this delay period that will be used to predict the reach target.

The monkeys were trained over several months, and multiple data sets of the same behavioral task were collected. Each data set was collected in one day's recording session. For monkey H, all reaches were made to canonically placed targets. For monkey L, each data set was split into two segments, the first comprising reaches to a canonical target topology, and the second to an OTP constellation. After collecting 700-2000 trials of the canonical topology, the task was stopped. Units isolated by the spike sorting method were fit to the cosine tuning model of Eq. 6.1. We counted spikes for the 200 ms period that started 150 ms after target onset (a 200 ms integration window, *i.e.*, $T_{skip} = 150$ ms and $T_{int} = 200$ ms, in the terminology of (Santhanam

et al., 2006)). We fit C, $\mathbf{d}$ from the neural data by maximizing the data likelihood (Eq. 6.2) taken across all trials. This fitting problem is convex (in C, $\mathbf{d}$) and can be readily solved using Newton's Method (Boyd and Vandenberghe, 2004) (`glmfit` in MATLAB will also readily solve this problem). This population of neural fits was then given to the OTP algorithm, which then generated an optimal target topology. This entire OTP process generally took less than ten minutes on 2006era workstations (Linux Fedora Core 4 with 64 bit, 2.2-2.4GHz AMD processors and 2-4GB of RAM) running MATLAB (R14). For monkey H, this neural population fitting was done offline to provide neural tuning data for OTP simulation. For monkey L, the task was begun again with reaches to the newly configured OTP target topology, typically for 700-1500 more trials. For both the canonical and OTP trials, we only analyzed successful trials (where the monkey obeyed the hold times, reached to the target with a proper reaction time, etc.) that had delay periods long enough to allow T_{skip} and T_{int} as just described. This screening typically left 300-800 valid OTP and 300-800 valid canonical trials for analysis (we used equal numbers of OTP and canonical trials so performance comparisons could be meaningfully made). This segmentation allows us to analyze and compare decode performance from the canonical and optimal target topologies.

6.2.5 Evaluating Decode Performance in Experimental Data

Collecting experimental data allows us to verify the performance improvements we see in simulation. In "Spike Count Model and Decoding" (above), we introduced a maximum likelihood decoder that (when the model assumptions are satisfied) minimizes the probability of decode error (Eq. 6.5). For simulated trials, we know the neural parameters C and $\mathbf{d}$, and thus we calculate the firing rate $f_k(\mathbf{x}_m)$ exactly for any target position $\mathbf{x}_m$ (in simulation, the data fit the model of Eq. 6.1 and Eq. 6.2 exactly). In this case, we use the ML decoding rule of Eq. 6.5 directly. However, in experimental reach trials, we do not have access to the neural parameters C and $\mathbf{d}$, and thus we do not have $f_k(\mathbf{x}_m)$. Instead, for each target $\mathbf{x}_m$, we must fit an estimate $\hat{f}_k(\mathbf{x}_m)$. Since each unit y_k is modeled as Poisson (conditioned on the target $\mathbf{x}_m$), y_k has expected value of $f_k(x_m)\Delta$. With a set of training trials to a particular target, our estimate $\hat{f}_k(\mathbf{x}_m)$ is the empirical mean (normalized by Δ) of those training trials (a ML estimator of $f_k(\mathbf{x}_m)$ (Papoulis and Pillai, 2002)).

In an experimental data set, we have J blocks of trials, where a block consists of one trial to each of the M reach targets. We define the neural data collected during the delay period of each trial as $\mathbf{y}^{(j,m)}$ for a block $j \in \{1, \ldots, J\}$ and a target $m \in \{1, \ldots, M\}$. To decode a single trial, we use J-fold cross validation (Duda et al., 2001). For a given block j of reach trials, we exclude the block as a test data set and use all other $(J-1)$ blocks as the training set to train the decoder (*i.e.*, estimate $\hat{f}_k(x_m)$ for all $k \in \{1, \ldots, K\}$) and $m \in \{1, \ldots, M\}$). With these parameter estimates, we can again use the ML decoder of Eq. 6.5, as in (Shenoy et al., 2003; Hatsopoulos et al., 2004; Musallam et al., 2004; Santhanam et al., 2006). This J-fold cross validation is repeated across all blocks of trials and produces a total decode performance for a given data set.

We note that $\hat{f}_k(\mathbf{x}_m)$ does not in general equal $f_k(\mathbf{x}_m)$ because the empirical mean over the training trials we collected will not be exact (even if the firing rate model holds). This factor may degrade decoder performance in experimental data, but such performance reductions should be seen equally for OTP and canonical topologies. In this study we are only interested in how the different target constellations compare in decode accuracy, not their absolute values. We confirmed in simulation that using the empirical mean resulted in similar performance reductions across both topologies, thereby suggesting that OTP is no more susceptible to this source of error than is the canonical topology. The accuracy of the target decoder also varies with the duration and placement of the time window in which spikes are counted and the spike count model $P(\mathbf{y} \mid \mathbf{x}_m)$ that is used (Hatsopoulos et al., 2004; Santhanam et al., 2006). Optimizing these aspects of the target decoder (which we again expect to affect performance equally across topologies) is beyond the scope of this work and is treated in detail in (Santhanam et al., 2006).

6.3 Results

As we saw in Fig. 6.2, for small numbers of targets and neural units, we can make a reasonable prediction about where the optimal targets should be placed, even without the use of an optimization algorithm. In the simplest case, we seek to place two targets optimally with only one neural unit (Fig. 6.2, panel A). Given the preferred direction of the unit $\mathbf{c}_1$, the targets should be placed as far apart as possible (on

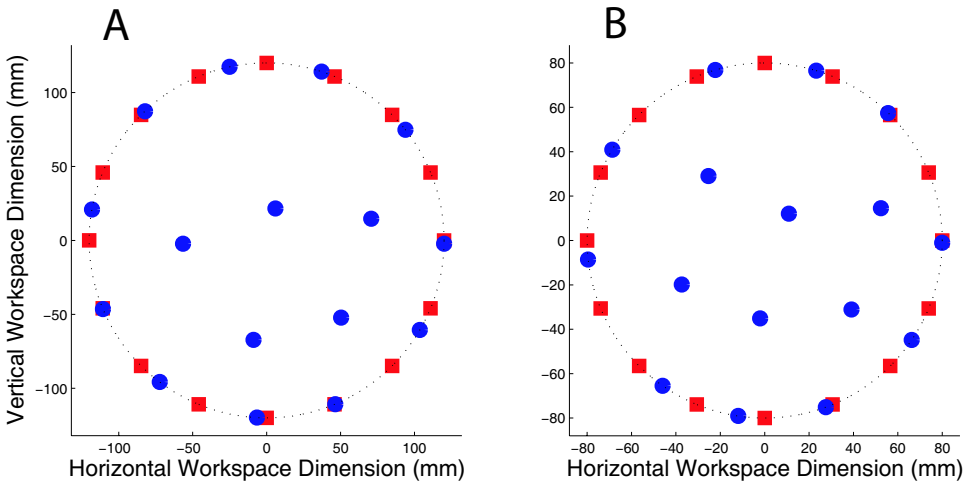


Figure 6.4: Sixteen target placement examples from: A) data set `H20041119`, and B) data set `L20061030`. Blue circles: OTP solution; red squares: a canonical ring topology. Workspace bound shown as a dotted line (γ=120 mm in A, 80 mm in B).

the circular workspace bound, as firing rate is typically modulated by reach target distance) along the axis defined by $\mathbf{c}_1$. In this configuration, the presentation of one target elicits maximal firing, while the other target gives miminal firing. Indeed, our SQP approach to optimal target placement yields this result. Extending beyond this trivial case, the utility of OTP becomes apparent when looking at larger neural populations and larger numbers of targets.

With a population of neural units that are fairly uniform in their preferred directions and tuning strength, we imagine that the placement of four or eight targets will reduce effectively to a geometric problem, and placing the targets evenly around a ring will produce a near optimal result. We will validate this intuition below. When the number of targets grows larger, intuition breaks down: for example, with sixteen targets, should they be placed evenly around the circular workspace bound? Should they be placed in two rings; if so, how many targets in each ring? OTP gives answers to these questions. Two examples are shown in Fig. 6.4, where OTP returns a constellation (blue circles) with eleven targets spaced roughly evenly around the circular workspace bound, and with five targets placed elsewhere in the workspace for monkey H (in panel A). For monkey L (panel B), OTP finds a constellation with ten targets on the workspace bound and six placed on the workspace interior. Despite the complexity of this problem, there is some intuition to be gleaned from the constellations

discovered by the OTP algorithm; see DISCUSSION (Intuition Gained from OTP).

6.3.1 Simulated Data Results

Having shown a few examples of optimal target placements, we now turn to systematic performance comparisons of OTP vs. canonical ring topologies. We randomly drew a set of K units from one of two collected data sets: one from monkey H (`H20041119`), and one from monkey L (`L20061030`). We ran OTP to find the constellation $\boldsymbol{\chi}_{otp}$, and then we generated simulated spike counts for 1000 trials to each of M optimally placed targets according to Eq. 6.2 (*i.e.*, $M \times 1000$ total trials). We then computed decode accuracy using the method described above in "Evaluating Decode Accuracy." We also simulated 1000 trials to each target of the canonical ring topology (again, $M \times 1000$ total trials).[3] This whole procedure was repeated 100 times (10 times for the sixteen target case, due to computational limitations) for each K.

These results are shown for monkey H in Fig. 6.5 for two, four, eight, and sixteen targets, and similarly for monkey L in Fig. 6.6. In Fig. 6.5, panel A, for two targets, OTP provides up to 8% improvement in decode accuracy (from 71% to 79% with K=2, for example). OTP provides similar results for Monkey L, raising performance 6% (from 71% to 77% with K=4, for example). As K grows and decode accuracy saturates to the performance ceiling of 100%, we expect the canonical topology to approach the performance of OTP, and indeed we see this effect. In both the four and eight target cases, there is less improvement above the ring topology for both monkeys H and L, with performance improvements ranging from 0 to 3% (monkey H) and 0 to 1% (monkey L). This is not surprising: the OTP layouts closely resemble canonical ring topologies. For example, a four or eight target OTP layout is often just a rotated version of a canonical layout, which does not look much different than a canonical layout (*i.e*, a rotated version of the black targets Fig. 6.2, panels D and E appears quite similar to an unrotated constellation). Contrast this to a two target case, where a rotation of a pair of targets can look significantly different (*i.e.*, in

[3] To get a true average performance for the canonical topology, we rotated the ring topology across trials to prevent any possible bias in the results. For example, if we chose a vertical canonical layout in the two target case (as in Fig. 6.2, panel A, right subpanel) and these particular neural populations had more tuning strength in the horizontal axis, then canonical layouts would be artificially punished in decode performance (so too, canonical performance could be artificially inflated if we instead chose the layout of Fig. 6.2, panel A, left subpanel). Rotating the canonical targets ensures a fair comparison between decode performances.

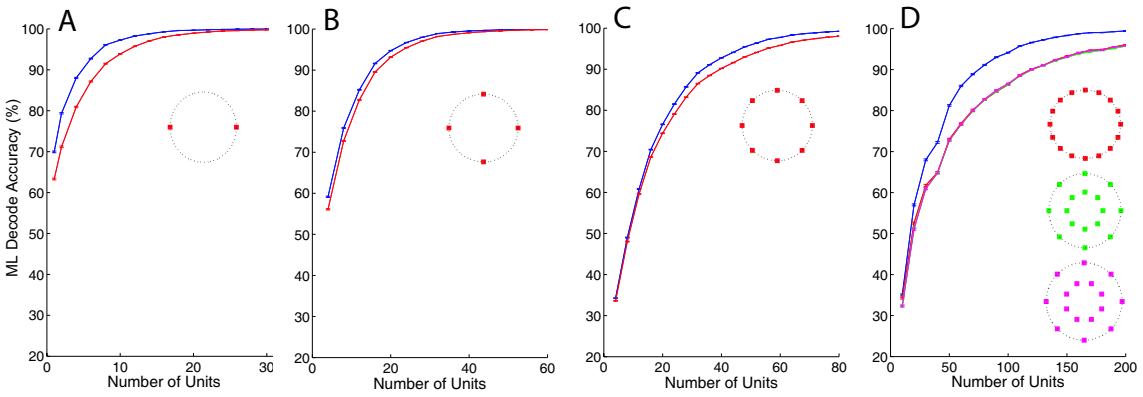


Figure 6.5: Comparison of performance in simulated data: Optimal target placements vs ring topologies. Monkey H (H20041119). A) Two Targets. B) Four Targets. C) Eight Targets. D) Sixteen Targets. Blue and red lines show performance under OTP and a single ring topology, respectively. In D), green and magenta lines show performance under the aligned and staggered double ring topologies, respectively (red, green, and magenta curves are highly overlapped). Error bars (vanishingly small, due to hundreds of thousands of simulation trials) based on a binomial distribution with 95% confidence level (see Zar (1999)). Insets show different ring topologies tested.

Fig. 6.2, panel A, the black and gray pairs of targets are quite different). Thus, we do not expect a substantial performance difference in the four and eight target cases.

At larger target constellations, we can again see substantial improvements offered by OTP. Fig. 6.5, panel D illustrates the performance of the optimal configuration in the sixteen target case with monkey H, and similarly in Fig. 6.6, panel D for monkey L. We compare OTP to three canonical ring topologies: sixteen targets evenly spaced around the workspace bound, two radially aligned rings of eight targets each, and two radially staggered rings of eight targets each. In our experience, most OTP constellations seen in the sixteen target case for both monkeys (for different values of K and different sets of units drawn at random) place four to six interior targets and ten to twelve on the workspace bound; examples are shown in Fig. 6.4. See DISCUSSION (Intuition Gained from OTP) for comments about why these constellations are sensible results of the algorithm. Over a range from 50-100 units, OTP target topologies yield 8-9% average improvement over the canonical ring topologies in monkey H.[4] For monkey L, more units are required to see substantial performance gains,

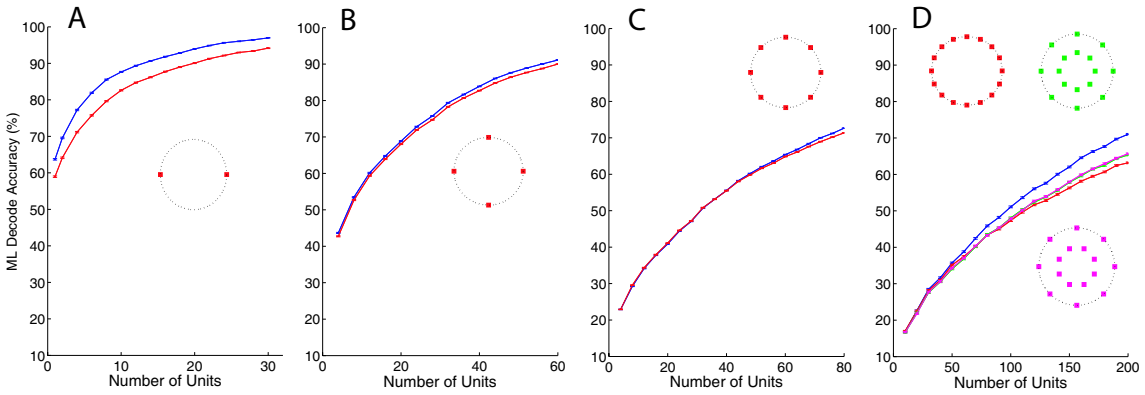


Figure 6.6: Comparison of performance in simulated data: Optimal target placements vs ring topologies. Monkey L (L20061030). A) Two Targets. B) Four Targets. C) Eight Targets. D) Sixteen Targets. Blue and red lines show performance under OTP and a single ring topology, respectively. In D), green and magenta lines show performance under the aligned and staggered double ring topologies, respectively (red, green, and magenta curves are highly overlapped). Error bars (vanishingly small, due to hundreds of thousands of simulation trials) based on a binomial distribution with 95% confidence level (see Zar (1999)). Insets show different ring topologies tested.

achieving 4-5% improvement for 140-200 neural units.[4] We discuss this difference between monkeys H and L in DISCUSSION (Comparing Results from Two Monkeys). Again, in the sixteen target case, we see that the OTP and canonical layouts perform comparably when either we have very many neural units (performances saturate), or when we have very few neural units (there is insufficient neural information, and thus many constellations will be indistinguishable in terms of performance). These findings should ideally all be confirmed with real experimental data.

6.3.2 Experimental Data Results

Having developed some expectation of the improvements offered by OTP in simulation, we wanted to verify the algorithm in real experiments, in at least some regime of the data studied in simulation. Across the the four different constellation sizes tested above, creating Fig. 6.5 and Fig. 6.6 required 100 full data sets (each with 1000 trials

[4]To put these results into the context of a few other prostheses studies, (Santhanam et al., 2006) reported recording 80-130 units in a typical session, and (Hatsopoulos et al., 2004) reported recording 32-143 units in a typical session.

Table 6.1: Decode performance in experimentally gathered neural data for canonical and OTP methods on monkey L.

	Data Set				
	L20061106	L200611113	L20061117	L20061122	Total
Neural Units (K)[1]	46	41	43	35	**NA**
Num. Targets(M)	16	16	16	16	**16**
Total OTP Reach Trials[2]	324	528	720	544	**2116**
Canonical Decode Perf.	13.0%	15.3%	14.2%	10.8%	**13.4%**
OTP Decode Perf.	15.9%	15.5%	20.6%	14.5%	**17.0%**
% Decode Improvement	2.9%	0.2%	6.5%	3.7%	**3.6%**

[1] Units include all automatic spike sort isolations used to fit the OTP constellation. This includes all units regardless of tuning strength or modulation significance.
[2] We compared equal numbers of OTP and canonical reach trials.

per condition) for each choice of neural population size. This implies tens of thousands of experimental days to replicate this result in experimental data; clearly this is infeasible. Instead, we tested this algorithm with four full day data sets in monkey L, with the goal of providing some evidence that the proposed algorithm offers improvements in a real experimental setting. Though many more experiments are needed to fully validate the simulation results, finding similar improvements in these experimental results should give confidence that, over a broader range of conditions, the method could well perform as predicted by simulation.

As described in METHODS, monkey L first performed many reaches to sixteen canonically placed targets, and a subset of these reaches were used to fit an OTP constellation. In each of the four data sets (L20061106-L20061122), we used roughly 40 neural units (sorted by our automatic spike sorter), regardless of unit quality (single unit, multi unit, or 'noise' unit (Wahnoun et al., 2006)) or tuning depth. The results for these data sets are in Table 6.1. Note that, as Table 6.1 comes from a sixteen target experiment with monkey L, Table 1 is comparable to Fig. 6.6D. Looking at Fig. 6.6D at 35-45 neural units (the x-axis), simulation suggests OTP should realize 0-2% decode improvement, and we see in Table 6.1 that we achieved 3.6%, so the results are comparable. Factors such as array lifetime and the quality of unit tuning resulted in these experimental days having low decode performance, regardless of the topology used. We see that in each data set OTP improved our decode accuracy.

We want to ask, for the data we collected, if OTP shows a statistically significant improvement over canonical placements in terms of decode performance. Using a binomial significance test with 95% confidence level (Zar, 1999), we see across all our data that indeed OTP does statistically outperform canonical placements (Table 6.1), confirming what we saw in simulation to be a meaningful improvement.[5] This improvement is captured in the final column of Table 6.1, where we see in this case that the decode performance is raised from 13.4% to 17.0% on over two thousand trials. The purpose of this data is to validate experimentally that the OTP algorithm is giving us the improvements we anticipate. Though the absolute decode accuracy is low, we nonetheless see that there is a meaningful improvement in decode accuracy. Further, we see that in no case (of the four full data sets) is there a reduction in decode accuracy. Thus, our experimental results serve to verify that OTP is indeed making good use of the neural population available to find a nontrivial improvement to decode performance.

6.4 Discussion

We have shown, for communication prosthetic systems using spiking activity, that reach target decode accuracy can be improved by optimally placing the reach targets. We have introduced this general problem, and we have created a first-of-its-kind algorithm that finds an improved target constellation by approximating an intractable problem with a tractable form. For four and eight targets, OTP layouts closely resembled canonical layouts, thus validating the canonical topology used in (Santhanam et al., 2006). Also, we realized substantial decode performance improvements in simulation for two and sixteen target configurations across a wide range of unit counts. Our experiments in real data (Table 6.1) confirm the expected improvement offered

[5]Note that doing statistical tests on the data by day (or by any subdivision) will reduce the statistical power (fewer trials) of the data and may lead to inconsistent results (some data sets significant; others not). Thus the total number of trials should be used to show that OTP does indeed outperform canonical topologies. One might also want to know whether or not, by using OTP on a given experimental day, the performance will be improved over canonical placements. Our results indicate that indeed an improvement will be seen, based on the two thousand trials we collected. Note also that, had we run this experiment on a stronger array with more units, we would expect statistically significant effects with much smaller numbers of trials, since the magnitude of the performance improvement would be greater (*cf.* Fig 6.5, panel D, at 50-150 units, or Fig 6.6, panel D, at 140-200 units).

by OTP, at least in the limited regime tested, indicating that target placement is a valuable consideration in the design of neural prostheses.

6.4.1 Intuition Gained from OTP

In Fig. 6.2 and Problem Intuition, we saw that intuition for how to place targets quickly breaks down when faced with many neurons and many targets. The OTP algorithm addresses this difficulty, and indeed it gives us reasonable solutions that outperform canonical layouts. Besides the performance improvements, is there anything to be learned from the results of this algorithm? We have noted that ring topologies have classically been chosen based on the observation that neural activity is more strongly modulated by reach direction than reach distance (Riehle and Requin, 1989; Fu et al., 1993; Moran and Schwartz, 1999a; Messier and Kalaska, 2000; Churchland et al., 2006a). If direction were essentially the only source of discriminability, a single sixteen target ring would presumably be optimal. If the opposite were true, perhaps a line of targets at various distances from the origin would be chosen. When placing sixteen targets with OTP, we typically see four to six targets on the interior, and ten to twelve on the workspace bound. The performance improvements seen in these OTP results (Fig. 6.5, Fig. 6.6, and Table 6.1) support the mixtures of tuning reported in previous studies (Riehle and Requin, 1989; Fu et al., 1993; Moran and Schwartz, 1999a; Messier and Kalaska, 2000; Churchland et al., 2006a). However, it is important to note that this observation depends on the choice of tuning model (here the cosine model of Eq. 6.1). More tuning functions should be tried (recall that OTP is general to the choice of tuning function; see Eq. 6.10) before this claim can be formalized.

6.4.2 Approximations in OTP Algorithm

Across the range of unit and target counts tested in simulation, OTP outperforms each canonical ring topology, with performance gains of up to 9%. The minimum improvement was in all cases 0%, in the case of: *i)*very many neural units, where performances saturate to 100%; or *ii)*very few neural units, where noise dominates and many different layouts are indistinguishable. We speculate that this logical simulation result would also hold in experimental data, but future experiments should confirm

more points on these performance curves. The results shown here are subject to three approximations, which we summarize here: *i)* in Eq. 6.7, we solve a minimax problem (minimizing an upper bound) instead of a total probability of error problem; *ii)* we use KL divergence as a proxy for pairwise error probability in Eq. 6.11; and *iii)* we optimize a non-convex problem in Eq. E.6 via a sequence of local quadratic approximations (SQP). Each of these approximations is necessary to put the problem in a tractable form and enables us to address this previously unanswerable question. Although the impact of these approximations has yet to be fully characterized, their use allows us to achieve performance gains (*cf.* Fig. 6.5 and 6.6) that would not otherwise be possible.

It is important also to note that, even when we bundle these approximations together (as we do in the OTP algorithm), we still get consistent improvements in decode accuracy vs. canonical target placements. It is possible in theory for OTP to underperform canonical layouts, if, for example, one of the approximations was highly inappropriate. Interestingly, OTP never underperforms canonical layouts in either the simulated data or the experimental data. We anticipate performance improvements beyond the 9% shown here, by using better approximations and improved algorithmic techniques.

6.4.3 Comparing Results from Two Monkeys

Comparing the simulation results for monkeys H and L, there is an apparent difference in the decode accuracies. We found in our experiments that monkey H had significantly better tuned delay period activity than did monkey L. Factors such as electrode array lifetime (Polikov et al., 2005), array positioning in M1/PMd (Crammond and Kalaska, 2000), and behavioral training could all contribute to these differences. The net result in monkey L is that a given number of neural units did not decode as well as in monkey H. Hence, the performance curves in monkey L saturate less quickly than in monkey H. It is encouraging nonetheless to see that OTP has similar performance effects at similar regions of the performance curves for both monkeys, regardless of the performance scaling introduced by different strengths of neural populations.

6.4.4 Comparing Simulated Results to Experimental Results

For monkey L, comparing Table 6.1 results to Fig. 6.6, panel D, one sees a difference between the predicted decode performances at given numbers of units and the actual results found in experiments. In simulated data, while the parameters of the firing rate model (Eq. 6.1) were fit to real neural data, the spike counts used to measure performance in Fig. 6.5 and 6.6 were generated from the model in Eq. 6.2 (simulated data). The performance improvements for real experimental data depend further on how well the spiking model (firing rate - Eq. 6.1 - and output distribution - Eq. 6.2) fits the neural data collected, how well the model generalizes to other target locations for which we have no neural data, and a host of other factors (spike sort instabilities, behavioral changes, etc.). These factors can reduce the performance of both the canonical and OTP topologies (*e.g.*, spiking model) or can reduce just the performance of the OTP topology (*e.g.*, generality of the firing rate model).

Regarding the spiking model approximation, in our simulation study, a unit fit with a particular tuning model behaved according to that model, and its spiking was Poisson. In real experiments, these assumptions do not hold for any target constellation. Real units can be untuned to target position and/or tuned to some other behavioral correlate; both possibilities can introduce a punitive source of noise to the decoder with a limited number of training trials. The spiking model assumptions are approximations that can only reduce performance for both topologies. This performance reduction should be equivalent across topologies, and so we focus our results on the performance differences between topologies, and not the absolute accuracies of each decoder. Using a different output distribution (*e.g.* (Barbieri et al., 2001; Truccolo et al., 2005; Cunningham et al., 2008c)) might improve decoding for both OTP and the canonical topology. Nonetheless, the simple Poisson choice allows us to readily demonstrate the improvements offered by OTP.

Our experiments also require an assumption about how different target locations modulate neural firing. The cosine tuning model in Eq. 6.1 is a simple first approach. The tractability of the OTP algorithm does not, however, rely on this specific firing rate form, so any improved model (*e.g.* (Kaufman et al., 2005)) can be seamlessly incorporated into OTP (as noted in Eq. 6.10; to be clear, this is the case with the firing rate model of Eq. 6.1, not the spiking model of Eq. 6.2). As tuning models were not the focus of this study, however, we chose a simple firing rate model to show the

improvements offered by OTP even in this case. A more accurate firing rate model, as it would improve the ability of OTP to find an optimal constellation, should only increase the OTP performance gains from a canonical topology. Our presentation of OTP is conservative in this regard.

6.4.5 Implementation Considerations

When considering implementing OTP in a neural prosthetic system, an investigator may consider some factors regarding usage mode. In our experiments with monkey L, we split our experimental days, using the first half for canonical topology reaches and the second half for OTP topology reaches. In a real system, this training time need not be spent daily. Realistically, the extent to which one records the same neural units (with the same tuning properties) dictates how often one needs to reoptimize the target constellation. We recently reported that neural tuning is stable at least during an experimental day and potentially over multiple days (Chestek et al., 2007). If this is the case, the target configuration would need only be trained infrequently and possibly during an offline period (*e.g.*, while the subject is sleeping). Anecdotally, we find that OTP fits similar constellations across adjacent days, which further supports this possibility. Furthermore, in our experiments we used neural units isolated by our automatic spike sorting algorithm, regardless of the quality of these units. We did this to focus on the difference in decode accuracy from canonical to OTP, but this choice drags down absolute decode accuracy (due to untuned 'noise' units, for example, see (Wahnoun et al., 2006)). In a real prosthetic system, better sorts and unit isolations may be made and fed into the OTP algorithm. Doing so would likely raise the decode accuracy of both canonical and OTP topologies. Again, this step may be done offline to improve overall system performance without compromising the availability of the prosthetic system. The OTP algorithm can also be run on data collected from any target constellation, so one could also iteratively run OTP on a previous OTP configuration (there is no need to revert the system to a canonical topology).

6.4.6 Future Work

As mentioned earlier in this chapter (*e.g.*, "Comparing Simulated Results to Experimental Results" above), future work should focus on extending OTP beyond the cosine tuning and Poisson spiking models of Eq. 6.1 and Eq. 6.2. Future work could also incorporate an iterative OTP algorithm that would monitor for new units appearing, old units rolling off, and units changing tuning, all the while updating the target constellation appropriately. Technology is being developed to allow this recording capability (Santhanam et al., 2007; Chestek et al., 2009). The experiments in this chapter presented targets with equal frequency, but future experiments should relax this assumption and extend OTP to handle this case. Furthermore, we have shown here an algorithm using spiking activity only. As multiple modalities (LFP, EEG, ECoG, etc.) are incorporated into a prosthetic system, OTP could be extended to place target constellations based on those sources of neural information as well.

6.5 Acknowledgements

We thank Maneesh Sahani, Gopal Santhanam, and George Gemelos for valuable technical discussions. We thank Gopal Santhanam and Afsheen Afshar for the neural data used to fit the firing rate model for monkey H. We thank Mackenzie Risch for veterinary care, Drew Haven for technical support, and Sandy Eisensee for administrative assistance.

6.6 Grants

This work was supported by the Michael Flynn Stanford Graduate Fellowship (JPC), NDSEG Fellowships (BMY & VG), Gatsby Charitable Foundation (BMY), NSF Graduate Research Fellowships (BMY & VG), NIH-NINDS-CRCNS-R01, Christopher and Dana Reeve Foundation (SIR & KVS), and the following grants to KVS: Burroughs Wellcome Fund Career Award in the Biomedical Sciences, Stanford Center for Integrated Systems, NSF Center for Neuromorphic Systems Engineering at Caltech, Office of Naval Research, Sloan Foundation, and Whitaker Foundation.

Chapter 7

Firing Rate Estimation and its Relevance to Neural Prosthetic Systems

The previous chapter investigated one particular aspect of neural prosthetic systems, and it showed that algorithmic optimization can improve our ability to decode control signals from motor cortex. This success raises the question: what other areas offer significant performance improvements, and what areas do not? In this chapter, we return to the problem of firing rate estimation and investigate the relevance of this signal processing effort to neural prosthetic systems. Numerous method for estimating neural firing rates, including the method of Chapter 2, have been developed in recent years, but to date no systematic comparison has been made between them. Here we review both classic and current firing rate estimation techniques. We compare the advantages and drawbacks of these methods. Then, in an effort to understand their relevance to the field of neural prostheses, we also apply these estimators to experimentally-gathered neural data from a prosthetic arm-reaching paradigm. Using these estimates of firing rate, we apply standard prosthetic decoding algorithms to compare the performance of the different firing rate estimators, and, perhaps surprisingly, we find minimal differences. This chapter serves as a review of available spike train smoothers and a first quantitative comparison of their performance for brain-machine interfaces. This work, which has been published as Cunningham et al. (2009), was done jointly with Vikash Gilja, Stephen Ryu, and Krishna Shenoy.

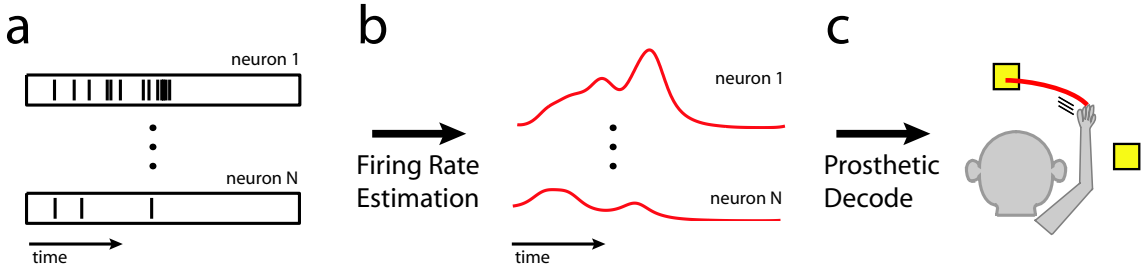


Figure 7.1: Context for firing rate estimation and neural prosthetic decode. (a) N single spike trains are gathered from N neurons on one experimental trial. (b) Those spike trains are denoised and smoothed using a firing rate estimation method. (c) Those firing rates are used by a decoding algorithm to estimate, for example, a reaching arm trajectory.

7.1 Introduction

Neuronal activity is highly variable. Even when experimental conditions are repeated closely, the same neuron may produce quite different spike trains from trial to trial. This variability may be due to both randomness in the spiking process and to differences in cognitive processing on different experimental trials. One common view is that a spike train is generated from a smooth underlying function of time (the firing rate) and that this function carries a significant portion of the neural information (vs. the precise timing of individual spikes). If this is the case, questions of neuroscientific and neural prosthetic importance may require an accurate estimate of the firing rate. Unfortunately, these estimates are complicated by the fact that spike data gives only a sparse observation of its underlying rate. Typically, researchers average across many trials to find a smooth estimate (averaging out spiking noise). However, averaging across many roughly similar trials can obscure important temporal features (Nawrot et al., 1999; Yu et al., 2006, 2009b). Trial averaging can be especially problematic in a brain-machine interface (BMI) setting, where physical behavior is not under strict experimental control, and so motor movements and their associated neural activity can vary considerably across trials. Thus, estimating the underlying rate from only one spike train is an important but challenging problem.

To address this problem, researchers have developed a number of methods for estimating continous, time-varying firing rates from neural spike trains. The goal of any firing rate estimator then is two-fold: first, the method seeks to return a smooth, continuous-time firing rate that is more amenable to analytical efforts than the spiking

neural signal. Second, as is the goal of any statistical signal processing algorithm, the firing rate estimator seeks to denoise the signal (separate the meaningful fluctuations in underlying firing rate from the noise introduced by the spiking process). This firing rate estimation step is shown in Fig. 7.1. Panel (a) shows a single spike train (one experimental trial) for each of N neural units. The spike train is shown as a train of black rasters, where each raster (vertical tick) represents the occurence of a spike at that time in the trial. The firing rate estimator seeks to process each of these noisy spike trains into smooth, continuous-time firing rates that are denoised and simpler to analyze, as shown in panel (b). Finally, in a BMI setting (our case of interest here), these firing rates may then be used by a prosthetic decoding algorithm to estimate a motor movement, as shown in Fig. 7.1, panel (c).

In this study, we review the methods that have been developed both classically and more recently, from the fields of statistics, machine learning, and computational neuroscience (see "Firing Rate Methods" below). We point to the relevant publications and give high level overviews of each method, noting a few potential strengths and weaknesses with respect to the problem of estimating firing rates from single spike trains.

Having reviewed several estimation methods, we then turn to the question of performance. To date, no comparison between these methods exists; such comparisons may assist researchers in determining what firing rate estimator is appropriate for what application. In this study, we choose the BMI application of neural prosthetic decode in an arm-reaching setting. We train a monkey to make point-to-point reaches in a 2-D workspace. Using a multi-electrode array implanted in pre-motor/motor cortex, we record spike trains from ten to fifteen neural units (we consider only high quality single units) during this reaching task. There are many prosthetic decoding algorithms that can decode the arm movement from the recorded neural activity (some papers include: Georgopoulos et al. (1986); Brown et al. (1998); Serruya et al. (2002); Taylor et al. (2002); Carmena et al. (2003); Kemere et al. (2004); Wu et al. (2004); Brockwell et al. (2004); Carmena et al. (2005); Wu et al. (2006); Hochberg et al. (2006); Yu et al. (2007); Srinivasan et al. (2007); Chestek et al. (2007); Velliste et al. (2008)). Some of these algorithms use smooth estimates of firing rates as input. Here we investigate how the performance of these decoders changes, depending on what firing rate estimation method is used. In particular, we choose the widespread

linear decoder (as recently used in Carmena et al. (2005); Chestek et al. (2007)) and the Kalman filter (as recently used in Wu et al. (2002, 2004, 2006)). We individually smooth thousands of spike trains (from many trials and many neural units) with each firing rate estimation method, and we decode arm trajectories from these firing rate estimates with the same decoding algorithms.

The purpose of this chapter then is both to review available firing rate estimators and to get some understanding of their relevance to BMI applications. This study does not attempt to address the many other important avenues for investigation in BMI or spike train signal processing. For BMI performance, these avenues include at least: prosthetic decode algorithms (Georgopoulos et al., 1986; Brown et al., 1998; Wu et al., 2004; Brockwell et al., 2004; Wu et al., 2006; Srinivasan et al., 2007; Yu et al., 2007), recording technology (Wise et al., 2004), the design of prosthetic end effectors and interfaces, be that a robotic arm or computer screen (Schwartz, 2004; Velliste et al., 2008; Cunningham et al., 2008b), and multiple signal modalities (*e.g.*, EEG, ECoG, LFP, and spiking activity) (Mehring et al., 2003). Two reviews in particular give a thorough overview of these and other important areas of BMI investigation (Lebedev and Nicolelis, 2006; Schwartz, 2004). For spike train signal processing, there are also many avenues of research not addressed in this study, including at least: spike-sorting (Lewicki, 1998), information-theoretic studies (Borst and Theunissen, 1999; Nirenberg et al., 2001), neural correlations (Shlens et al., 2006; Pillow et al., 2008), methods for multiple simultaneously recorded neurons (Chapin, 2004; Churchland et al., 2007; Yu et al., 2009b), and more accurate spiking models (Johnson, 1996; Barbieri et al., 2001; Ventura et al., 2002; Kass and Ventura, 2003; Truccolo et al., 2005; Koyama and Kass, 2008). Two reviews in particular discuss these and other issues in spike train processing (Brown et al., 2004; Kass et al., 2005).

Linking methodological developments to observable physical behavior (such as neural prosthetic decode performance) is critical for increasing the adoption and usefulness of these methods. This study takes an important first step in that direction for the problem of firing rate estimation.

7.2 Firing Rate Methods

This section reviews several popular and current firing rate estimation methods. We introduce each method at a high level, point to relevant publications, and suggest potential advantages and disadvantages of each. We then summarize the reviewed methods and discuss related methods and other possibilities that are not yet included in literature.

7.2.1 Kernel Smoothing (KS)

The most common historical approach to the problem of estimating firing rates has been to collect spikes from multiple trials in a time-binned histogram known as a *peri-stimulus-time histogram* (PSTH), which produces a piecewise constant estimate. To achieve a smooth, continuous firing rate estimate, as is often of interest in single trial settings (such as neural prostheses), researchers instead typically use kernel smoothing (KS); that is, they convolve the spike train with a kernel of a particular shape (*e.g.*, Nawrot et al. (1999)). This convolution produces an estimate where the firing rate at any time is a weighted average of the nearby spikes (the weights being determined by the kernel). A Gaussian shaped kernel is most often used (see, *e.g.*, Kass et al. (2005)), and this kernel serves to smooth the spike data to a firing rate that is higher in regions of spikes, lower otherwise. However, The kernel shape and time scale (*e.g.*, the standard deviation of the Gaussian) are frequently chosen in an *ad hoc* way, which largely alters the frequency content of the resulting estimate (in other words, how quickly firing rate can change, and how susceptible the estimate is to noise).

The most obvious advantage of kernel smoothing is its simplicity. KS methods are extremely fast and simple to implement, which has led to wide adoption. In this study, we implement three Gaussian kernel smoothers of various bandwidths (which determine smoothness): 50ms standard deviation (KS50), 100ms (KS100), and 150ms (KS150). These are common choices for single trial studies, and they produce significantly different estimates of firing rate. This *ad hoc* choice of smoothness is typically considered a major disadvantage of KS methods.

7.2.2 Adaptive Kernel Smoothing (KSA)

In Richmond et al. (1990), the authors address two concerns with standard KS: first, the *ad hoc* smoothness choice as noted above, and second, the fact that the kernel width can not adapt at different regions of smoothness in the firing rate. We call this fixed frequency behavior *stationarity*. KSA incorporates a nonstationary kernel to allow the spike train to determine the extent of firing rate smoothness at various points throughout the trial. It does so by forming first a stationary firing rate estimate (called a pilot estimate), and from that pilot, it forms a set of local kernel widths at the spike events. These local kernels are then used to produce a smoothed firing rate that changes more rapidly in regions of high firing, and less in regions of less firing. This trend is sensible, as regions of little spiking give fewer observations into the firing rate process underlying the data.

KSA benefits from the simplicity of KS methods, and the added complexity of the local kernel widths increases the computational effort only very slightly. Further, this approach lifts the strict stationarity requirement of many methods. A possible shortcoming is that, even though it adapts the kernel width, KSA still requires an *ad hoc* choice of kernel width for the pilot estimate.

7.2.3 Kernel Bandwidth Optimization (KBO)

In KS methods, as latter sections in this paper will show, the *ad hoc* choice of smoothness can have a significant impact on the firing rate estimate. KBO seeks to remove this shortcoming of kernel smoothing by establishing a principled approach to choosing the kernel bandwidth. In Shimazaki and Shinomoto (2007b), a method is developed for automatically choosing the bin width of a PSTH. By assuming that neural spike trains are generated from an inhomogeneous Poisson process (*i.e.*, a Poisson process with time-varying firing rate), the authors show that the mean squared error (MSE) between the PSTH and the true underlying firing rate can be minimized using only the mean rate (rate averaged across time), *without* knowledge of the true underlying firing rate.

In Shimazaki and Shinomoto (2007a), this PSTH method is adapted to similarly optimize the bandwidth of a smoothing kernel. The authors of that report provide a

simple algorithm for the popular Gaussian kernel, which we implemented for the purposes of this study. Once the optimal kernel bandwidth is chosen with the algorithm of Shimazaki and Shinomoto (2007a), we then perform standard kernel smoothing (as defined in KS above) with the optimized kernel bandwidth. We refer to this method as KBO.

We also note here a method quite similar in spirit to KBO. In Nawrot et al. (1999), a heuristic method is developed to find the optimal bandwidth of a kernel smoother. We also implemented this method and found that, with the particular motor cortical data of interest for this BMI study, the method of Nawrot et al. (1999) produced very often a flat, uninformative firing rate function (*i.e.*, a very large kernel bandwidth). Accordingly, we chose the newer, principled method of Shimazaki and Shinomoto (2007a,b) (which produces a range of different kernel bandwidths, depending on the spike data) to demonstrate the performance of kernel bandwidth optimization methods.

KBO has the advantage of simple implementation and correspondingly very fast run time (only slightly longer than a regular kernel smoother, due to the overhead required to calculate the optimal bandwidth). Shortcomings of this approach may include the Poisson spiking assumption (required for this method), as much research has shown that neural spiking often deviates significantly from Poisson spiking statistics (see, *e.g.*, Barbieri et al. (2001); Miura et al. (2007); Paninski et al. (2004a)).

7.2.4 Gaussian Process Firing Rates (GPFR)

All kernel smoothing methods, including KS, KSA, and KBO as above, act as low pass filters to produce a smooth, time-varying firing rate. Alternatively, several methods take a probabilistic approach. If one assumes a prior probability distribution for firing rate functions (*e.g.*, some class of smooth functions), and a probability model describing how spikes are generated, given the underlying firing rate (*e.g.*, an inhomogeneous Poisson process (Daley and Vere-Jones, 2002)), one can then use Bayes rule (Papoulis and Pillai, 2002) to infer the most likely (or expected) underlying firing rate function, given an observation of one or multiple spike trains. The following methods - GPFR, BARS, and BB - are variations on this general approach.

In Cunningham et al. (2008c), firing rates are assumed *a priori* to be draws from a Gaussian process. Gaussian processes place a probability distribution on firing

rate functions which allow all functions to be possible, but strongly favor smooth functions (Rasmussen and Williams, 2006). This study then assumes that, given the firing rate function, spike trains are generated according to an inhomogeneous Gamma interval process, which is a generalization of the familiar Poisson process to allow spike history effects such as neuronal refractory periods. Bayesian model selection and Bayes' rule are then used to infer the most likely underlying firing rate function, given an observation of one or multiple spike trains. Owing to this probabilistic model, the computational overhead of such a firing rate estimator can be significant, so the authors developed numerical methods to alleviate these challenges (Cunningham et al., 2008a).

GPFR has the advantage of using a probabilistic model, which allows automatic smoothness detection (in contrast to the *ad hoc* smoothness choices made in, for example, KS), and which naturally produces error bars on its predictions (which may be useful for data analysis purposes). GPFR also has the benefit of being able to readily incorporate different *a priori* assumptions about firing rate (such as known, stimulus-driven nonstationarities in the firing rate, which can be controlled through the Gaussian process prior). Even with the significant computational improvements developed in Cunningham et al. (2008a), GPFR still requires seconds of computational resource (for spike trains roughly one second in length), which may be a disadvantage compared to kernel smoothers (which work in tens to hundreds of milliseconds).

7.2.5 Bayesian Adaptive Regression Splines (BARS)

Instead of a Gaussian Process prior on smooth firing rate functions, BARS, as introduced and used in DiMatteo et al. (2001); Kass et al. (2005); Kaufman et al. (2005); Behseta and Kass (2005), models underlying firing rate with a spline basis. Splines generally are piecewise polynomial functions that are connected at time points called "knots." In DiMatteo et al. (2001), the authors choose a prior distribution on the number of knots, the position of the knots, and other parameters of the spline function. Conditioned on firing rate, BARS then assumes that spikes are generated according to a Poisson spiking process.

This model choice allows Bayesian inference to be carried out. Owing to the forms of the probability distributions chosen, approximate inference methods must be used

(an analytical solution is intractable). BARS uses the well established techniques of reversible-jump Markov chain Monte Carlo and Bayesian information criteria to estimate the underlying firing rate (which in this case is the mean of the approximate posterior distribution, given the observed data). BARS is fully described in DiMatteo et al. (2001), and further applications and explanations can be found in Olson et al. (2000); Kass et al. (2005); Kaufman et al. (2005); Behseta and Kass (2005). This study uses the MATLAB implementation of BARS available at the time of publication at `http://lib.stat.cmu.edu/~kass/bars/bars.html`

One major advantage of BARS is that the spline basis allows different regions of firing rate to change more or less smoothly, which allows high frequency changes in rate while still removing high frequency noise (this is not possible in traditional kernel smoothers). Further, like other probabilistic methods, BARS produces an approximate posterior distribution on firing rates, so valuable features like error bars are available. BARS, like GPFR, suffers from technical complexity that translates into meaningful computational effort and run-time, compared to more basic kernel smoothers.

7.2.6 Bayesian Binning (BB)

Instead of assuming a continuous, time-varying firing rate as in many of the above approaches, the authors of Endres et al. (2008) assume neural firing rates can be modeled *a priori* by piecewise constant regions of varying width (in contrast to a fixed-width binning scheme like the classic PSTH). This BB approach, like BARS and GPFR, constructs a probabilistic model for spiking, where both the firing rates in piecewise constant regions and the boundaries between the regions themselves have associated probability distributions (together, the boundaries and the firing rates at each interval fully specify a firing rate function). BB then assumes an inhomogeneous Bernoulli process for spiking (*i.e.*, each time point contains 0 or 1 spikes), given the underlying firing rate.

With these assumptions made, Bayes rule is then used to infer the underlying firing rate from the above model. Importantly, because the boundaries and height of the firing rate bins are probabilistic, the result of this firing rate inference is a smooth, time-varying firing rate, and BB is thus comparable to the other methods highlighted in this study. The BB method is fully described in Endres et al. (2008),

Table 7.1: Summary of reviewed firing rate methods.

	Method					
	KS	KSA	KBO	GPFR	BARS	BB
Automatic Smoothness Detect.	N	N	Y	Y	Y	Y
Probabilistic Model	N	N	N	Y	Y	Y
Runtime/Comp. Complexity	millisec.	millisec.	millisec.	seconds	seconds	minutes
Nonstationary Smoothness	N	Y	N	N	Y	Y
Spike History Effects	N	N	N	Y	N	N

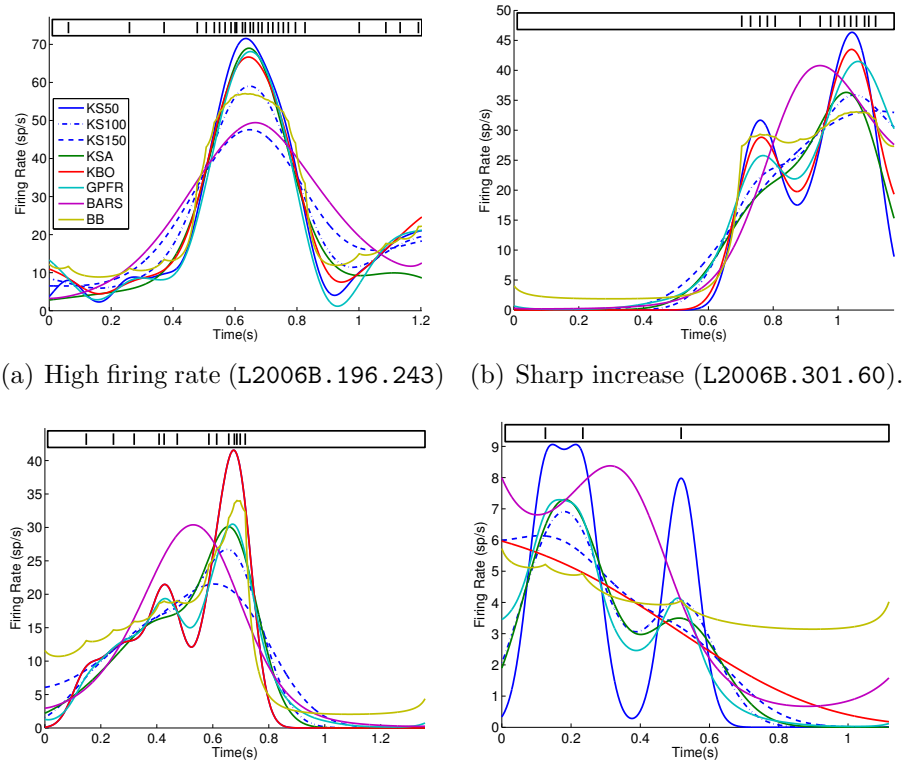
and we implemented the algorithm using the authors' source code, which is available at the time of publication at `http://mloss.org/software/view/67/`.

Like GPFR and BARS, BB has the advantage of being a fully probabilistic model, which allows automatic smoothness detection (in contrast to the *ad hoc* smoothness choices made in, for example, KS), and which produces error bars on its predictions. Also, like BARS and KSA (and unlinke GPFR, KBO, and KS), BB is a nonstationary smoothing model, so it can adapt its smoothness to regions of faster or slower firing rate changes. However, as BB constructs a thorough probabilistic model for spiking and solves it exactly, the method requires significant computational resource (generally an order of magnitude more than BARS and GPFR, the other computationally expensive methods), which may limit the use of BB in some applications.

7.2.7 Summary of Reviewed Methods

These methods were chosen in that they all can be used as single trial, single neuron firing rate estimators (as is relevant for neural prosthetic applications). In Fig. 7.2, we show four examples of firing rates inferred by all eight methods reviewed above. Each panel represents a different spike train, which is denoted above the firing rates as a train of black rasters (as in Fig. 7.1). These four panels show a range of spiking patterns, including: (a) high firing, (b) sharply increasing activity, (c) sharply decreasing activity, and (d) low firing. Though there are infinite possible firing rate patterns, these four example spike trains illustrate the wide range of firing rates profiles that can be estimated from the same neural activity, depending on the estimation method used.

The methods above also demonstrate a range of approaches and features that one might consider in designing a firing rate estimator. Table 7.1 compares the above



(a) High firing rate (`L2006B.196.243`) (b) Sharp increase (`L2006B.301.60`).

(c) Sharp decrease (`L2006B.170.92`). (d) Low firing rate (`L2006B.170.216`).

Figure 7.2: Example of various firing rate methods applied to data from different neurons and different trials. Each method (see legend) produces a smooth estimate of underlying firing rate from each of the four separate spike trains. The spike trains are represented as a train of black rasters above each panel. Note that KBO obscurs KS50 in panel (c).

methods in terms of five important features, where we indicate generally desireable features in green and undesireable features in red. The first row notes which methods offer principled, automatic determination of the firing rate smoothness (vs. choosing a kernel bandwidth in an *ad hoc* way). The second row indicates if the method is a proper probalistic model, which carries advantages previously discussed. The drawback of probabilistic models lies in their computational complexity (and, as a result, run time); the third row of Table 7.1 details ballpark run-time requirements for estimating one firing rate function from one single spike train. The fourth row details which methods are nonstationary; that is, which methods can adapt the smoothness of the estimate at different points in the spike train. Finally, we also noted above that spike trains are known to depart significantly from Poisson statistics (*e.g.*, refractory

periods); the fifth row illustrates which methods are Poisson-based and which are not.

It is important to note that all of these methods can also be used for multiple-trial firing rate analyses. Some methods, including BARS and BB, were introduced more with a multi-trial motivation than a single-trial motivation. This study makes no claim on the effectiveness of any of these methods at larger numbers of trials, as such a circumstance is not germane to BMI applications. Thus, the forthcoming results should not be viewed as a statement about the quality of a particular firing rate estimator *in general*, but rather for the single-trial analyses that are relevant in BMI studies.

7.2.8 Other Related Methods

Despite the range of methods already discussed, the above list of recent and classic firing rate estimators is by no means exhaustive. We here discuss a few other possibilities and avenues of investigation not covered by the above methods.

First, we note that none of the above methods are implemented as cross-validation schemes (Bishop, 2006). The probabilistic models (GPFR, BARS, BB) all do Bayesian model selection to adapt their smoothness. KBO uses an MSE criterion and KSA uses a criteria based on the amount of local spiking to adapt their smoothness, whereas KS uses only a user-defined kernel width choice. Another possibility is to cross-validate, where other trials of data are used to inform the parameter (*e.g.*, smoothness) choices when estimating firing rate on a novel spike train (Bowman (1984) reports on the related topic of probability density estimation). For example, one might believe that all firing rates in a particular BMI application evolve with roughly equal smoothness. Even though the firing rates may be quite different trial to trial, one could cross-validate with some criterion (such as decode performance) to choose the smoothness for the firing rate estimation on the new spike train in question. This report does not review that possibility, as we wish to focus on methods that produce firing rates from spike trains based on *only* those spike trains (not a validation set). Further, many, if not all, of the above methods could incorporate a cross-validation scheme: for example, GPFR, BARS, and BB could choose their parameters via cross-validation instead of Bayesian model selection. Thus, cross-validation is a feature of model selection more than it is of the firing rate method used, and we chose to focus on the methods as previously published.

Second, we also note that the methods outlined above are all *unsupervised*, in that they infer firing rates without knowledge of an extrinsic covariate such as the path of a rat foraging in a maze, or the kinematic parameters of a moving arm. Instead, if one has a good idea about how some measureable behavior translates to firing rate, one might assume a parametric form for firing rate based on behavior, learn the parameters from the data, and use that model to infer time-varying firing rate. Some studies using this approach include Brown et al. (1998); Barbieri et al. (2001); Brown et al. (2002); Ventura et al. (2002); Eden et al. (2004); Truccolo et al. (2005); Stark et al. (2006); Pillow et al. (2008). These approaches are specific to particular neural areas, particular experimental setups, and they are susceptible to biases of their own. Thus, we chose not to review these techniques to again focus on methods that produce firing rates for a given spike train, using that spike train alone.

Third, we note that, although the methods described above are quite specific, there are many areas in which they can be extended or combined with other approaches. Simple first examples include: KBO and KSA could be combined in a two-stage method, or the method of Miura et al. (2007) could replace a part of the model selection method in GPFR. As a more interesting example, one advantage of probabilistic models (including BARS, GPFR, and BB) is that they can readily be extended to different spiking probability models. One spiking model, the so-called generalized linear model (GLM), has received much attention of late (Eden et al., 2004; Barbieri et al., 2001; Truccolo et al., 2005; Srinivasan et al., 2007; Coleman and Sarma, 2007; Czanner et al., 2008; Koyama and Kass, 2008; Pillow et al., 2008) for its ability to model neural spiking quite well and its flexibility in being extended to many different problem domains. This GLM spiking model may inform firing rate estimation as well.

Finally, we note that all the methods above are single-neuron firing rate estimators that are independent of the activity of other neurons. Firing rate estimation methods that consider multiple units (as is often collected with electrode arrays in BMI experiments) may be able to leverage the simultaneity of recordings to improve the quality of firing rate estimates. Some work has begun to investigate this general question, including (Chapin, 2004; Brown et al., 2004; Churchland et al., 2007; Pillow et al., 2008; Yu et al., 2009b) (and the GLM model of Czanner et al. (2008) could also

be readily extended for this purpose). However, none of these multi-dimensional approaches specifically addresses unsupervised firing rate estimation as do the methods of this report, so we will leave multidimensional extensions to future work.

In summary, the problem of firing rate estimation (and, more generally, inferring meaningful information from spiking data) is quite broad. The methods reviewed in this report are all directly comparable, but there are many opportunities for extensions and adaptations of these models.

7.3 Paradigm for Evaluating Firing Rate Methods

Having reviewed several firing rate estimators, we now investigate their relevance for neural prosthetic applications. We first describe the experimental setting we employed to study this question ("Reach Task and Neural Recordings" below). We then describe two popular prosthetic decoding algorithms ("Decoding Algorithms" below) and performance metrics ("Calculating Decode Performance" below) that we can use to evaluate the quality of our firing rate estimation.

7.3.1 Reach Task and Neural Recordings

Animal protocols were approved by the Stanford University Institutional Animal Care and Use Committee. We trained an adult male monkey (*Macaca mulatta*) to perform point-to-point reaches on a 5-by-5 grid (25 targets) for juice rewards. Visual targets were back-projected onto a fronto-parallel screen 30 cm in front of the monkey. The monkey began each trial with his hand held at a particular target, which must be held for a random time interval. These hold times were exponentially distributed with a mean of 300 ms (but shifted to be no less than 150 ms). This exponential distribution prevented the monkey from preempting the movement cue. After the hold time, a pseudo-randomly chosen target was presented at one of the target locations. The 25 targets were spaced evenly on an 8 cm by 8 cm grid. Concurrent with the target presentation, the current hold point disappeared, cueing the monkey to reach to the target (the "go cue"). The monkey was motivated to move quickly by a reaction time constraint (maximum allowable reaction time of 425 ms, minimum of 150 ms, again to prevent preemption). The monkey reached to the target and then held the target for 300 ms, after which the monkey received a liquid reward. The next trial

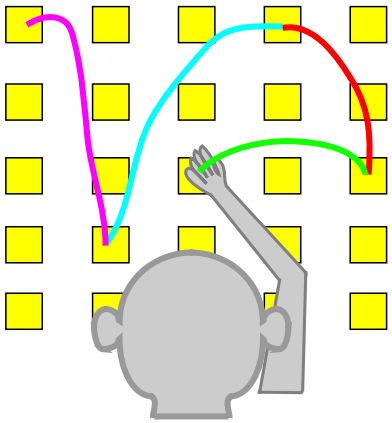


Figure 7.3: Cartoon of the reaching task as in `L2006A` and `L2006B`. Four sample trials are shown (one each in magenta, cyan, red, and green).

started immediately after the successful hold period. In total, all trials are 850 to 1500 ms long (these times vary depending on the length and speed of the reach and the randomized hold time). Fig. 7.3 illustrates four sequential trials of the reaching task.

During experiments, the monkey sat in a custom chair (Crist Instruments, Hagerstown, MD) with the head braced. The presentation of the visual targets was controlled using the Tempo software package (Reflective Computing, St. Louis, MO). A custom photo-detector recorded the timing of the video frames with 5 ms resolution. The position of the hand was measured in three dimensions using the Polaris optical tracking system (Northern Digital, Waterloo, Ontario, Canada; 60 Hz, 0.35 mm accuracy), whereby a passive marker taped to the monkey's fingertip reflected infrared light back to the position sensor. Eye position was tracked using an overhead infrared camera (Iscan, Burlington, MA; 240 Hz, estimated accuracy of 1°).

A 96-channel silicon electrode array (Cyberkinetics, Foxborough, MA) was implanted straddling dorsal pre-motor (PMd) and motor (M1) cortex (left hemisphere), as estimated visually from local landmarks, contralateral to the reaching arm. Surgical procedures have been described previously (Churchland et al., 2006b; Santhanam et al., 2006; Hatsopoulos et al., 2004). Spike sorting was performed offline using techniques described in detail elsewhere (Sahani, 1999; Santhanam et al., 2004; Zumsteg et al., 2005). Briefly, neural signals were monitored on each channel during a two minute period at the start of each recording session while the monkey performed the behavioral task. Data were high-pass filtered, and a threshold level of three times

the RMS voltage was established for each channel. The portions of the signals that did not exceed threshold were used to characterize the noise on each channel. During experiments, snippets of the voltage waveform containing threshold crossings (0.3 ms pre-crossing to 1.3 ms post-crossing) were saved with 30 kHz sampling. After each experiment, the snippets were clustered as follows. First, they were noise-whitened using the noise estimate made at the start of the experiment. Second, the snippets were trough-aligned and projected into a four-dimensional space using a modified principal components analysis. Next, unsupervised techniques determined the optimal number and locations of the clusters in the principal components space. We then visually inspected each cluster, along with the distribution of waveforms assigned to it, and assigned a score based on how well-separated it was from the other clusters. This score determined whether a cluster was labeled a single-neuron unit or a multi-neuron unit. For this report, as many firing rate methods are based on biophysical properties of single neurons, we use units labelled only as high quality, single-neuron units.

The monkey (monkey L) was trained over several months, and multiple data sets of the same behavioral task were collected. We chose two such data sets to evaluate prosthetic decode (`L2006A and L2006B`), from which we took 14 and 15 high quality, single-neuron units, respectively (note that more units would be available were we to consider "possible single units" or multi-units, as is often done in prosthesis studies). For the purposes of this study, we selected the first 300 successful trials (about five minutes of neural activity and physical behavior), which is ample for fitting the decoding models used here. Thus, we use two data sets, each with 14 or 15 neural units and 300 experimental trials. This produces a total of 8700 spike trains that were all analyzed by each of the 8 firing rate methods (and subsequently by the two decoding algorithms). Across all these firing rate estimations and their subsequent prosthetic decodes, this analysis required roughly four weeks of fully dedicated processor time on five to ten 2006-era workstations (Linux Fedora Core 4 with 64 bit, 2.2-2.4GHz AMD processors and 2-4GB of RAM) running MATLAB.

7.3.2 Decoding Algorithms

Having detailed the experimental collection of neural spike trains and physical behavior, and having reviewed methods for processing spike trains into firing rates, we now

address how to decode arm trajectories from neural firing rates. As with the firing rate methods above, we discuss the methods at a high level and point to the relevant literature which offers more methodological description.

Linear Decode

The Linear Decode algorithm, as used for example in Carmena et al. (2005); Hochberg et al. (2006); Chestek et al. (2007), is a simple first approach to decoding arm trajectories from neural activity. This algorithm assumes the physical behavior at a particular time t is a linear combination of all recorded neural activity (across all N recorded neural units) that preceeds t by some amount of time. We chose to consider the preceeding 300ms of neural activity[1]. This period of neural activity can be considered a row in a matrix of firing rates (as many rows as time points in the experimental trials). If each dimension of the behavior (*e.g.*, horizontal hand position and vertical hand position) is a vector of length also equal to the number of time points, then simple least squares can solve for the linear weights that relate neural activity to physical behavior. These weights can then be applied to novel neural activity to produce a decoded reach trajectory, which hopefully matches the true reach well. More mathematical details can be found in, *e.g.*, Chestek et al. (2007).

For completeness, we note here a few specifics of our implementation of this algorithm. To provide the algorithm with a finely time-resolved firing rate, we sampled the firing rate estimates (from all firing rate methods) every 5ms. We found that increasing this sampling rate did little more than increase the computational burden of the decode, and reducing this rate ignored features of the firing rate estimates, which would be detrimental to our comparison of methods. Further, because of the 300ms integration window and the trial structure of the data (there is a time break in between each trial), for the decode analysis, we decode only the length of the trial beginning 300 ms after the beginning of the trial (this prevents the linear decode filter from going into a region of undefined neural activity). Owing to the random hold time and the reaction time of the monkey (both enforced to be no less than 150ms,

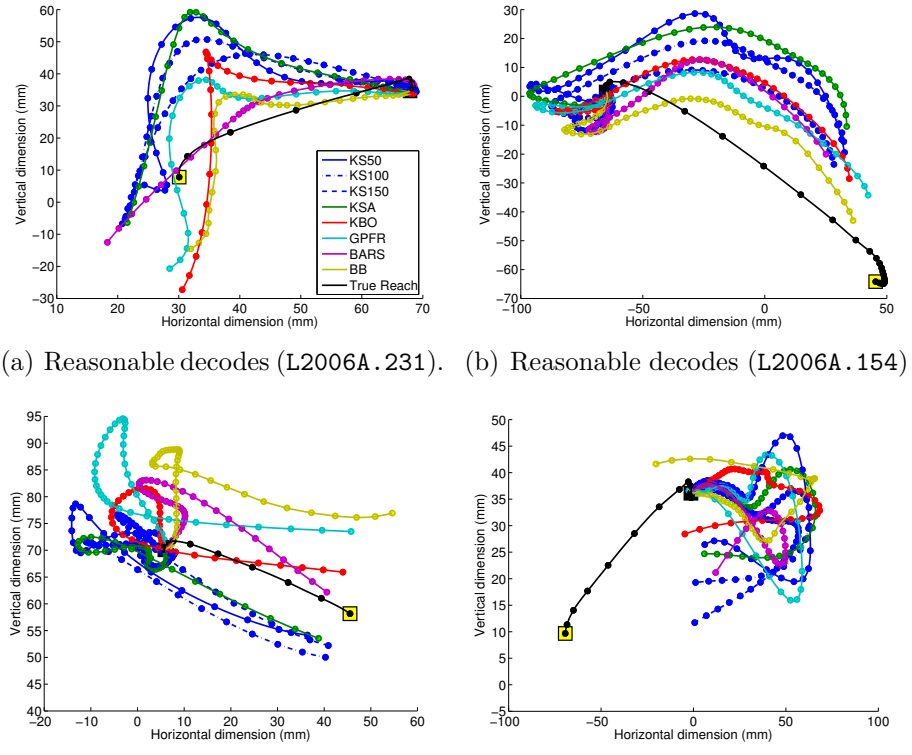
[1] We chose 300ms as a number on par with the timescale of arm movements and motor processing. Ideally, one might run this analysis at a variety of temporal window sizes. However, we note that this choice has no discernable bias in favor of any particular firing rate estimation method. We also found that using 300ms produced decode results of similar quality to using longer periods. Finally, we note that the Kalman Filter does not make this assumption, providing yet another cross-check.

see "Reach Task and Neural Recordings" above), there was no movement for the first 300ms of the trial, so this step is reasonable. Furthermore, we found that including this portion of the trials did not change the result considerably.

Kalman Filter

To employ the popular Kalman Filter (Kalman, 1960), we assume that the arm state (in this case, horizontal and vertical position and velocity) evolves as a linear dynamical system: the arm state at discrete time t is a linear transformation of the arm state at time $t-1$, plus Gaussian noise. We also assume a linear relationship between arm state and neural activity at that time t (again, plus noise). With this done, the Kalman Filter allows the inference of the hand state from the observation of neural data only. Starting from arm state at the beginning of the trial, the Kalman filter proceeds iteratively through time, updating its estimates of arm state and error covariance at every time step t, before and after the inclusion of neural data at that time step. These steps are entirely based on mathematical properties of the Gaussian, and the algorithm is fast and stable. Importantly, the Kalman Filter has been previously and successfully used as a BMI decoding algorithm, and more explanation and mathematical detail can be found in Wu et al. (2002, 2004, 2006).

As above, we note here a few implementation specifics. To parallel with the Linear Decode, we also sampled firing rates at 5ms intervals when fitting the Kalman Filter model and when estimating reach trajectories from it. In the Linear Decode, we chose to remove the first 300ms of the trial, during which the monkey did not move. In the Kalman Filter decodes, we truncated 300ms from the end of the trials. Choosing this slightly different time interval allows us to look across the Linear Decode and the Kalman Filter and rule out any potential idiosyncracies with the starting and ending of a trial. We also varied this choice and found that it had no effect on the relative decode performance of the different firing rate methods. Next, we note that we included horizontal and vertical position and velocity in our arm state. Acceleration is sometimes included, but the inclusion of this data in our Kalman Filter had little effect on the decode quality, so we chose not to consider it further. Finally, we note that we did not impose a temporal lag (or a group of lags) between neural data and physical behavior. Our testing with different lags produced minor differences that agreed generally with the results of Wu et al. (2006). As this aspect did not influence



(a) Reasonable decodes (`L2006A.231`). (b) Reasonable decodes (`L2006A.154`)

(c) Better decodes (`L2006A.69`). (d) Failed decodes (`L2006A.169`).

Figure 7.4: Example of decoded arm trajectories derived from different firing rate estimates of the same neural data (see legend). All data shown are decoded using a Kalman Filter and the data set `L2006A`. In all cases the true reach is shown in black (moving from black square hold point to the yellow square target). To give an idea of the velocity profile of the true reach and decoded trajectories, marks are placed on each trajectory at 20ms intervals.

the comparisons between firing rate estimators, we do not report further on it.

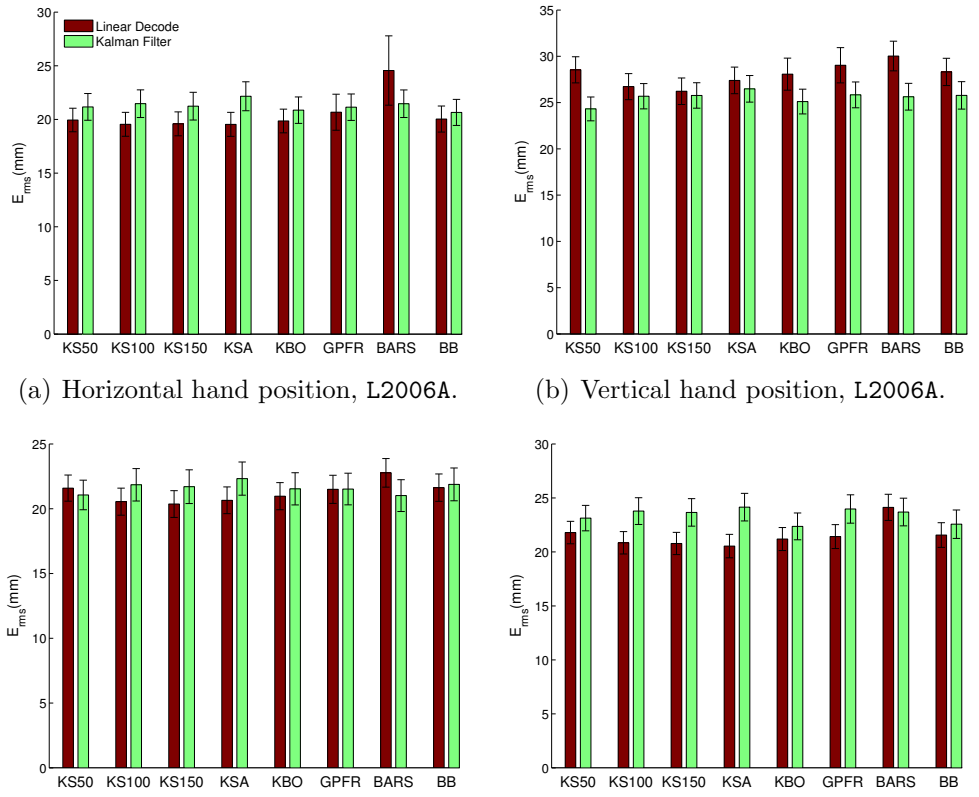
To provide a sample of these decodes, we show in Fig. 7.4 four decoded trials from `L2006A` that use the Kalman Filter. Each panel shows the true reach as a black trace moving from the black square hold point to the yellow square target. Trajectories decoded with each firing rate method (but *the same* neural data) are shown in colors corresponding to those in Fig. 7.2 (see legend). Marks are placed on each trajectory at 20ms intervals to give an idea of decoded velocity profiles. Panels (a) and (b) show reasonably average decodes (in terms of the RMS error, see the panel captions). Panel (c), a trial which decodes rather well, shows the wide variety of decoded trajectories that can arise from different firing rate estimations (but the same spike trains). Finally, Fig. 7.4, panel (d), shows that indeed the Kalman

Filter, like the Linear Decode (not shown) does sometimes fail entirely to decode the true reach, regardless of the firing rate method used. In the following sections, we generalize these specific examples, calculating performance metrics across all trials, decode methods, and data sets.

7.3.3 Calculating Decode Performance

Given any decoded arm trajectory, there are a number of possible metrics to evaluate accuracy. We use two of the most common metrics: root-mean-square error (RMSE) and correlation coefficient. For any given firing rate method, RMSE on each trial is the square root of the mean of the squared errors (across time) between the true arm trajectory and the decoded trajectory. RMSE is likely the most often-used performance metric; some examples of its use (or MSE, which is simply RMSE squared) include: Serruya et al. (2002); Brockwell et al. (2004); Kemere et al. (2004); Wu et al. (2006); Yu et al. (2007); Srinivasan et al. (2007). Correlation coefficient (ρ or r^2) is another commonly used performance metric that reflects how well the decoded trajectory matches the true arm trajectory. Considering each time step as a draw from a random variable, this metric correlates the true and decoded trajectories across time to calculate how well one trajectory predicts the other ($\rho = 1$ implies perfect linear correlation). Some previous literature using correlation coefficients to evaluate decode performance includes (Wu et al., 2002; Carmena et al., 2005; Wu et al., 2006; Chestek et al., 2007).

To calculate these performance metrics, we use leave-one-out cross validation (LOOCV) (Bishop, 2006). That is, for each data set, we select one experimental trial (one arm trajectory) to test, and we exclude both that trial's neural activity and physical behavior. We then train a decoder model based on the other 299 trials collected in that data set (`L2006A` or `L2006B`). We can then use the decode algorithm (Linear Decode or Kalman Filter) to decode the arm trajectory on the excluded trial, using only the neural activity from that trial. We repeat this same procedure 300 times (once per trial), which provides 300 decoded trials. We calculate the RMSE for each trial, and then we can average these and produce 95% confidence intervals (Zar, 1999). We also correlate all the decoded arm trajectories with the true trajectories, producing one overall correlation coefficient ρ and 95% confidence intervals on the estimate of this metric (see Zar (1999), section 19.3 for details on calculating



(a) Horizontal hand position, `L2006A`.

(b) Vertical hand position, `L2006A`.

(c) Horizontal hand position, `L2006B`.

(d) Vertical hand position, `L2006B`.

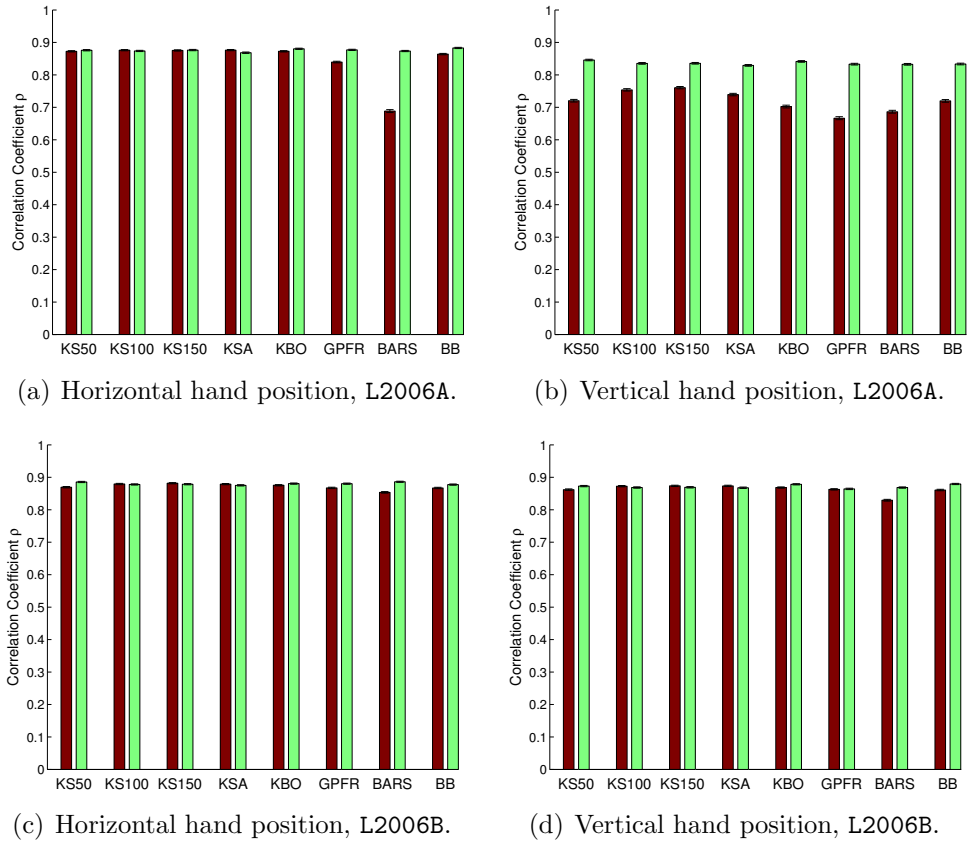
Figure 7.5: The decode performance of spike trains smoothed with different firing rate methods. Error is root mean squared error (RMSE). In all panels, red bars are decode performance with a Linear Decode; green bars are performance numbers with a Kalman Filter. Error bars indicate the 95% confidence interval.

confidence intervals for a population correlation coefficient).

7.4 Performance Results

In "Firing Rate Methods" above, we described a host of methods that estimate firing rates from experimentally gathered spike trains. We then used these firing rates to decode arm trajectories using two different decoding algorithms (above in "Decoding Algorithms") and two different performance measures (above in "Calculating Decode Performance"). We now compare different firing rate estimation methods in terms of their decode performance.

In Fig. 7.5 and Fig. 7.6, we show the RMSE and correlation coefficient results



(a) Horizontal hand position, `L2006A`.

(b) Vertical hand position, `L2006A`.

(c) Horizontal hand position, `L2006B`.

(d) Vertical hand position, `L2006B`.

Figure 7.6: The decoder performance of spike trains smoothed with different firing rate methods. Vertical axis is correlation coefficient with the true reach. In all panels, red bars are decode performance with a linear filter; green bars are performance numbers with a Kalman Filter. Error bars (vanishingly small) indicate the 95% confidence interval on the estimate of the correlation coefficient (see Zar (1999)).

(respectively) from several different decoding scenarios. Each panel shows the decode performance across all eight of the reviewed firing rate methods (KS50, KS100, KS150, KSA, KBO, GPFR, BARS, and BB). Within each panel, red bars represent the decode error using the Linear Decode method, and green bars represent the decode error using the Kalman Filter method. Panels (a) and (b) show decoding results from data set `L2006A`, and panels (c) and (d) show results from data set `L2006B`. Also, the left panels (a and c) and the right panels (b and d) show the results from decoding horizontal and vertical hand position, respectively. Thus, each firing rate estimate has sixteen performance metrics (two decode methods, two data sets, horizontal and vertical dimensions, RMSE and correlation coefficient). This variety is important to ensure that any effects are robust across data sets and decode algorithms and different

strengths of neural tuning.

First, we note several important cross-checks with existing literature. The RMSE and correlation coefficient numbers match well to the results of, for example, Wu et al. (2002, 2006); Yu et al. (2007); Chestek et al. (2007). The errors are in some cases higher than those seen in previous literature, which may be due to the complexity of this task (vs. a simpler, center-out task as in Yu et al. (2007)) or the restrictive choice of using only single neural units (rather than the many more multi-units which are often informative in a decode setting). Indeed, when we altered the number of neural units, the absolute decode performance changed as expected, but the relative differences between the decode results (from the various firing rate methods) did not. Accordingly, we are satisfied that the selected neural populations are representative. Specifically to the Linear Decode, our performance may also be different in that we used only 300 ms of preceeding neural data vs. prior literature which has used, for example, 1000 ms (Chestek et al., 2007) or 550 ms (Wu et al., 2002). Specifically to the Kalman Filter, as noted above, our performance may also be different in that we did not impose a temporal lag between neural data and physical behavior. Again, we tested changing the temporal lags and found relative performance between firing rate methods insensitive to this choice, so we are satisfied that this choice is also representative. We also visually compared trajectories decoded in this study (*e.g.*, Fig. 7.4) to decoded trajectories from Wu et al. (2002, 2006), and we found these similar, giving confidence that we are successfully reproducing similar decode quality as existing literature.

The most salient feature in Fig. 7.5 and Fig. 7.6 is the similarity in performance across all firing rate methods. Let us consider, for example, the Kalman Filter results from Fig. 7.5, panel a. Looking across these eight green bars, there is no statistically significant difference between the RMSE results produced by any of the methods. If we consider different decoding algorithms (Linear Decode - red bars - or Kalman Filter - green bars), different performance metrics (RMSE - Fig. 7.5 - or correlation coefficient - Fig. 7.6), different dimensions of physical activity (horizontal - left panels - or vertical - right panels), and different data sets (`L2006A` - upper panels - or `L2006B` - lower panels), the story is unchanged: all seem to produce very similar performance results no matter what firing rate estimation method is used. In some cases the Kalman Filter may generally outperform (Fig. 7.5, panel d) or underperform the

Linear Decode (Fig. 7.6, panel c), or there may be generally higher error in data set `L2006A` than `L2006B`. In all cases though, there is very little trend that can be seen in the data suggesting that one firing rate method consistently outperforms any other. This finding is perhaps surprising, given the variety of firing rate estimates that are produced from the same spike trains using these different methods, as seen in Fig. 7.2.

We further note that, from our testing, this similarity in decode performance remains if different numbers of neurons are used, or if different lengths of trials are considered, or if different temporal lags are imposed between neural activity and physical behavior (as is often done in BMI studies), or if the firing rate data is considered at finer or coarser time intervals. In addition to these summary performance statistics, we note that, from our visual inspection of many decoded trials (*e.g.*, Fig. 7.4), all the firing rate estimators had the same performance in terms of how many decoded trajectories we described as "better" (*cf.* Fig. 7.4, panel c), "reasonable" (Fig. 7.4, panels a and b), and "failed" (Fig. 7.4, panel d). Thus, across all quantitative and qualitative analysis of the data that we have investigated, firing rate estimation offers little difference in terms of the quality of prosthetic decode. We discuss the implications of this seemingly general finding below.

7.5 Discussion and Conclusions

Optimally inferring neural firing rates from spike trains is an unanswered research question, and many groups have addressed this interesting problem. In this chapter, we reviewed some recent and some classic firing rate estimators. We discussed the theoretical motivation for each and discussed some potential advantages and disadvantages of competing methods. Firing rate estimation is a broad question that is applicable to neuroscientific and BMI applications, multiple and single trials, multiple and single neurons, and more. Each firing rate method should be considered specifically for its potential applications.

In this chapter, after reviewing these methods, we investigated the relevance of firing rate estimation methods for an important BMI application: decoding individual arm movements from simultaneously recorded neural populations. We trained a monkey in a standard reaching paradigm (as described in "Reach Task and Neural

Recordings" and in Fig. 7.3), and we used two standard decoding algorithms to estimate arm trajectories from neural activity. These algorithms - the Linear Decode and the Kalman Filter (as described in "Decoding Algorithms") - accept as input neural firing rates over a population of neurons. Using the same neural spike trains, we inferred neural firing rates using eight different firing rate methods, and then we decoded arm trajectories using these firing rates.

Though the firing rates found by all eight methods appear quite different (see Fig. 7.2), the decoding test indicated that in fact firing rate estimation matters very little for this domain of prosthetic decode. We showed in Fig. 7.5 and Fig. 7.6 that RMSE and correlation coefficients of the decode are rather insensitive to the firing rate estimation method that is used to process the neural spike trains. Looking across two dimensions of decode (horizontal and vertical), two different data sets with different neural populations (`L2006A` and `L2006B`), and two different decoding algorithms (Linear Decode and Kalman Filter), no discernable trend appears to indicate that one method (or one class of methods) is unambiguously better than any other. Thus, we believe the relevance of firing rate estimation, as it pertains to neural prosthetic decode, is in doubt.

Naturally the question then arises: how do such different firing rates (as in Fig. 7.2) produce such similar decode performance (as in Fig. 7.5 and Fig. 7.6)? We consider three possible explanations: (1) the decoding algorithms themselves are insensitive to differences in firing rate estimation; (2) the firing rate methods all have particular strengths and weaknesses but result in essentially the same signal-to-noise ratio (SNR); and (3) the ability to decode depends much more on factors other than firing rate estimation, and thus the firing rate estimator is not meaningful.

To the first point, if the decoding algorithms themselves smoothed over any differences in the firing rate estimations, we might expect very similar decoded trajectories. However, the different firing rate methods do in fact produce quite different decoded trajectories. Fig. 7.4 demonstrates this variety in four sample cases. Across the Linear Decode and the Kalman Filter, we find that the RMSE between different decoded trajectories (estimates derived from different firing rate methods) is typically 30-50% of the error with the true reach, and thus these estimates are indeed consistently different. Further, if the decoder was insensitive to firing rate estimates, we should be able to remove the firing rate estimator entirely (simply binning firing rate counts)

without change to the decode quality. We tried a simple binning scheme, using both 50ms (as used in, *e.g.*, Wu et al. (2002)) and 100ms bins (as used in, *e.g.* Chestek et al. (2007)). Interestingly, we find this simplifying step can change error meaningfully, increasing error considerably in the case of the Linear Decode (but less so with the Kalman Filter; indeed, sometimes binning reduces error in the Kalman Filter case). Thus, temporal smoothing of firing rates seems valuable, and the method of smoothing influences the decoded arm trajectory meaningfully. Based on these findings, we see that the decode algorithms themselves are indeed sensitive to differences in firing rate estimation.

To the second possibility, each firing rate method does seem to make particular tradeoffs between signal and noise. In the simplest case, a low bandwidth kernel smoother (such as KS150) will produce a slowly varying firing rate with a similar time course to the arm activity. However, it also eliminates steep changes in firing rate, which likely provide a meaningful signal to the timecourse of arm movement. Fig. 7.2, panel (b), shows this possibility: while KS50, GPFR, and others pick up the sharp "ON" transient in the firing rate, they also pick up noise in the subsequent high firing rate. In contrast, KS150 smooths out both the noise and the step change in firing rate. Thus, it is likely that these firing rate methods and others each represent some balance between capturing or removing both signal and noise. Loosely, while each method may result in very different firing rate estimates, the SNR of each estimate may in fact be similar.

To the third possibility, it seems quite likely that the biggest effect on decode performance comes from aspects of the decoding system that are not neural firing rates. For example, the addition or removal of one or more very informative neurons to the neural population does often alter these performance numbers considerably (we found this effect in our additional testing), thus suggesting that recording technology (such as Wise et al. (2004)) may be more critical. Furthermore, the consideration of neural plan activity (before the movement begins) has been found to significantly reduce decoding error (Kemere et al., 2004; Yu et al., 2007). These are two examples of a host of avenues that may be significant determinants of prosthetic performance. Other avenues, as previously noted, may include prosthetic decode algorithms in general (Georgopoulos et al., 1986; Brown et al., 1998; Wu et al., 2004; Brockwell et al., 2004; Wu et al., 2006), the prosthetic interface itself (Schwartz, 2004; Velliste

et al., 2008; Cunningham et al., 2008b), and multiple signal modalities (*e.g.*, EEG, ECoG, LFP, and spiking activity) (Mehring et al., 2003). Even if these other factors are much larger determinants of performance than firing rate estimation, one might still hope to see that certain firing rate estimators performed unambiguously better (albeit only slightly better) than others. Looking across decoders and data sets and error metrics, such a claim can not be made.

Despite the questionable relevance of firing rate estimation to the problem of neural prosthetic decode, we want to strongly clarify that we do not call into question the validity of firing rate estimation in general. Many of the excellent papers in this domain (several of which were reviewed in this study) may have important applications in neuroscientific studies or some other domain of neural signal processing. For example, these methods may be especially important in settings, unlike arm movements, where experimental conditions can be closely copied on each trial, producing similar neural responses (*e.g.*, visual stimuli shown to *in vitro* retinal neurons (Pillow et al., 2005)).

Neural prostheses and BMI have received much attention in the last decade. As a result, many researchers from many fields have studied ways to improve our ability to understand and decode neural signals. Despite this preponderance of methodological development, very few systematic comparisons have been made in real experimental settings. The gold standard for such a comparison is perhaps online (closed loop) clinical trials, where the BMI user may engage learning, neural plasticity, and a host of other feedback mechanisms. Prior to that step, offline comparisons should be made on a variety of experimentally gathered data, and these comparisons can be made between all aspects of neural prosthetic systems. It behooves the field to review and compare available methods at each step in the BMI signal path. In this chapter, we have made a first effort in that direction by reviewing and comparing different firing rate estimation methods. Prosthetic decoding algorithms may be another attractive target for such a review and comparison. The field should largely benefit from such studies, both in terms of benchmarking the past and helping to set research agendas for the future.

7.6 Acknowledgments

We thank Cindy Chestek and Rachel Kalmar for valuable technical discussions. We thank Mackenzie Risch for veterinary care, Drew Haven for technical support, and Sandy Eisensee for administrative assistance.

7.7 Grants

This work was supported by the Michael Flynn Stanford Graduate Fellowship (JPC), NDSEG Fellowship (VG), NSF Graduate Research Fellowship (VG), NIH-NINDS-CRCNS-R01, Christopher and Dana Reeve Foundation (SIR & KVS), and the following grants to KVS: Burroughs Wellcome Fund Career Award in the Biomedical Sciences, Stanford Center for Integrated Systems, NSF Center for Neuromorphic Systems Engineering at Caltech, Office of Naval Research, Sloan Foundation, and Whitaker Foundation.

Chapter 8

Neural Prosthetic Systems: Current Problems and Future Directions

Having demonstrated one area where performance improvements were found (Chapter 6), and one area where they were not found (Chapter 7), this chapter now turns to the broader question of what areas are available for future algorithmic research in neural prosthetic systems, and what opportunities they might present for performance improvement. This chapter highlights several outstanding problems that exist in most current approaches to prosthetic signal processing. These include two problems that we argue are unlikely to result in further dramatic increases in performance, specifically spike sorting and spiking models (as detailed in the previous chapter). We also discuss three issues that have been less examined in the literature, and we argue that addressing these issues may result in dramatic future increases in performance. These include: non-stationarity of recorded waveforms, limitations of a linear mappings between neural activity and movement kinematics, and the low signal to noise ratio of the neural data. We demonstrate these problems with data from 39 experimental sessions with a non-human primate performing reaches and with recent literature. In all, this study suggests that research in cortically-controlled prosthetic systems may require reprioritization to achieve performance that is acceptable for a clinically viable human system. This work, which is being published as Chestek* et al. (2009), was done jointly with Cindy Chestek, Vikash Gilja, Paul Nuyujukian,

Stephen Ryu, and Krishna Shenoy. Cindy and I acted as joint first authors on this project, and I was particularly involved in data analysis and paper writing.

8.1 Introduction

In recent years, advances in neural technologies have enabled the creation of neural prosthetic systems (variously called neural interfaces, brain-machine interfaces, or BMI) that aim to help severely disabled human patients. There are many medical, scientific, and engineering challenges in developing such systems (Lebedev and Nicolelis, 2006; Schwartz, 2004; Donoghue, 2008; Linderman et al., 2008; Sanchez et al., 2008), and all neural prosthetic systems share in common a signal processing backend. This backend takes as input raw voltage waveforms from multi-electrode recordings (or other technologies), and it produces as output a control signal such as kinematic parameters to control a prosthetic arm. Along this signal flow, there are two major steps: first, raw voltage must be separated into spike trains from single or multiple neural units, called "spike sorting"; second, these spike trains must be processed by a decoding algorithm to produce behavioral control signals. Both of these steps have been well studied: spike sorting (Lewicki, 1998; Wood et al., 2004; Wood and Black, 2008) and decode algorithms (Georgopoulos et al., 1986; Gao et al., 2002; Kemere et al., 2004; Brockwell et al., 2004; Kim et al., 2006; Wu et al., 2006; Santhanam et al., 2006; Shakhnarovich et al., 2006; Yu et al., 2007; Srinivasan et al., 2007; Velliste et al., 2008; Ventura, 2008; Wu and Hatsopoulos, 2008; Kulkarni and Paninski, 2008; Sanchez et al., 2008; DiGiovanna et al., 2009). These works have delivered important proofs of concept that brain-machine interfaces can translate neural signals into physical commands. However, moving to a clinically viable system will require several significant developments. These developments exist at all stages: in the recording technologies (Wise et al., 2004), the signal processing backend, and prosthetic end effectors such as robotic arms and computer interfaces (Cunningham et al., 2008b). This study introduces several problems in the signal processing domain.

First, the field must better understand how the recorded signals change over time, as there has been much work suggesting various levels of stability in recorded neural activity over time (Santhanam et al., 2007; Suner et al., 2005) - discussed in Section 8.3.1 below. Next, noisy spike trains must be meaningfully processed into neural firing

rates or other quantities appropriate for input into decode algorithms; we address this potential problem in Section 8.3.2 below. Decode algorithms calculate a mapping between physical behavior and neural activity. We introduce unresolved questions in these models in Section 8.3.3 below. Further, a large problem may be fundamental limitations in the data - discussed in Section 8.3.4. These limitations exist due to an insufficient signal to noise ratio in the limited number of neural channels available, as well as model mismatch (e.q., many algorithms assume linear mappings to model nonlinear relationships). Other limitations may also exist in experimental design and algorithmic testing, and we discuss those potential issues in Section 8.3.5. Many of these aspects of BMI performance can interact in complex ways. However, as a starting point in this study, we will address them individually.

8.2 Methods

8.2.1 Animal Task and Neural Recordings

Animal protocols were approved by the Stanford University Institutional Animal Care and Use Committee. We trained a rhesus monkey (*Macaca mulatta*), monkey L, in a standard reaching paradigm that has been extensively reported elsewhere (Chestek et al., 2007; Cunningham et al., 2008b, 2009). We give a short overview here. We implanted a 96-electrode Utah electrode array (Blackrock Microsystems, Salt Lake City, UT) into premotor cortex. The array was implanted 10 months prior to the experiments, showed substantial neural activity, and continued to do so for several months after the experiments[1]. The monkey is trained to make instructed reaches to a number of points (28 peripheral targets at 4 radial distances from the central target, uniformly distributed in 7 directions) on a vertical screen. Monkey L begins with his hand on a target at the screen center. After a brief hold time, a peripheral target appears, indicating the goal of his reach. Restrictions on reaction time ensure that the monkey will reach quickly and accurately to the peripheral target, then receiving a juice reward (Chestek et al., 2007). This experiment was performed on 39 days over a period of 7 weeks. Prior to this time, this monkey had been heavily trained on similar tasks for several years. Here we analyze the first hour of data from each day,

[1]Previous reports discuss the same monkey. Here we use a newer implant (same technology) and a very similar experimental paradigm.

with an average of 1655 reaches per dataset.

8.2.2 Neural Prosthetic Decoding

In the results that follow, we will demonstrate the quality of decoding neural activity based on different segmentations of the neural data. Accordingly, we need a method with which to decode neural activity into action so that we can compare performance of different signal processing techniques. We describe those methods briefly here, where we refer to blocks of the general signal flow for a BMI, as shown in Fig. 8.1.

To extract spike trains from raw voltage, neural units were isolated off-line using a PCA-based spike sorting algorithm (Sahani, 1999), and quality was assessed by hand using the waveforms and clusters in principle component space. Units were labeled single unit, contaminated single-units (with waveforms from other neurons), and multi-units. For analyses using threshold crossings only, all events that crossed a threshold of three times rms noise were used; more explanation can be found in Cunningham et al. (2009). All of these threshold crossings were classified as single or multi-units in the full spike sorting analysis.

First, we use a simple maximum likelihood (ML) decoder, as seen in Santhanam et al. (2006); Cunningham et al. (2008b). This method uses training data to build an expectation, for reaches to each of the reach targets, of the number of spikes recorded from each neuron. Given test data, the ML decoder evaluates the likelihood (under a Poisson noise model) and picks the reach condition with the largest value (hence maximum likelihood) as the decoded reach. The percentage of reach conditions correctly decoded is reported as overall performance (Cunningham et al., 2008b).

ML decoding makes a discrete choice. In some cases, we also want to decode moment-by-moment parameters of the subject's reach. To do this, we use the popular linear decoder (LD), which assumes that movement is a linear combination of recorded neural activity. Using least squares, the movement can be decoded from neural activity, and common metrics such as root-mean-squared-error (RMS) or correlation coefficient can be used to determine the quality of decode (Chestek et al., 2007; Cunningham et al., 2009). A third common approach is the Kalman filter (Kalman, 1960), which stipulates a linear relationship between physical behavior over time and between neural activity and physical behavior (Wu et al., 2006; Cunningham et al., 2009).

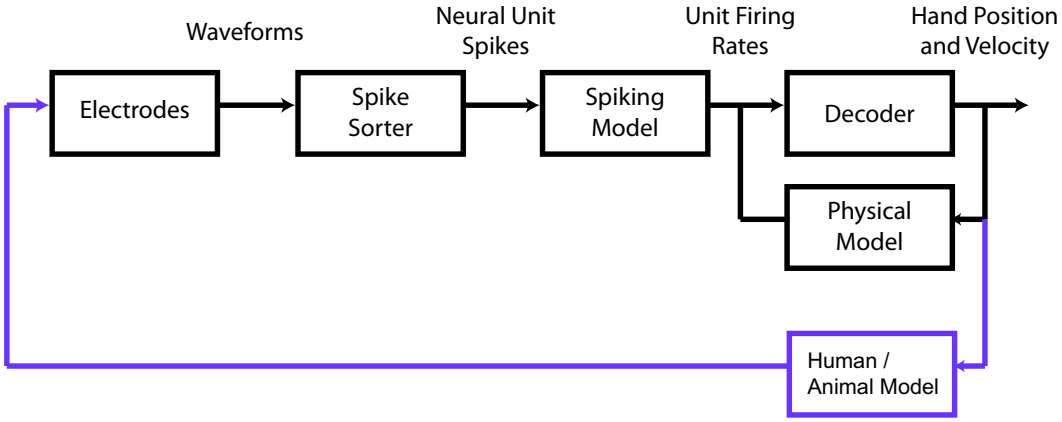


Figure 8.1: Block diagram of a typical BMI illustrating potential areas for improvement. The lower feedback loop illustrates aspects of neural adaptation that can be engaged only in closed-loop experiments.

8.3 Results and Discussion

Here we discuss the problems we highlighted in the introduction, and we demonstrate these problems in our experimental data and recent publications.

8.3.1 Spike Sorting

Spike sorting is a major challenge in neural signal extraction, both for basic neuroscience studies and for neural prosthetic systems (Lewicki, 1998; Wood et al., 2004; Wood and Black, 2008). We discuss here the importance of isolating single neuron activity, and the instability of neural recordings over time.

Single Unit Activity

When studying the properties of individual cells, it is important to isolate "single units" with accurate spike sorting. Often, recorded neural activity that likely arises from multiple cells is excluded from analysis, despite the fact that such activity typically comprises a substantial portion of recorded neural activity. For example, from a single dataset, we differentiated all neural events into 205 clusters. Of these 205 neural units, only 53 came from well-isolated or somewhat contaminated units (units with clearly differentiated waveforms that were either not adjacent to other waveforms in voltage and PCA space (well isolated) or adjacent but clearly distinct (somewhat

contaminated). The remaining 152 neural units were classified as likely multi-unit. Table 8.1 shows the percent correct for a ML decode of reach directions from single unit only and multi-unit activity. Also, adding multi-unit activity to single unit activity increases performance from 74% to 82%. Therefore, it seems clear that multi-units should be included in prosthetic decoders, despite being "unclean" isolations in a basic science sense.

The complexity of the spike sorting process has substantial power implications for integrated circuits that may be used as part of future clinical systems to transmit wireless neural data from the patient (Wise et al., 2004; Harrison et al., 2007; Zumsteg et al., 2005), since this complexity changes the number of bits required per channel for full waveforms versus threshold crossings. This may be partially alleviated by small process technologies or novel powering methods, but is still likely to be a substantial concern. This raises the question, does full spike sorting produce a large improvement in decode performance over threshold based systems? Table 8.1 shows a performance comparison between PCA based spike sorting, using all single and multi-units (third row) and a single threshold per channel (3 times RMS noise, fourth row). While small increases in performance can be important to users, using sorted spikes instead of thresholds produces a surprisingly small improvement of 7%. Also, the threshold number represents a base level performance which could likely be improved by setting the thresholds optimally on a per-channel basis. Further, two thresholds per channel could also substantially make up the difference in performance without requiring full broadband data. The optimal may resemble (O'Driscoll et al., 2006), in which bits of resolution are distributed to channels based on information content.

Waveform Shape Instability

While there is evidence that neurons themselves maintain stable tuning properties at least over the course of a day (Chestek et al., 2007), there is significant doubt about the stability of the raw voltage recordings of those neurons over the same time periods (due to changing position of the electrodes with respect to the neurons, or similar) (Santhanam et al., 2007; Suner et al., 2005). If these recordings are not stable, accurate spike sorting will require additional sophistication to track neural units over the course of minutes, hours, and days (Bar-Hillel et al., 2006).

Fig. 8.2 shows the dramatic effect of this instability on decode performance. We

Table 8.1: Decode performance by unit type.

	Number of Units	Decode Performance	
		ML Decode[1]	Correlation Coeff.[2]
Single units[3]	53	74%	0.86
Multi-units	152	79%	0.91
All sorted units	205	82%	0.92
Thresholds[4]	96	75%	0.89

[1] Note that chance ML decode accuracy is 1/7, or 14%.
[2] Correlation coefficient based on a linear decoder.
[3] Includes definite and high-confidence single units.
[4] Standard thresholding at 3 times RMS noise.

fix a decode algorithm to the population recorded on an array on the first day of recording. We fix both the maximum likelihood parameters and the spike-sorting projections (waveform shapes) across seven weeks. Performance falls precipitously after only a few days, which must be due to changes in the recorded neural activity (the signal processing backend has been held constant). This suggests that nonstationarity will be a substantial problem in future clinical systems. Human systems to date have used daily calibration by skilled technicians (Hochberg et al., 2006), but this approach will not economically scale to broad use.

Fig. 8.3 shows how much a recording can change over the course of a single experimental recording session. While the average change in waveform shape is small (many remain within +/- 5% of their time zero size) several neurons indeed change their waveforms significantly over just one hour (e.g. red growing by 25% and the blue shrinking by 25%). The average absolute change was 0.3%/min. Looking at the slopes in absolute voltages, the average change was 1 μV/min, but changes above 5 μV/min were observed on several units. Examination over even greater time spans may reveal even greater excursions. More accurately characterizing these changes requires analysis of multi-week wireless recordings (Chestek et al., 2009) from additional animals[2]. These waveform changes can and do cause serious spike sorting

[2]It is possible that some of the change in performance over days is due to slight differences in connector impedance. However, this likely makes only a small contribution since the noise across the array was fairly stable (mean 1.1 μVrms, std 0.2 μVrms). Also, there were a few electrodes with highly similar waveforms across days. At the the same time, waveform changes on individual

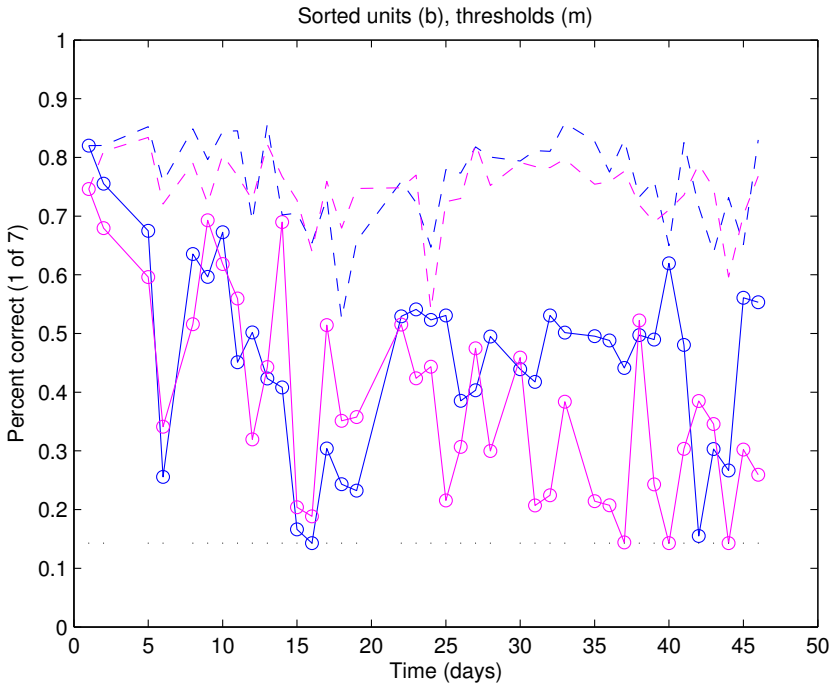


Figure 8.2: Decoder performance (ML accuracy) determined by generating spike sorting templates and maximum likelihood coefficients from the first day of the experiment, and applying those models across 7 weeks of similar experiments. Sorted data shown in blue, threshold data shown in magenta; dotted lines indicate data that was re-fit on each experimental day.

difficulties. Fig. 8.4 shows a single unit that remained well isolated over several weeks (uncontaminated units like this are rare). The tuning curves in the second row suggest this is the same neuron, despite substantial waveform changes. More commonly, as shown in the third row, a unit that is initially well isolated disappears over days into multi-unit activity. Since spike-sorting algorithms rely on waveform shape, these instabilities may confound spike sorting significantly over the course of several hours, and certainly across days.

Some aspects of these instabilities might be particular to the experimental preparation considered here. However, these devices were approved for initial human studies (Hochberg et al., 2006), are likely to be used for future human work, and are believed to be at least as stable (if not more so) than other multi-electrode technologies due

electrodes could be dramatic over only a few hours.

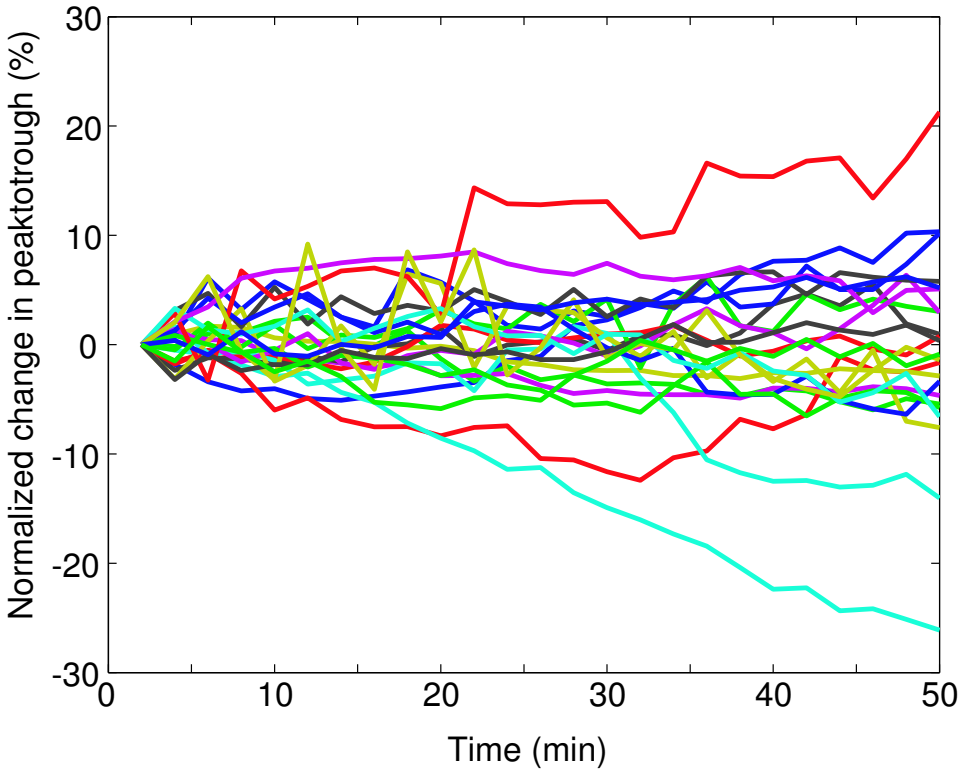


Figure 8.3: Waveform change over one hour for 23 example neurons. Change is normalized to the size of the initial waveform.

to its ability to move with the brain rather than being secured to the skull. Accordingly, one might consider different strategies going forward to compensate for these instabilities. This may be substantially easier in neural prosthetics than basic science because it is arguably not important to track single units. Perhaps decode algorithms can be designed by sorting on tuning alone over time or using simply no spike-sorting algorithm whatsoever (Ventura, 2008). In any event, since the effect on performance is high, serious effort to address these instabilities must be committed.

8.3.2 Models for Neural Spiking

Spike trains present analytical challenges due to their noisy, spiking nature. A common view is that spikes are generated from a smooth function of time (the firing rate) and that this function carries a significant portion of the neural information (vs. the precise spike timing). If so, decoding neural activity may require accurately estimate firing rate. There has been extensive work in modeling spike trains

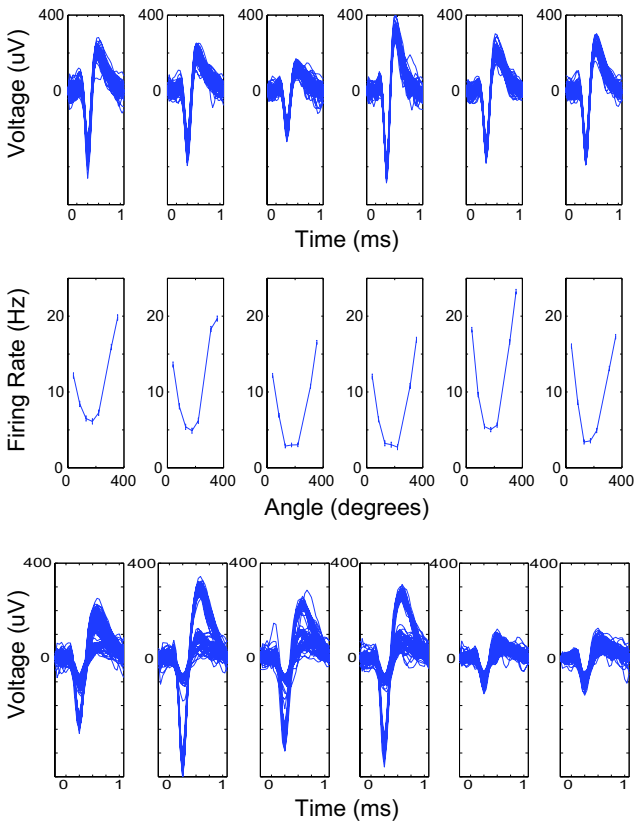


Figure 8.4: Example waveforms over days. Panel A shows an isolated unit changing over time but (usually) remaining well isolated. From left to right, top panels show the single unit from days 1,5,16,17,34, and 45. Panel B shows the tuning pattern across 7 angular directions on those days. Panel C shows a more common example of a waveform shape collapsing into the multi-unit activity. Waveforms crossing threshold are shown for one electrode on six consecutive days.

(Johnson, 1996; Barbieri et al., 2001; Ventura et al., 2002; Kass and Ventura, 2003; Truccolo et al., 2005; Sanchez et al., 2008) and estimating firing rates (Shimazaki and Shinomoto, 2007a; Cunningham et al., 2008c; DiMatteo et al., 2001; Kass et al., 2005; Endres et al., 2008). While some decode algorithms average over neural activity in small temporal windows (Yu et al., 2007), some algorithms use firing rates or use spiking models directly (Srinivasan et al., 2007). Spiking models are another source of approximation in BMIs. Though sophisticated firing rate estimation has proven valuable in basic neuroscience, a recent study found minimal differences in prosthetic decode performance using different estimators (Cunningham et al., 2009) (as discussed in Chapter 7). Perhaps models for neural spiking, though clearly yet one more approximation in decoding, may not be a source of major performance gain

for future research.

8.3.3 The Mapping between Physical Behavior and Neural Spiking

To date, essentially all prosthetic decode algorithms (population vectors, linear decoders, Kalman filters, etc.) have assumed a linear mapping between kinematic parameters of the arm and neural firing rates (or, in some cases, spiking activity directly) (Georgopoulos et al., 1986; Gao et al., 2002; Kemere et al., 2004; Brockwell et al., 2004; Kim et al., 2006; Wu et al., 2006; Santhanam et al., 2006; Shakhnarovich et al., 2006; Yu et al., 2007; Srinivasan et al., 2007; Velliste et al., 2008; Ventura, 2008; Wu and Hatsopoulos, 2008; Kulkarni and Paninski, 2008; Sanchez et al., 2008; DiGiovanna et al., 2009). There are a few potential shortcomings with this linear choice, including the fact that most algorithms ignore meaningful nonlinearities in neural data, and the poor generalization of these models.

Nonlinearities

There is wide variation in how well the activity of a neuron can be linearly related to a given kinematic parameter, shown in Fig. 8.5. The top panel shows the average reach speed, and the second panel shows X-position for reaches to 7 out of 28 targets. These respresent typical kinematic signals that one would like a linear model to accurately predict. The middle panel shows that some neurons that have a strongly linear relationship with speed given a specific time lag. In general, a subset of neurons may have a strong linear relationship with a given kinematic parameter. However, the remaining two panels show firing rates from four neurons with firing rates that do not have an obvious linear relationship to any parameter. For example, activity in the fourth panel comes from two units with long plateaus of activity that precede and follow movement. The two bottom units show double peaks, that also have no obvious linear transformation to kinematic parameters.

One way linear decoders can cope with non-linearities is to use only units with clear linear relationships and set other coefficients to low values. In our data, linear decoders can come within 10% of the optimal error using between 15-29% of the 53 predominantly single units (for x and y position and velocity). This number was

obtained by sorting individual units for correlation with the various parameters, and adding them to the decoder until the error was within 10% of its value for the whole ensemble. To achieve this performance level on all 4 kinematic parameters together, only 49% of the units were required. More than half the units were unused. This underutilization may occur because neurons with nonlinear relationships to behavior provide a source of model mismatch to linear decoders. However, model mismatch is not noise; there is possibly information in these neural units that linear decoding models (like the linear filter, the population vector, and the Kalman filter) are unable to exploit. Future algorithmic designs may offer significant performance improvements by modeling nonlinearities in this mapping. Also, nonlinearities could be introduced at many points in the signal flow shown in Figure 8.1, not just the mapping considered here.

Generalization

The ability of a model to generalize to novel conditions is a major concern with any decoding algorithm. Linear algorithms in particular may generalize poorly to novel reaches. For example, in the current dataset, determining an optimum linear filter using 27 out of the 28 targets and testing on reaches to the remaining target resulted in a 4x greater squared error on average than training on a dataset that included reaches to that target. This occurred despite the fact that the training dataset included 3 other examples of reaches to the same angle and 6 other reaches to the same distance. It is notable that many other prosthetic experiments to date have used highly constrained movement tasks which may overestimate the ability of linear models to generalize (Yu et al., 2007; Kim et al., 2008). While these tests indeed demonstrate useful signal extraction from cortex, they do not test a broad range of behavior. Accordingly, it may be that these constrained experimental settings pose an unrealistic proxy to the eventual user mode.

A real prosthesis user will desire a broad range of potentially novel behaviors. An accurate model mapping physical behavior to neural activity must be able to decode novel reaching conditions. Moving to unconstrained settings in three dimensions, with many other types of reaching - curved, straight, point-to-point, continuous, and more - there are many possible model mismatches. Further, arbitrary movement in three dimensions engages long-studied questions of reference frames and coordinate

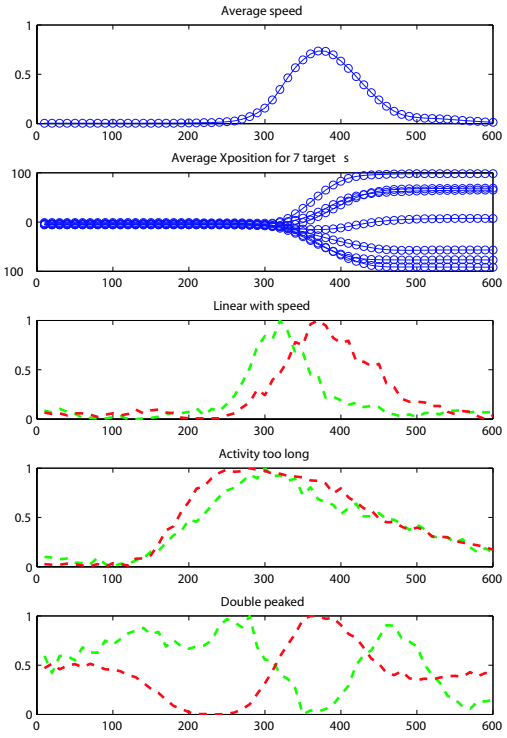


Figure 8.5: Comparison of kinematic parameters (blue traces) with neural firing rates (red and green traces). (A) Average speed during reaches. (B) Average X-position during reaches to 7 out of 28 targets. (C) Normalized average firing rates from two units with a strong linear relationship to velocity. (D) Two example units whose activity precedes and postcedes neural activity. (E) Two example units with double peaked average firing rates. For both (D) and (E) there is no obvious linear transform between neural activity and any of the kinematic parameters.

transformations (Shadmehr and Wise, 2005), which may complicate things further. In short, experimental constraints may not translate to a prosthesis that generalizes to the needs of a human user. Some effort should be made in vetting all BMI developments with a range of experimental control (including very little).

8.3.4 Limitations on Precision

There is obviously not arbitrarily large information content in a given number of neural channels. For example, while the output of a continuous linear decoder can exhibit high correlation with the actual hand movement, single trials often decode to erratic reach behavior. Figure 8.6A shows an example of actual reaches to one of the 28 targets in the center out task. A position-based linear decoder trained on

the first half of the dataset predicted reaches in the second half of the data with a correlation coefficient of R=0.88, which shows similar performance to other results in the literature (Wu et al., 2006; Kim et al., 2008). Reaches decoded by the linear model are shown in Fig. 8.6B. While the average correlation is apparent, the endpoints exhibit a much higher standard deviation (21 mm vs 6 mm) than actual reaches (the red ellipse).

This illustration represents an "offline" linear decode. One might argue that these incorrect trajectories can be corrected using feedback in an "online" BCI experiment. However, online linear models have shown a tendency to move erratically as well (Kim et al., 2008). This may place limitations on how closely spaced potential targets can be and whether undesired targets can be avoided. Moving from computer control to the control of a robotic limb would further emphasize this problem.

Models for Physical Behavior

One weakness of algorithms like the linear decoder (and population vector) is that these algorithms do not have an explicit physical behavior model, and thus all noise in the recorded signal is passed through to the decoded arm trajectory. In contrast to this shortcoming, models such as the Kalman filter (Kalman, 1960), which stipulate a model for physical behavior in arm reaches, have been shown to outperform the linear decoder in a variety of cases (Wu et al., 2006; Kim et al., 2008). This success led to extensions that assume similar models for physical behavior (Yu et al., 2007; Srinivasan et al., 2007; Wu et al., 2008; Kim et al., 2006; Wu and Hatsopoulos, 2008; Kulkarni and Paninski, 2008; Sanchez et al., 2008; DiGiovanna et al., 2009). Unfortunately, this class of models for physical behavior is inappropriate in some ways for reaching movements.

Specifically, the Kalman filter assumes a linear dynamical system ($\mathbf{x}_t = A\mathbf{x}_{t-1} + v$, where v is some noise). Depending on the matrix A, reaches from this distribution can only converge to the origin, oscillate, or diverge to infinity, which conflicts with the reality that the majority of arm reaches are point-to-point (Shadmehr and Wise, 2005). Fig. 8.6C shows that this model can decoded reaches that fail to stop. Overall, the performance does not appear substantially less erratic than those from the simple linear decoder. The Kalman filter, like other linear models, fails to infer the correct reach goal and stop precisely. These inadequate physical models have been chosen in

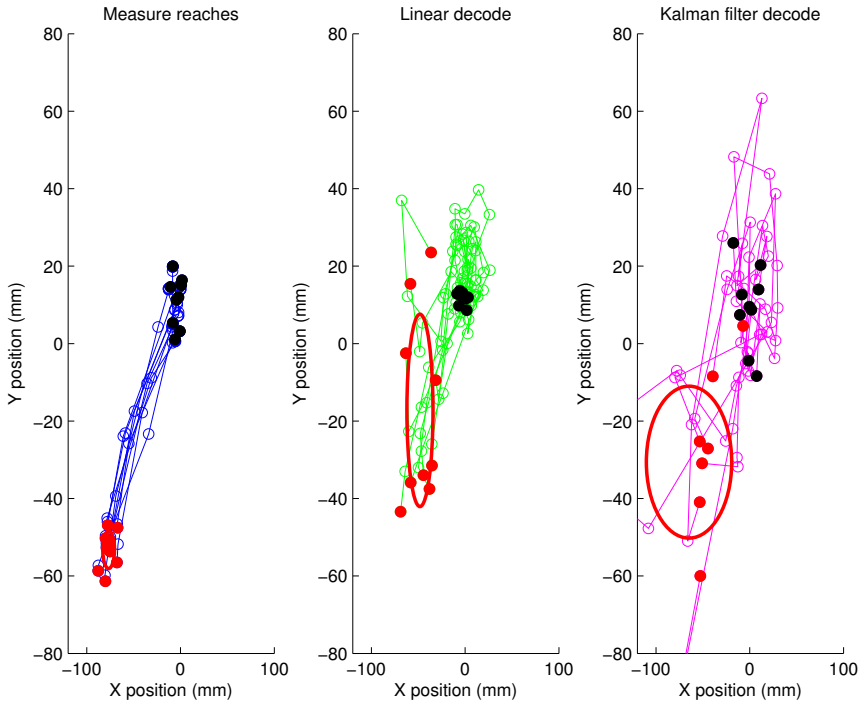


Figure 8.6: Panel A shows actual reaches to one of 28 targets measured using an infrared motion tracking system. Panel B shows the linear decode of neural activity during those reaches. Panel C shows a Kalman filter decode of that same activity. Black dots denote the end points of each reach, and the black ellipse denotes the standard deviation in the X and Y direction. Note the red end-point variance ellipse is very small in the first panel.

large part because of their mathematical tractability. Instead, a model could exploit the deep literature describing how reaches are actually made in human behavior (Shadmehr and Wise, 2005).

Data Limitations

Current recording technologies can record from up to hundreds of individual neurons, which is a tiny fraction of the many millions involved in arm reaching. Accordingly, the field is and will continue to be limited in the amount of information it can record from cortex.

BMI devices using lower SNR sources such as ECoG often try to maximize information throughput by using an "indirect" signal source. For example, imagining

something that can be somewhat unrelated to arm movement in order to generate cursor movement (Leuthardt et al., 2006). Cortical BMI's have relatively higher SNR, and can attempt a "direct" decode (Donoghue, 2008). For example, Santhanam et al. (2006); Yu et al. (2007) decoded movements towards particular targets, but the number and position of targets was small, fixed, and known. Kim et al. (2008) demonstrated improved accuracy with humans controlling a computer cursor by using a training paradigm of reaches that moved very slowly.

Much traction might be gained by restricting the space of movements that can be decoded from neural activity. Researchers may consider the field of human motor control (*e.g.*, Shadmehr and Wise (2005)), where work has shown fundamental constraints on the human reaching system. By similarly constraining the space of movements that can be decoded from neural activity, some performance improvements may be achieved using currently available signal sources. By recognizing that there is not an arbitrary amount of information in the recorded neural activity, the field can begin to ask meaningful questions about what actions we may hope to extract from cortex. Designing decode strategies in this way will be critical in moving towards a clinically viable system.

8.3.5 Experimental Limitations

In this final section we introduce another potential issue in current prosthetic design, and we discuss why we think addressing this issue may be a valuable direction for future investigation. As previously noted, experimental constraints do not necessarily translate to a prosthetic device that can generalize well. For example, decoding success is often determined by how well the decoded arm trajectory matches the true arm movement that was recorded alongside the (possibly synthetic) neural activity. Unfortunately, this "offline" approach neglects potentially important features of a real neural prosthesis, including the prosthetic user's ability to modify behavioral strategies to improve control of the prosthetic device (via the decode algorithm). In other words, as soon as the prosthesis user sees the prosthetic device act, he/she will bring to bear all his/her behavioral modification strategies to attempt to drive a natural, desired reach. In moving towards a usable prosthesis, experimental paradigms should be tested in this "online" context in order to provide a realistic proxy to clinical use.

This feature is noted by the large feedback loop in Fig. 8.1. The field should investigate the extent to which the subject can (for a given decode algorithm, spike sorting approach, or other signal processing choice) engage feedback mechanisms, learning and adaptation, and other control strategies to improve decode performance.

8.4 Conclusions

Neural prostheses have received much attention in the last decade. In this study, we used 39 neural datasets, from a single monkey making center out reaches day after day, to examine potential areas for future advances. These analyses suggest that areas such as single unit spike sorting and advanced spiking models, while useful to pure neuroscience research, may not provide dramatic performance increases in future BMIs. However, there are three areas that we believe may provide more space for improvement. First, non-stationarity of neural waveforms must be addressed when moving towards long term clinical systems. Second, linear models may not be fully exploiting information available from particularly non-linear neurons. This may also lead to observed difficulties in model generalization. Third, erratic decoded movements cause difficulty in predictably controlling a BMI cursor. This shortcoming could be mitigated by more careful analysis of the neural information content, by limiting the types of reaches based on the information available, and by meaningfully testing algorithmic developments in an online context. In all of these issues, it is of great value for the field to review and compare available methods at each step in the BMI signal path, and to design future studies (both experimental and algorithmic) with those results in mind.

8.5 Acknowledgments

We thank M. Risch for veterinary care, D. Haven for technical support, and S. Eisensee for administrative support.

8.6 Grants

This work was supported by: Burroughs Wellcome Fund Career Award in the Biomedical Sciences, Christopher and Dana Reeve Foundation, HHMI Fellowship (P.N.), Johns Hopkins University Applied Physics Laboratory under the DARPA Revolutionizing Prosthetics program contract N66001-06-C-8005, The McKnight Endowment Fund for Neuroscience, NDSEG Fellowship (V.G.), NIH-NINDS N01-NS-4-2362, NIH-CRCNS-R01, NSF Graduate Research Fellowships (C.A.C., V.G.), Soros Fellowship (P.N.), Stanford's CIS, SGF (C.A.C., J.P.C.), and Medical Scholars Program (P.N.).

End of Part II

Part III

Conclusions and End Material

Chapter 9

Conclusions and Future Work

The last seven chapters have demonstrated several algorithmic contributions towards the broad goals of understanding motor cortical processing and improving the design of neural prosthetic systems. The following sections briefly summarize these findings at a high level and then point to interesting avenues of future work.

9.1 Part I

In the first part (understanding motor cortical processing), we developed novel signal processing methods to interrogate single trial neural data, first from single neurons (Chapter 2), and we then extended that to populations of simultaneously recorded neurons (Chapter 5). We showed, based on various error metrics, that these methods model neural activity better than competing, simpler methods. The first part also demonstrated that these technically complex algorithms can cause potentially serious computational problems, but that this burden can be alleviated significantly by problem-specific implementation considerations (Chapter 3). Then, we discussed how these computational considerations led to a finding of broad statistical interest, namely a new method for calculating multivariate Gaussian probabilities (Chapter 4). Finally, scientifically, using the GPFA method of Chapter 5, we then showed that this method allows us to visualize across-trial convergence of neural activity during the planning phase of a delayed-reach task, an effect which was only previously seen indirectly.

This across-trial convergence is an example of the neuroscientific questions that

can be asked and answered with analytical methods designed to investigate cortical processing. However, it is certainly the case that this effect was seen previously in Churchland et al. (2006b), albeit through a carefully designed indirect method. A critical question for future study then is: are these neural trajectory algorithms (GPFA and similar, as described in Part I) more than visualization tools? Answering this question is a critical area of future work. Here, I address two areas of future study that aim at this question. First, I discuss the utility of direct visualization methods in driving scientific hypotheses. Second, I discuss the importance of denoised, reduced-dimension views of neural activity in driving towards an understanding of the computational mechanisms at work in cortical processing.

First, the scientific value of visualization methods should not be underestimated. As detailed in Section 5.1, there are a number of experimental settings where single-trial, population-level neural data should be visualized. For each of the examples shown in Fig. 5.1, researchers have shown ways to detect (or indirectly view) trial-to-trial differences in the neural responses, including computing streak indices (Horwitz and Newsome, 2001), estimating firing rates from a single spike train (Roitman and Shadlen, 2002; Cook and Maunsell, 2002; Cunningham et al., 2008c), and measuring the across-trial variability of neural responses (Churchland et al., 2006b). In all of these cases, however, what one really wants is a direct view of the time-evolution of the neural response on single trials. Were such a visualization available, perhaps these and other effects could be seen more readily, and perhaps these and other effects could be reported with fewer experimental and analytical controls. Indeed, the PSTH and firing rate estimation in general are visualization tools that have allowed many scientific discoveries by allowing researchers to view their data in a more compact and intuitive representation (rather than just looking at rows of spike rasters), thereby facilitating hypothesis generation. I believe the scientific merit of the neural trajectory algorithms of Part I should not be underestimated from a visualization standpoint alone. "Data-driven" hypotheses in science are frequent, and they rely on data visualization. Particularly as future data sets are becoming larger and more complex, future work in developing useful visualization tools is important.

To highlight this point, I here give two examples of future work that is in various stages of development in our research group. The scientific hypotheses in these

ongoing studies were formulated by viewing low-dimensional visualizations of populations of simultaneously recorded neural activity. First, I return to the direct view of across-trial convergence of neural trajectories during the course of motor planning (as in Chapter 5 and, previously, in Churchland et al. (2006b)). This convergence hypothesis suggests that there is an optimal configuration of neural activity that indicates a specific reach plan, and thus the proximity of neural activity to that optimum correlates with how quickly a reach can be subsequently triggered (this reaction time effect is thoroughly discussed in Churchland et al. (2006b)). Another researcher visualized populations of neural activity using a neural trajectory algorithm and formed a different hypothesis: rather than an optimum, perhaps reaction time is determined by how far the neural activity, at the time of the movement cue, has progressed along the path towards the average point (in the space of neural activity) at which the reaching movement begins. This hypothesis significantly improves the scientific correlate to behavior. This study, which is discussed preliminarily in Afshar (2008), was enabled by neural trajectory visualization tools. Indeed, the eventual analysis can be done in the raw data, but the hypothesis came from viewing compact representations as discussed in Part I. A similar hypothesis has been tested by Rachel Kalmar in the oculomotor system of monkeys, where she has shown an even stronger effect predicting reaction time in an eye-saccade task. This ongoing work has also shown that looking at the neural population as a whole creates a significantly better correlate than looking at single neurons individually. This finding indicates there is meaningful structure in the high-dimensional neural data. This high-dimensional structure, however, can not be directly viewed in a high-dimensional space. Instead, visualization techniques such as GPFA are required to give a parsimonious and human understandable low-dimensional projection. Thus, ongoing and future work has relied on neural trajectory visualization to make scientific hypotheses, highlighting the scientific merit of these algorithms in understanding motor cortical processing.

Second, it is important to discuss future work that may use neural trajectory algorithms for more than visualization purposes. The GPFA algorithm of Chapter 5, by definition, partitions the covariance of the high-dimensional neural data into shared network variance (the low-dimensional neural trajectory space that is mapped up to high dimensions by the matrix C in Eq. 5.1) and individual-neuron "private" variance (noise, that which is not shared across neurons, as captured in the matrix R

in Eq. 5.1). By this construction, the algorithm trades off between explaining data with low-dimensional, smooth trajectories and high-dimensional private noise. This fundamental trade-off is discussed in Chapter 5 as a benefit of GPFA vs. two-stage methods (such as smoothing and subsequently applying PCA).

Importantly, this construction means that the algorithm discovers smooth, low-dimensional commonalities between many neurons. With electrode arrays, researchers record up to a few hundred motor cortical neurons simultaneously. However, very few researchers believe that those hundreds of neurons represent hundreds of completely independent degrees of freedom, and few believe that the activity of those neurons is exclusively task-relevant. Instead, many researchers believe that neural computation is done with fewer degrees of freedom, and that these neurons have both redundancy and non-task-related activity that can be considered as noise. To return then to neural trajectory algorithms, this is precisely the coordinated, low-dimensional projections that methods like GPFA find.

Accordingly, researchers may want to focus not on the high-dimensional, noisy data, but rather on the low-dimensional trajectories that will hopefully better reflect the computational mechanisms actually at play in the motor system. Of course, such a statement comes with many caveats, including controls to ensure that the neural trajectory algorithm is "finding something real." This concern existed in earlier neural trajectory work (Yu et al., 2006) that led to the more flexible GPFA algorithm. Current work (with Mark Churchland and Byron Yu) uses the GPFA neural trajectory algorithm to show, among other findings, that outlier behavioral trials are also outliers in this low-dimensional neural space. Importantly, this work finds that this outlier effect is more pronounced in the low-dimensional, smoothed space than it is in the high-dimensional space, indicating that the algorithm is in fact correctly identifying a meaningful projection of the data. Finally, future work just in the early stages of development (with Mark Churchland) is investigating the ability for low-dimensional projections in premotor cortex to explain data from motor cortex, which is further pushing the importance of denoising and reducing high-dimensional neural data to a more compact space where computation is meaningfully carried out. These preliminary ideas are certainly less developed than the body of this dissertation, but they point to the analytical and scientific importance of neural trajectory approaches in general.

I include that discussion to highlight my opinion that neural trajectory algorithms are and will continue to be valuable algorithms for data visualization, driving scientific hypotheses, and moving towards a real understanding of the computational mechanisms at play in the motor system. The progress of Part I, in particular its culmination in the GPFA algorithm of Chapter 5, is only a first step towards these important analytical and scientific goals. In the field's general aim to understand motor cortical processing, a wealth of interesting and valuable algorithmic challenges remain.

9.2 Part II

In the second part (neural prosthetic systems), we developed a novel optimization algorithm which allowed meaningful performance improvements in a neural communications prosthesis by placing the reach targets in a principled way (Chapter 6). We showed in simulated data that this algorithm should improve performance over a wide range of neural populations, and we confirmed performance improvements in real experimental data by having a monkey reach to both canonical and optimized target placements. Chapter 7 then turned to firing rate estimation: another area of prosthetic design that has been discussed as an opportunity for algorithms to lead to performance improvements. We reviewed the many preexisting methods for estimating firing rates, and we found that, despite different firing rate estimators producing very different decoded reaches, indeed there was very little overall performance difference. Chapter 8 then panned out to look across many algorithmic challenges in neural prosthetic systems, and we analyzed large experimental datasets that suggest potential reprioritization of signal processing efforts for neural prosthetic systems. I placed this work (Chapter 8) last to point to a number of exciting areas of future algorithmic and prosthetic design work.

One of those areas, closed-loop human prosthesis studies, is particularly exciting as an area of future work, in that it may allow human users to drive algorithmic development in a meaningful way going forward. I briefly expand on that idea here. This future work was pointed to in Section 8.3.5, where we argued that offline evaluation of algorithms neglects potentially important features of a real neural prosthesis,

including the user's ability to modify behavioral strategies to improve prosthetic control. Truly understanding decode performance requires the human learning machine (the brain and motor plant) to be in closed-loop with the decode algorithm. In this online, closed-loop setting, as soon as a prosthesis user sees a decoded arm reach (the action of a robotic arm or the path of a cursor on a computer screen), he/she will bring to bear all of his/her behavioral modification strategies to drive a desirable reach. As a simple example, previous studies have found that prosthetic decode error is minimized when the time bin over which neural activity is integrated (a windowed spike count in the Kalman filter) is around 280ms (Wu et al., 2006). This bin width represents the time step at which the algorithm updates its estimate of the decoded reach. However, it may be that in a closed-loop experiment, when reaches last only roughly 1000ms, the intermittent "hopping" behavior of a decoded reach will frustrate the user. Perhaps better control could be gained with a more frequent update. Indeed, the authors of Kim et al. (2008), despite their own earlier 280ms result (Wu et al., 2006), chose 50 and 100ms bin widths in their closed-loop human experiments. Importantly, this parameter choice was not optimized in their closed-loop human clinical trial (nor in any closed loop animal study), and it is our understanding that they did not do so based on trial count limitations and the overall difficulty/challenges of testing with disabled human participants. Thus, it remains unclear how this and other parameters should be set in future studies. The field must investigate the extent to which the subject can, for a given decode algorithm, engage online control strategies to improve decode performance. Closed-loop testing may suggest different priorities for algorithmic development than offline analyses. This critical question, which has not been investigated, is an exciting line of future work.

Addressing this problem is highly challenging, since fully doing so would imply validating every algorithmic choice, ideally, in a human clinical trial. Algorithmic choices include both the structure of the algorithm itself and the parameter settings that should be optimized, resulting in thousands of decode possibilities. Given the invasiveness and expense of a neural prosthetic clinical trial, this approach is clearly infeasible. The field has next turned to an appropriate animal model such as a rhesus monkey, but given the large time and financial cost of neural implants, such an approach to widespread algorithm design is still impractical. Faced with this reality, most algorithmic work has been done in offline simulated or real neural data. We

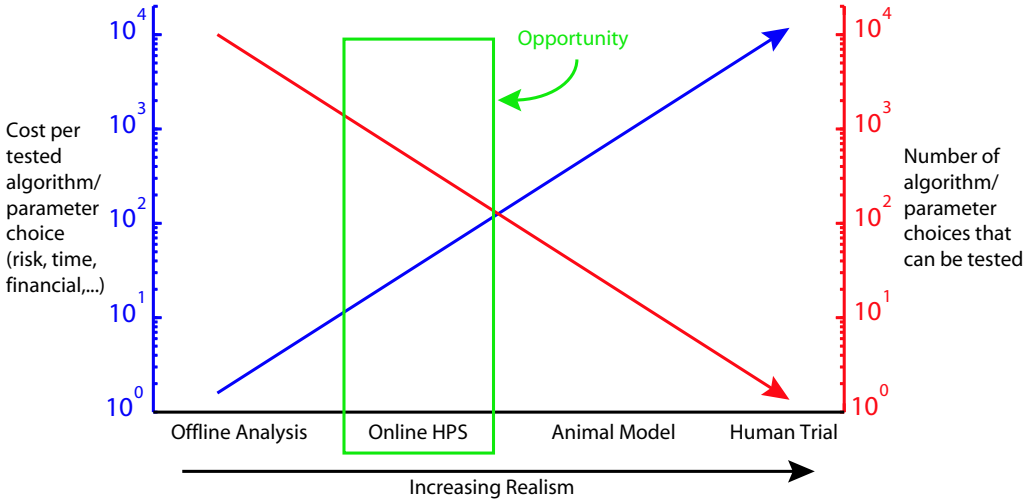


Figure 9.1: Concept figure for Online HPS opportunity. The x-axis shows four testing paradigms in terms of increasing realism. Offline data analysis is the least reasonable proxy to eventual user mode, as it entirely neglects the closed-loop control. On the other end of the spectrum is the human clinical trial, which is precisely the eventual user mode. Left axis (blue) shows the cost associated with testing each algorithm or algorithmic parameter setting. Right axis (red) shows the number of algorithm and parameter choices that are testable, given costs and other constraints.

hypothesize that a healthy human subject, using an entirely noninvasive prosthetic device driven by synthetic neural activity, can meaningfully inform the design of prosthetic decode algorithms. This synthetic neural activity can be generated based on a healthy human subject's muscle activity (recorded with EMG), arm kinematics, and other behavioral parameters. When a decoded reach is rendered to the user in a virtual display environment (and this reach will not match the user's true reach, due to model mismatch in the decoder and injected neural noise), the user will then be able to modify his/her own behavioral control strategy to drive a desired reach. Importantly, this setup allows the user to interact with the decode algorithm in closed-loop. We detail the concept of this opportunity, which we call an Online Human Prosthesis Simulator (Online HPS), in Figure 9.1. This figure shows in blue the dramatically increasing cost of testing each algorithmic choice (algorithm or just parameter setting within a single algorithm) as researchers move towards animal studies or human clinical trials. This figure also shows (in red) the corresponding dramatic decrease in the number of algorithmic choices that can be meaningfully tested. The Online HPS represents a middle ground between low cost (but low reality) offline testing and

more realistic (but quite costly) animal model and human clinical trials. Essentially, this approach is a neural prosthetic device simulator, akin to a surgical simulator (for training laproscopic and similar procedures), flight simulator, or silicon integrated circuit simulation software like SPICE, that allows human subjects to control prosthetic devices using synthetic neural activity. It also allows algorithms to be rapidly tested in closed-loop with low cost and risk. Once these algorithms are vetted in this online human simulation environment, the best performing algorithms should then be used in our own monkey and human online experiments.

I include those details to emphasize what I believe is an important way forward for neural prosthetic system design. The field has not produced a prosthetic with performance rivaling the human arm, and one reason for this may be the testing and design reality of Figure 9.1. It is our hope that pursuing this prosthetic simulator will provide a helpful bridge for algorithmic design. This work is a logical extension of Part II of this dissertation. Chapters 6 and 7 both studied individual signal processing features, and then Chapter 8 panned out to look across many possible areas for improvement. In addition to this simulation environment, we are also pursuing other areas that may hold great promise for performance improvements. Thus, in prosthetic systems, as in the scientific thrust of understanding motor cortical processing, a wealth of interesting and valuable algorithmic challenges remain.

9.3 Future Outlook

In closing, I return to the original motivation of this work. The introduction described the classic systems neuroscience paradigm and how this paradigm has been changing. If we consider the field-wide goal of a deep understanding of motor cortical processing, researchers will need to bring to bear many new technologies, including new recording/stimulation technologies and new experimental paradigms. The same is true for the broad goal of developing a neurally controlled prosthetic arm with speed and accuracy comparable to human arm. As we record more and more complicated behaviors with more and more recording modalities, along with these improvements comes a dramatic increase in data analysis needs.

This dissertation has focused on creating analytical methods to address this precise need, both for understanding motor cortical processing and for prosthetic systems. While important and significant performance improvements have been shown throughout the chapters, and while these studies have led to other interesting findings, research into these questions is by no means finished. The above sections detail exciting avenues for future work, and there are many scientific and biomedical engineering questions still unanswered. Thus, as a departure point for this dissertation, I believe that the preceeding chapters have shown solid progress towards a hugely important scientific and biomedical goal, and that this goal has careers of exciting work still remaining.

Appendix A

Publications

The preceeding chapters describe the main thrust of my research over the course of my Ph.D. work, much of which was done jointly with other researchers. I also participated as a joint contributor on a number of other projects which led to various publications that were not included in this dissertation. I include here a list of published work that lists me as an author and that was completed during the course of my thesis work at Stanford. I also include comments describing my relevant efforts for each project. Note that I do not report conference abstracts or talks, as those are quite numerous and typically subsumed by relevant conference or journal publications.

A.1 Journal Publications

1. J.P. Cunningham (2009) "Approximating Multivariate Gaussian Probabilities with Expectation Propagation" In Revision.

- This work, currently in revision for a machine learning journal publication, was presented in this document as Chapter 4. I am heavily involved in all aspects of this work.

2. J.P. Cunningham, V. Gilja, S.I. Ryu, and K.V Shenoy (2009) "Methods for estimating neural firing rates, and their application to brain-machine interfaces" **Neural Networks**, doi:10.1016/j.neunet.2009.02.004. In Press.

- This work was presented in this document as Chapter 7. I was heavily involved in all aspects of this work.

3. J.P. Cunningham, B.M. Yu, V. Gilja, S.I. Ryu, and K.V. Shenoy (2008) "Toward Optimal Target Placement for Neural Prosthetic Devices" **J. Neurophysiology**, vol. 100, no. 6, pp. 3445-3457.

- This work was presented in this document as Chapter 6. I was heavily involved in all aspects of this work.

4. C. Chang, J.P. Cunningham, and G.H. Glover (2009) "Influence of heart rate on the BOLD signal: the cardiac response function" **NeuroImage**, vol. 44, no. 3, pp. 857-869.

- This work was not covered in this dissertation. This work developed a nonparametric Bayesian method to find a biologically plausible transfer function between the rate and variation of cardiac and respiratory activity and the fMRI BOLD signal, and we showed how deconvolving this function led to greater noise reductions than previously seen. My involvement in this work was principally in the development of the nonparametric Bayesian algorithm.

5. B.M. Yu, J.P. Cunningham, G. Santhanam, S.I. Ryu, K.V. Shenoy, and M. Sahani (2009) "Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity" **J. Neurophysiology**, vol. 102, no. 1, pp. 614-635.

- This work was presented in this document as Chapter 5. I was heavily involved in all aspects of this work.

6. M.M. Churchland, B.M. Yu, J.P. Cunningham, L.P. Sugrue, M.R. Cohen, G.S. Corrado, W.T. Newsome, A.M. Clark, P. Hosseini, B.B. Scott, D.C. Bradley, M.A. Smith, A. Kohn, J.A. Movshon, K.M. Armstrong, T. Moore, S.W. Chang, L.H. Snyder, N.J. Priebe, I.M. Finn, D. Ferster, S.I. Ryu, G. Santhanam, M. Sahani, and K.V. Shenoy (2009) "Stimulus onset quashes neural variability: a widespread cortical phenomenon" In Preparation.

- This work was not covered in this dissertation. This work, currently being prepared for submission to **Nature Neuroscience**, analyzes many data sets from many different brain areas and finds that stimulus onset precipitates a drop in the variability of neural activity. This finding suggests that neural activity, which may be highly variable when unconstrained or undriven, becomes consistent across experimental trials when driven by an external stimulus such as a reach cue or reward anticipation. My involvement

in this work is in the development and use of analytical methods (this work uses the GPFA method of Chapter 5) and the preparation of the manuscript.

7. C.A. Chestek, A.P. Batista, G. Santhanam, B.M. Yu, A. Afshar, J.P. Cunningham, V. Gilja, S.I. Ryu, M.M. Churchland, and K.V. Shenoy (2007) "Single-neuron stability during repeated reaching in macaque premotor cortex" **J. Neuroscience**, vol. 27, no. 40, pp. 10742-10750.

- This work was not covered in this dissertation. This work analyzed neural activity over the course of an experimental session (typically 2-6 hours) and showed that neural activity during a consistent reaching task is stable in terms of neural tuning and firing activity. This finding importantly suggests that neurons are changing their response properties slowly, not on the course of minutes or hours. My involvement in this work was principally in the collection of neural data from monkey L.

A.2 Book Chapters

8. B.M. Yu, J.P. Cunningham, K.V. Shenoy, and M. Sahani (2007) "Neural decoding of movements: From linear to nonlinear trajectory models" In **Neural Information Processing (ICONIP 2007)**, Part I, vol. 4984, (M. Ishikawa, K. Doya, H. Miyamoto, and T. Yamakawa, eds.), pp. 586-595. ISBN 978-3-540-69154-9.

- This work was not covered in this dissertation. This work analyzed run times for various neural modeling algorithms. I was involved in a supporting role in all aspects of this project.

A.3 Refereed Conference Publications

9. J.P. Cunningham, K.V. Shenoy, and M. Sahani (2008) "Fast Gaussian process methods for point process intensity estimation" In **Proc. of the 25th Ann. Int. Conf. on Machine Learning (ICML 2008)**, (Andrew McCallum and Sam Roweis, eds.), pp. 192-199.

- This work was presented in this document as Chapter 3. I was heavily involved in all aspects of this work.

10. J.P. Cunningham, B.M. Yu, K.V. Shenoy, and M. Sahani (2008) "Inferring neural firing rates from spike trains using Gaussian processes" In **Advances in Neural Information Processing Systems 20 (NIPS 20)**, (J. Platt, D. Koller, Y. Singer, and S. Roweis, eds.), (Cambridge, MA), pp. 329-336. [Selected for spotlight presentation]

- This work was presented in this document as Chapter 2. I was heavily involved in all aspects of this work.

11. C.A. Chestek*, J.P. Cunningham*, V. Gilja, P. Nuyujukian, S.I. Ryu, and K.V. Shenoy (2009) "Neural Prosthetic Systems: Current Problems and Future Directions" In **Proc. 31st Annual Conf IEEE EMBS**, In Press.

- This work was presented in this document as Chapter 8. I was particularly involved in the data analysis and paper writing aspects of this work.

12. B.M. Yu, J.P. Cunningham, G. Santhanam, S.I. Ryu, K.V. Shenoy, and M. Sahani (2009) "Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity" In **Advances in Neural Information Processing Systems 21 (NIPS 21)**, (Cambridge, MA).

- This work was subsumed by Yu et al. (2007) (item **5** above), which was presented in this document as Chapter 5. I was heavily involved in all aspects of this work.

13. J.P. Cunningham, B.M. Yu, and K.V. Shenoy (2006) "Optimal target placement for neural communication prostheses" In **Proc. 28th Annual Conf IEEE EMBS**, pp. 2912-2915.

- This work was subsumed by Cunningham et al. (2008b) (item **3** above), which was presented in this document as Chapter 6. I was heavily involved in all aspects of this work.

14. K.V. Shenoy, G. Santhanam, S.I. Ryu, A. Afshar, B.M. Yu, V. Gilja, M.D. Linderman, R.S. Kalmar, J.P. Cunningham, C.T. Kemere, A.P. Batista, M.M. Churchland, and T.H. Meng (2006) "Increasing the performance of cortically-controlled prostheses" In **Proc. 28th Annual Conf IEEE EMBS**, pp. 6652-6656.

- This work reviewed a number of developments in neural prosthetic systems. I was heavily involved in the section describing the Optimal Target Placement algorithm of Chapter 6.

Appendix B

Appendix Material for Fast Computational Methods for Rate Estimation

This appendix derives the Expectation Propagation algorithm updates for approximating the posterior distribution on intensity in a conditionally inhomogeneous gamma interval process with a Gaussian Process prior (GP IGIP), as described in Chapter 3.

B.1 Expectation Propagation Algorithmic Details

Like the Laplace approximation, Expectation Propagation (EP) is a posterior approximation method (Minka, 2001b) that creates a Gaussian approximation (or another exponential family distribution) to the true posterior. EP has been found to be superior to Laplace in many contexts (*e.g.* Kuss and Rasmussen (2005)). EP considers global posterior information via iterative local likelihood approximations, whereas Laplace uses information only at the mode of the posterior, setting that mode as the mean of the approximate posterior and the curvature at that point as the covariance. Thus, if the mode does not give an accurate summary of the posterior distribution, Laplace may be ineffective. We will not cover the details of EP here; see Rasmussen and Williams (2006) for implementation notes and further explanation of EP applied to GP.

In the context of this problem, we have a GP prior on the intensity function $\{x(t)\}$ and the conditionally IGIP likelihood. For model selection (be that modal hyperparameter selection or approximate integration over hyperparameters), we are interested in the model posterior $p(\theta \mid \mathbf{y}) \propto p(\mathbf{y} \mid \theta)p(\theta)$, which requires the intractable data evidence $p(\mathbf{y} \mid \theta) = \int_{\mathbf{x}} p(\mathbf{y} \mid \mathbf{x}, \theta)p(\mathbf{x} \mid \theta)d\mathbf{x}$. We would like to use EP to evaluate this data evidence. However, since EP makes iterative updates at each site x_i, running EP on the vector $\mathbf{x}$ is cumbersome and inherits the computational burdens previously discussed (for example, doing rank one updates to the full n-by-n covariance matrix are still necessary). Instead, we can exploit more problem specifics to make EP feasible on a much lower dimensional integral. To do so, we will step away from $\mathbf{x}$, the quantity of interest, and return to the original gamma interarrival distribution $f_z(z) \sim \Gamma(\gamma)$. We define $\mathbf{z}$ with $z_i = \int_{y_{i-1}}^{y_i} x(u)du$, and then the observed event times $\mathbf{y}$ have conditional distribution:

$$p(\mathbf{y} \mid \mathbf{z}) = \prod_{i=1}^{N} p(y_i \mid y_{i-1}, z_i) = \prod_{i=1}^{N} \frac{\gamma^\gamma}{\Gamma(\gamma)} z_i^{\gamma-1} \exp\{-\gamma z_i\}. \tag{B.1}$$

Then, we can equivalently write the data evidence, our quantity of interest, as $p(\mathbf{y} \mid \theta) = \int_{\mathbf{z}} p(\mathbf{y} \mid \mathbf{z}, \theta)p(\mathbf{z} \mid \theta)d\mathbf{z}$. Importantly, the data evidence is equivalent[1], but the integral is over the N dimensional vector $\mathbf{z}$ (number of time events), not the n dimensional integral $\mathbf{x}$ (number of time points).

Again, $\{x(t)\}$ is a GP in continuous time with a fixed mean and stationary kernel $K_x(\tau)$, that is $\{x(t)\} \sim \mathcal{N}(\mu, K_x(\tau))$. Conveniently, the vector $\mathbf{z}$ is also Gaussian distributed (since each z_i is a linear transformation of $\{x(t)\}$). Then, we have $\mathbf{z} \sim \mathcal{N}(\mathbf{m}, \Pi)$, where $\mathbf{m}_i = (y_{i-1} - y_i)\mu$. If we choose the squared exponential (SE) kernel for $K_x(\tau)$, namely:

$$K(t_i - t_j) = \sigma_f^2 \exp\left\{-\frac{\kappa}{2}(t_i - t_j)^2\right\} + \sigma_v^2 \delta_{ij}, \tag{B.2}$$

then Π will have the form:

[1]Since the gamma likelihood truncates our distribution over the nonnegative orthant, there is a minor difference in the integral over $\mathbf{x}$ and the integral over $\mathbf{z}$. This difference arises because truncating $\mathbf{z}$ is not the same as individually truncating the elements of $\mathbf{x}$ that sum to $\mathbf{z}$. This minor discrepancy, we believe, is much smaller than the error introduced by including density outside the nonnegative orthant, as does the Laplace approximation.

$$\Pi = \left\{K_z(i,j)\right\}_{i,j\in\{1,\ldots,N\}}$$

$$\begin{aligned} \text{where} \quad K_z(i,j) &= \int_{y_{i-1}}^{y_i}\int_{y_{j-1}}^{j_i} K_x(u-v)dudv \\ &= \sigma_f^2\int_{y_{i-1}}^{y_i}\int_{y_{j-1}}^{y_j}\exp\left\{-\frac{\kappa}{2}(u-v)^2\right\} + \sigma_v^2\delta_{u-v}dudv. \end{aligned} \tag{B.3}$$

Define $\tilde{y}_i = y_i\sqrt{\frac{\kappa}{2}}$ and $\text{erf}(u) = \int_0^u \frac{2}{\sqrt{\pi}}\exp(-v^2)dv$. By this definition, $\int \text{erf}(u)du = u\text{erf}(u) + \frac{1}{\sqrt{\pi}}\exp(-u^2)$, which can be carried through Eq. B.3 to yield the (lengthy but computationally simple) expression:

$$\begin{aligned} K_z(i,j) &= \frac{\sigma_f^2\sqrt{\pi}}{\kappa}\Big[(\tilde{y}_i-\tilde{y}_j)\text{erf}(\tilde{y}_i-\tilde{y}_j) + \frac{1}{\sqrt{\pi}}\exp\{-(\tilde{y}_i-\tilde{y}_j)^2\} \\ &\quad - (\tilde{y}_i-\tilde{y}_{j-1})\text{erf}(\tilde{y}_i-\tilde{y}_{j-1}) - \frac{1}{\sqrt{\pi}}\exp\{-(\tilde{y}_i-\tilde{y}_{j-1})^2\} \\ &\quad - (\tilde{y}_{i-1}-\tilde{y}_j)\text{erf}(\tilde{y}_{i-1}-\tilde{y}_j) - \frac{1}{\sqrt{\pi}}\exp\{-(\tilde{y}_{i-1}-\tilde{y}_j)^2\} \\ &\quad + (\tilde{y}_{i-1}-\tilde{y}_{j-1})\text{erf}(\tilde{y}_{i-1}-\tilde{y}_{j-1}) + \frac{1}{\sqrt{\pi}}\exp\{-(\tilde{y}_{i-1}-\tilde{y}_{j-1})^2\}\Big] \\ &\quad + \sigma_v^2\Big[(y_i-y_j)_+ - (y_{i-1}-y_j)_+ \\ &\quad - (y_i-y_{j-1})_+ + (y_{i-1}-y_{j-1})_+\Big], \end{aligned} \tag{B.4}$$

where the notation $(\cdot)_+ \triangleq \max(\cdot, 0)$. It is important to note in the details above that only the event times y_i appear, and thus this covariance matrix is calculated in $\mathcal{O}(N^2)$ time and memory, and the larger n is never required.

We have now constructed the distributions $p(\mathbf{y} \mid \mathbf{z})$ and $p(\mathbf{z}) \sim \mathcal{N}(\mathbf{m}, \Pi)$. EP approximates the true posterior $p(\mathbf{z} \mid \mathbf{y})$ with the normal distribution

$$q(\mathbf{z} \mid \mathbf{y}) \triangleq \frac{1}{Z_{EP}}p(\mathbf{z})\prod_{i=1}^{N}t_i(z_i) = \mathcal{N}(\mu, \Sigma) \quad \text{where} \quad t_i(z_i) \triangleq \tilde{Z}_i\mathcal{N}\left(\tilde{\mu}_i, \tilde{\sigma}_i^2\right). \tag{B.5}$$

The EP implementation, and from that the calculation of data evidence, is typical for GP (see Rasmussen and Williams (2006)). The only step particular to this problem

is that of fitting a local (unnormalized) Gaussian to the product of the ith cavity distribution $q_{-i}(z_i)$ and the ith likelihood $p(y_i \mid y_{i-1}, z_i)$. By the standard Kullback-Leibler minimization, we must match the first and second moments (and zeroth, for good measure) of $\hat{q}(z_i) \triangleq \hat{Z}_i \mathcal{N}(\hat{\mu}_i, \hat{\sigma}_i^2)$ to the moments of $q_{-i}(z_i)p(y_i \mid y_{i-1}, z_i)$. In detail:

$$\hat{Z}_i = \int_0^\infty q_{-i}(z_i)p(y_i \mid y_{i-1}, z_i)dz_i \tag{B.6}$$

$$\hat{\mu}_i = \frac{1}{\hat{Z}_i}\int_0^\infty z_i q_{-i}(z_i)p(y_i \mid y_{i-1}, z_i)dz_i \tag{B.7}$$

$$\hat{\sigma}_i^2 = \frac{1}{\hat{Z}_i}\int_0^\infty z_i^2 q_{-i}(z_i)p(y_i \mid y_{i-1}, z_i)dz_i - \hat{\mu}_i^2. \tag{B.8}$$

Considering the first in detail:

$$\begin{aligned}
\hat{Z}_i &= \int_0^\infty \frac{1}{\sqrt{2\pi}\sigma_{-i}}\exp\Big\{-\frac{1}{2\sigma_{-i}^2}(z_i-\mu_{-i})^2\Big\}\frac{\gamma^\gamma}{\Gamma(\gamma)}z_i^{\gamma-1}\exp\Big\{-\gamma z_i\Big\}dz_i \\
&= \frac{\gamma^\gamma}{\Gamma(\gamma)}\exp\Big\{\frac{1}{2}\sigma_{-i}^2\gamma^2 - \mu_{-i}\gamma\Big\} \\
&\quad \cdot \int_0^\infty z_i^{\gamma-1}\frac{1}{\sqrt{2\pi}\sigma_{-i}}\exp\Big\{-\frac{1}{2\sigma_{-i}^2}\big(z_i - \big(\mu_{-i}-\gamma\sigma_{-i}^2\big)\big)^2\Big\}dz_i \\
&= \frac{\gamma^\gamma}{\Gamma(\gamma)}\exp\Big\{\frac{1}{2}\sigma_{-i}^2\gamma^2 - \mu_{-i}\gamma\Big\}\bar{E}_{r(z_i)}\big(z_i^{\gamma-1}\big),
\end{aligned} \tag{B.9}$$

where $\bar{E}$ represents the truncated expectation (integrating over the nonnegative half-line instead of the real line), and $r(z_i) \sim \mathcal{N}\left(\mu_{-i} - \gamma\sigma_{-i}^2, \sigma_{-i}^2\right)$. In words, the normalizing constant $\hat{Z}_i$ is the product of a constant and a truncated higher order moment (the $(\gamma-1)$th moment) of a univariate normal distribution $r(z_i)$. By the same logic as Eq. B.9, and substituting in for $\hat{Z}_i$,

$$\hat{\mu}_i = \frac{\bar{E}_{r(z_i)}\big(z_i^\gamma\big)}{\bar{E}_{r(z_i)}\big(z_i^{\gamma-1}\big)} \qquad \text{and} \qquad \hat{\sigma}_i^2 = \frac{\bar{E}_{r(z_i)}\big(z_i^{\gamma+1}\big)}{\bar{E}_{r(z_i)}\big(z_i^{\gamma-1}\big)} - \hat{\mu}_i^2. \tag{B.10}$$

Thus, the only difficult step in calculating an EP update is that of calculating high order truncated moments of a univariate normal distribution. There is no closed

form expression for non-integer moments, so we here restrict ourselves to the case of integer values of γ only. If, in a particular application, it is essential to have non-integer values of γ, these moments can be empirically calculated, at the cost of both accuracy and speed. For many applications, however, an integer γ should be adequate.

Though no simple closed form can be derived for truncated higher order integer moments of a normal distribution, we can recursively calculate these moments. We begin with the truncated moment generating function from Jawitz (2004). Given a normal distribution $u \sim \mathcal{N}(a, b^2)$, and letting $c = -\frac{a}{b\sqrt{\pi}}$, we write:

$$\bar{E}(u^M) = \frac{1}{2}\left[\sum_{k=0}^{M}\binom{M}{k}a^{M-k}(b\sqrt{2})^k\int_c^\infty\left(\frac{2}{\sqrt{\pi}}\right)v^k\exp\{-v^2\}dv\right]. \tag{B.11}$$

As suggested in Jawitz (2004), the integral in Eq. B.11 can be solved for any k in closed form using integration by parts and the $\text{erf}(\cdot)$ function as previously defined. However, it is tedious and impractical to detail the result of this integral for all reasonable integers that could be assigned to γ. Instead, for any k, we can recursively solve this integral (via two consecutive integrations by parts), and we see that:

$$\begin{aligned}
(k=0)\quad \left[\int_c^\infty\left(\frac{2}{\sqrt{\pi}}\right)v^0\exp\{-v^2\}dv\right] &= 1-\text{erf}(c), \qquad (B.12)\\
(k=1)\quad \left[\int_c^\infty\left(\frac{2}{\sqrt{\pi}}\right)v^1\exp\{-v^2\}dv\right] &= \frac{1}{\sqrt{\pi}}\exp\{-c^2\},\\
(k>1)\quad \left[\int_c^\infty\left(\frac{2}{\sqrt{\pi}}\right)v^k\exp\{-v^2\}dv\right] &= c^{k-1}\frac{1}{\sqrt{\pi}}\exp\{-c^2\}\\
&\quad +\frac{1}{2}(k-1)\left[\int_c^\infty\left(\frac{2}{\sqrt{\pi}}\right)v^{k-2}\exp\{-v^2\}dv\right].
\end{aligned}$$

The integral for any order k can be calculated using only a simple calculation and the $(k-2)$th order of the same integral. For the EP updates, we need the $(\gamma-1)$th, γth, and $(\gamma+1)$th truncated moments as in Eqs. B.9,B.10. By Eqs. B.11,B.12, we can calculate these moments precisely and in $\mathcal{O}(\gamma)$ time, which should for all reasonable purposes be instantaneous. We have shown that this detail of the EP update is exact and computationally simple (though, as is often the case with EP, care must

be taken to ensure numerical stability). Since all the other details are standard for EP, the entire EP update then has computational and memory complexity typical for EP, which is $\mathcal{O}(N^3)$ due to the Cholesky factorization required in updating the posterior covariance. Further, since the gamma likelihood is log concave in z_i, EP has only positive site updates (avoiding a known pitfall of EP, see Seeger (2002)) and has attractive convergence properties (it has been conjectured that EP with a log concave likelihood will always converge (Rasmussen and Williams, 2006)). Thus, we have developed a stable EP implementation that operates only on the number of events N instead of the larger n.

Accordingly, we can make a fast approximation of $p(\mathbf{y} \mid \theta)$ using either a Laplace approximation or EP on the transformed variable $\mathbf{z}$. In particular cases, one estimate may do better than others. In our specific application, we find that EP and Laplace perform similarly when the majority of the prior mass is in the nonnegative orthant, and that EP sometimes outperforms when this does not hold. More study is required to understand if EP offers a meaningful improvement in this setting.

Appendix C

Appendix Material for Calculating Gaussian Probabilities

The following appendices give additional details for the work discussed in Chapter 4.

C.1 Proof of Sensibility of KL Divergence

This section proves that minimizing KL divergence in our problem setting does correspond to matching the zeroth, first, and second moments of $q(\mathbf{x})$ to $p_{\mathcal{A}}(\mathbf{x})$. As minimizing KL divergence is the goal of the EP algorithm, this result shows that EPGCD is the appropriate choice to solve this probability (zeroth moment) problem. First, as defined in the main text, $p_{\mathcal{A}}$ and $q(\mathbf{x})$ do not normalize to 1. Thus, we use the general definition of KL divergence for non-negative distributions $f(\mathbf{x})$ and $g(\mathbf{x})$:

$$KL\big(f(\mathbf{x}) \parallel g(\mathbf{x})\big) = \int f(\mathbf{x}) \log \frac{f(\mathbf{x})}{g(\mathbf{x})} d\mathbf{x} \quad + \quad \int g(\mathbf{x}) d\mathbf{x} \quad - \quad \int f(\mathbf{x}) d\mathbf{x}, \qquad \text{(C.1)}$$

as in Zhu and Rohwer (1995); Minka (2005). Note that the typically-seen normalized KL divergence (*e.g.*, Cover and Thomas (1991)) is recovered when both $f(\mathbf{x})$ and $g(\mathbf{x})$ normalize to 1. We are interested in the distribution $q(\mathbf{x})$ that minimizes $KL\big(p_{\mathcal{A}}(\mathbf{x}) \parallel q(\mathbf{x})\big)$, where $p_{\mathcal{A}}(\mathbf{x})$ is defined in the main text, Eq. 4.3. To simplify this proof, as is often done when dealing with Gaussians, we will rewrite both $p(\mathbf{x})$ and $q(\mathbf{x})$ in their Exponential Family representation (see for example Bishop (2006); Seeger

(2003); we use the notation of the latter reference). This representation reparametizes main text Eq. 1, using instead the sufficient statistics vector $\phi(\mathbf{x})$, which for the Gaussians in this problem is a vector of all elements x_i and all pairs of elements $x_i x_j$. Then the mean and covariance parameters $\mathbf{m}, K$ of $p(\mathbf{x})$ are reparameterized also and become the so-called natural parameter vector $\boldsymbol{\theta}_p$. To be clear, this change is only a convenient reparameterization; no change to the distributions or to the moment-matching problem has taken place. In this form, we first redefine $p(\mathbf{x})$ (which does normalize to 1) and our distribution of interest, $q(\mathbf{x})$ (which does not normalize to 1):

$$p(\mathbf{x}) = \exp\Big\{\boldsymbol{\theta}_p^T \phi(\mathbf{x}) - \Phi(\boldsymbol{\theta}_p)\Big\}, \tag{C.2}$$

$$q(\mathbf{x}) = Z \exp\Big\{\boldsymbol{\theta}_q^T \phi(\mathbf{x}) - \Phi(\boldsymbol{\theta}_q)\Big\}, \tag{C.3}$$

where $\Phi(\boldsymbol{\theta}) = \log \int \exp\Big\{\boldsymbol{\theta}^T \phi(\mathbf{x})\Big\} d\mathbf{x}$ normalizes $p(\mathbf{x})$ to 1 and normalizes $q(\mathbf{x})$ to Z. Note that, since both $p(\mathbf{x})$ and $q(\mathbf{x})$ are Gaussian, both distributions share *the same* sufficient statistics $\phi(\mathbf{x})$ and normalizing function $\Phi(\boldsymbol{\theta})$ (this would not be true if these two distributions belonged to different Exponential Families, such as Poisson or multinomial). For convenience, we repeat here the definition of $p_{\mathcal{A}}(\mathbf{x})$:

$$p_{\mathcal{A}}(\mathbf{x}) = \begin{cases} p(\mathbf{x}) & \mathbf{x} \in \mathcal{A} \\ 0 & \text{otherwise.} \end{cases} \tag{C.4}$$

Thus, neither $p_{\mathcal{A}}(\mathbf{x})$ nor $q(\mathbf{x})$ normalize to 1. Since we seek the distribution $q(\mathbf{x})$ that minimizes $KL\big(p_{\mathcal{A}}(\mathbf{x}) \parallel q(\mathbf{x})\big)$, we must find $Z, \boldsymbol{\theta}_q$ minimizing

$$KL\big(p_{\mathcal{A}}(\mathbf{x}) \parallel q(\mathbf{x})\big) = \int p_{\mathcal{A}}(\mathbf{x}) \log \frac{p_{\mathcal{A}}(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x} + \int q(\mathbf{x}) d\mathbf{x} - \int p_{\mathcal{A}}(\mathbf{x}) d\mathbf{x} \tag{C.5}$$

$$= \int_{\mathcal{A}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x} + \int q(\mathbf{x}) d\mathbf{x} - \int_{\mathcal{A}} p(\mathbf{x}) d\mathbf{x} \tag{C.6}$$

$$= \int_{\mathcal{A}} p(\mathbf{x}) \Big\{(\boldsymbol{\theta}_p - \boldsymbol{\theta}_q)^T \phi(\mathbf{x})\Big\} d\mathbf{x} - \int_{\mathcal{A}} p(\mathbf{x}) \Big\{\Phi(\boldsymbol{\theta}_p) - \Phi(\boldsymbol{\theta}_q)\Big\} d\mathbf{x} - \log Z \int_{\mathcal{A}} p(\mathbf{x}) d\mathbf{x} + Z - \int_{\mathcal{A}} p(\mathbf{x}) d\mathbf{x}, \tag{C.7}$$

where all equations follow from standard properties of the logarithm and the definitions of $p(\mathbf{x})$ and $q(\mathbf{x})$ in Eq. C.2 and Eq. C.3 (and we have used the fact that Z does not depend on $\mathbf{x}$ to pull it out of the integrals). To minimize this KL, we first take the derivative with respect to the normalizer of $q(\mathbf{x})$, namely Z:

$$\begin{aligned} \frac{d}{dZ} KL\big(p_{\mathcal{A}}(\mathbf{x}) \parallel q(\mathbf{x})\big) &= -\frac{1}{Z}\int_{\mathcal{A}} p(\mathbf{x})d\mathbf{x} + 1 && \text{(C.8)} \\ &= 0 && \text{(C.9)} \\ \implies Z^* &= \int_{\mathcal{A}} p(\mathbf{x})d\mathbf{x}. && \text{(C.10)} \end{aligned}$$

From this we see that minimizing the KL divergence over the normalizing constant Z matches the zeroth moments, as we expected. Indeed, it is this equation that motivates the entire EPGCD approach, because it clarifies that minimizing global KL divergence will allow us to calculate the cumulative densities of Gaussians $F(\mathcal{A})$. For completeness, since the EP algorithm aims minimize global KL over the higher-order moments as well, we also can differentiate KL with respect to the natural parameters $q(\mathbf{x})$, namely $\boldsymbol{\theta}_q$:

$$\begin{aligned} \frac{d}{d\boldsymbol{\theta}_q} KL\big(p_{\mathcal{A}}(\mathbf{x}) \parallel q(\mathbf{x})\big) &= -\int_{\mathcal{A}} p(\mathbf{x})\phi(\mathbf{x})d\mathbf{x} + \int_{\mathcal{A}} p(\mathbf{x})\nabla_{\boldsymbol{\theta}_q}\Phi(\boldsymbol{\theta}_q)d\mathbf{x}. && \text{(C.11)} \\ &= -\int_{\mathcal{A}} p(\mathbf{x})\phi(\mathbf{x})d\mathbf{x} + Z^*\nabla_{\boldsymbol{\theta}_q}\Phi(\boldsymbol{\theta}_q), && \text{(C.12)} \end{aligned}$$

where we have used Z^* from Eq. C.10 and the fact that $\Phi(\boldsymbol{\theta}_q)$ does not depend on $\mathbf{x}$. By the definition of $\Phi(\cdot)$ (after Eq. C.3 above), we see that:

$$\begin{aligned} Z^*\nabla_{\boldsymbol{\theta}_q}\Phi(\boldsymbol{\theta}_q) &= Z^*\frac{\int \phi(\mathbf{x})\exp\Big\{\boldsymbol{\theta}_q^T\phi(\mathbf{x})\Big\}d\mathbf{x}}{\int \exp\Big\{\boldsymbol{\theta}_q^T\phi(\mathbf{x})\Big\}d\mathbf{x}} && \text{(C.13)} \\ &= Z^*\int \phi(\mathbf{x})\exp\Big\{\boldsymbol{\theta}_q^T\phi(\mathbf{x}) - \Phi(\boldsymbol{\theta}_q)\Big\}d\mathbf{x} && \text{(C.14)} \\ &= \int q(\mathbf{x})\phi(\mathbf{x})d\mathbf{x}. && \text{(C.15)} \end{aligned}$$

where we have set $q(\mathbf{x})$ to normalize to Z^*. Finally, returning to Eq. C.12 (and

substituting in the result from Eq. C.15), we see that:

$$\frac{d}{d\boldsymbol{\theta}_q} KL\big(p_{\mathcal{A}}(\mathbf{x}) \parallel q(\mathbf{x})\big) = -\int_{\mathcal{A}} p(\mathbf{x})\phi(\mathbf{x})d\mathbf{x} + \int q(\mathbf{x})\phi(\mathbf{x})d\mathbf{x} \quad \text{(C.16)}$$

$$= -\int p_{\mathcal{A}}(\mathbf{x})\phi(\mathbf{x})d\mathbf{x} + \int q(\mathbf{x})\phi(\mathbf{x})d\mathbf{x} \quad \text{(C.17)}$$

$$= -E_{p_{\mathcal{A}}}\Big(\phi(\mathbf{x})\Big) + E_q\Big(\phi(\mathbf{x})\Big) \quad \text{(C.18)}$$

$$\implies E_{q*}\Big(\phi(\mathbf{x})\Big) = E_{p_{\mathcal{A}}}\Big(\phi(\mathbf{x})\Big), \quad \text{(C.19)}$$

where we use $E(\cdot)$ to represent expectation with respect to the distributions $p_{\mathcal{A}}(\mathbf{x})$ and $q(\mathbf{x})$. Futher, since $\phi(\mathbf{x})$ captures first and second order statistics (elements x_i and $x_i x_j$, as above), $E_q(\phi(\mathbf{x}))$ is simply the first and second moments of $q(\mathbf{x})$ (so too for $p_{\mathcal{A}}(\mathbf{x})$). To summarize, these final equations Eq. C.10 and Eq. C.19 tell us that, to uniquely minimize the global KL divergence between the truncated distribution $p_{\mathcal{A}}(\mathbf{x})$ and the Gaussian $q(\mathbf{x})$, we do two things: first, we set Z to be the total mass (zeroth moment) of $p_{\mathcal{A}}(\mathbf{x})$; and second, we set $\boldsymbol{\theta}_q$, the natural parameters of $q(\mathbf{x})$ such that the mean (first moment) and covariance (second moment) of $q(\mathbf{x})$ equal exactly the first and second moments of $p_{\mathcal{A}}(\mathbf{x})$. Note that this proof also holds if the two distributions $p(\mathbf{x})$ and $q(\mathbf{x})$ do not have the same sufficient statistics $\phi(\mathbf{x})$; that is, $p(\mathbf{x})$ need not be Gaussian, as long as $q(\mathbf{x})$ is Gaussian. In that case, the result is then as in Eq. C.19, but both sides of the equation would have $\phi_q(\mathbf{x})$, the moments of the approximating Gaussian. However, the fact that the distributions in this problem *do* have the same sufficient statistics may help explain the rapid convergence of this algorithm to what our results indicate is very close to the global KL minimizer. Formalizing this claim into a proof is a subject of ongoing research.

As it pertains to calculating probabilities, minimizing global KL divergence calculates the zeroth moment of $p_{\mathcal{A}}(\mathbf{x})$, which is exactly $F(\mathcal{A})$, the probability of interest. As EP-based methods aim to solve this KL minimization, EPGCD is the sensible choice for probability calculations.

C.2 Genz Numerical Integration Method

The Genz method (as first described in Genz (1992), but see also Genz and Kwong (2000); Genz and Brentz (1999, 2002); Genz (2004)) is a sophisticated numerical integration method for calculating multivariate probabilities. The core idea in Genz (1992) is to make a series of transformations of the region $\mathcal{A}$ and the Gaussian $p(\mathbf{x})$ with the hope that this transformed region can be accurately integrated numerically. This approach performs three transformations to $F(\mathcal{A})$. It first whitens the Gaussian integrand $p(\mathbf{x})$ (via a Cholesky factorization of K, which changes the integration bounds), and secondly it removes the integrand altogether by transforming the integration bounds with a function using the cdf and inverse cdf of a univariate Gaussian. Finally, a further transformation is done (using points uniformly drawn from the $[0, 1]$ interval) to set the integration region to the unit box (in $\mathbb{R}^n$). Once this is done, Genz (1992) makes intelligent choices on dimension ordering to improve accuracy. With all orderings and transformations completed, numerical integration is carried out over the unit hypercube, using either quasi-random integration rules (Niederreiter, 1972; Cranley and Patterson, 1976) or lattice-point rules (Cranley and Patterson, 1976; Nuyens and Cools, 2004). The original algorithm (Genz, 1992) reported, "it is possible to reliably compute moderately accurate multivariate normal probabilities for practical problems with as many as ten variables." Further developments including Genz and Kwong (2000); Genz and Brentz (1999, 2002); Genz (2004) have improved the algorithm to the performance available in Table 4.1. Our use of this algorithm was enabled by the author's MATLAB code `QSILATMVNV`. This function runs a vectorized, lattice-point version of the algorithm from (Genz, 1992), which we found always produced lower error estimates (this algorithm approximates the accuracy of its result to the true probability) than the vectorized, quasi-random `QSIMVNV` (with similar run-times). We also found that these vectorized versions had lower error estimates and significantly faster run-times than the non-vectorized `QSIMVN`. Thus, all results shown as the "Genz method" were gathered using `QSILATMVNV`, available at the time of this report at: `http://www.math.wsu.edu/faculty/genz/software/software.html`. The software allows the user to define the number of lattice points used in the evaluation. To calculate the high accuracy estimate (our proxy for $F(\mathcal{A})$), we used 5×10^5 points (5×10^4 for $n = 500, 1000$). For the "Fast Genz" and "Accurate Genz" methods to which EPGCD was compared, we used 50 and 5000 points, respectively. We found

that many fewer points increased the estimate variance considerably without run-time benefit, and many more points increased run-time with little improvement to the estimates. We note also that several statistical software packages have bundled some version of the Genz method into their product (*e.g.*, `mvncdf` in MATLAB, `pmvnorm` in R). Particular choices and calculations outside the literature are typically made in those implementations (for example, MATLAB does not allow $n > 25$), so we have focused on just the code available from the author himself.

C.3 Choosing Random Regions, Means, and Covariances

In Table 4.1, for any dimension n, we used the following procedure to choose a random, n-dimensional Gaussian (defined by its mean $\mathbf{m}$ and covariance K) and a random probability region $\mathcal{A}$ (as in Fig. 4.1). For the results in Table 4.1, this procedure was repeated 1000 times at each n (100 times at $n = 500$ and $n = 1000$, due to the burden of calculating the high accuracy Genz method probabilities at these dimensions). First, we randomly drew one positive-valued n-vector from an exponential with mean 10 ($\lambda = 0.1$, n independent draws from this distribution), and we call the corresponding diagonal matrix (with this vector on the diagonal) the matrix of eigenvalues S. We then randomly draw an $n \times n$ orthogonal matrix U (orthoganilizing any random matrix with the singular value decomposition suffices), and we form the Gaussian covariance matrix $K = USU^T$. This procedure produces a good spread differing covariances K (with quite different eigenvalue spectra and condition numbers, as determined by the exponential draws of the eigenvalues in S). We note that this procedure produced a more interesting range of K (in particular, a better spread of condition numbers) than using, for example, draws from a Wishart distribution (Bishop, 2006). We then set the mean of the Gaussian $\mathbf{m} = 0$ without loss of generality (note that, were $\mathbf{m} \neq 0$, we could equivalently pick the region $\mathcal{A}$ to be shifted by $\mathbf{m}$, and then the problem would be unchanged; instead, we leave $\mathbf{m} = 0$, and we allow the randomness of $\mathcal{A}$ to suffice).

Now that we have the Gaussian $p(\mathbf{x}) = \mathcal{N}(\mathbf{m}, K)$, we must define the region $\mathcal{A}$ for the probability $F(\mathcal{A})$. The hyperrectangle $\mathcal{A}$ can be defined by two n-vectors: the upper and lower bounds $\mathbf{u}$ and $\mathbf{l}$. To make these vectors, We first randomly drew a

point from the Gaussian $p(\mathbf{x})$ and defined this point as an interior point of $\mathcal{A}$. We then added and subtracted randomly chosen lengths to each dimension of this point, calling the larger and smaller points the bounds u_i and l_i, respectively. These random lengths were chosen uniformly with range proportional to the dimension of the data, which was done to produce interesting cumulative densities on the range of 10^{-8} to 1 (otherwise, as the dimension n grows, all probabilities for a fixed region length will get smaller and smaller.

With $\mathbf{u}, \mathbf{l}, \mathbf{m}$, and K defined, we now have a randomly chosen $\mathcal{A}$ and the Gaussian $p(\mathbf{x})$, and we can test the probability methods as previously described. This procedure produces a variety of Gaussians and regions, so we believe our results suggest strongly that any valid Gaussian will produce similar results.

Appendix D

Appendix Material for Gaussian Process Factor Analysis

The following appendices give additional details for the work discussed in Chapter 5.

D.1 GPFA Model Fitting

This section details how the parameters of the GPFA model are fit using the EM algorithm, as well as the associated computational requirements.

E-Step. The E-step computes the relative probabilities $P(X \mid Y)$ of all possible neural trajectories X given the observed activity Y, using the most recent parameter estimates. We will first find the joint distribution of X and Y, which is Gaussian by definition. The desired conditional distribution $P(X \mid Y)$ is therefore also Gaussian and can then be obtained using the basic result of conditioning for jointly Gaussian random variables.

Eqs. 5.1 and 5.2 can be re-expressed as

$$\bar{\mathbf{x}} \sim \mathcal{N}\left(\mathbf{0}, \bar{K}\right) \tag{D.1}$$

$$\bar{\mathbf{y}} \mid \bar{\mathbf{x}} \sim \mathcal{N}\left(\bar{C}\bar{\mathbf{x}} + \bar{\mathbf{d}}, \bar{R}\right), \tag{D.2}$$

where $\bar{\mathbf{x}} = \left[\mathbf{x}'_{:,1} \ \ldots \ \mathbf{x}'_{:,T}\right]' \in \mathbb{R}^{pT \times 1}$ is a concatenation of the columns of X, and $\bar{\mathbf{y}} = \left[\mathbf{y}'_{:,1} \ \ldots \ \mathbf{y}'_{:,T}\right]' \in \mathbb{R}^{qT \times 1}$ is a concatenation of the columns of Y. The block

diagonal matrices $\bar{C} \in \mathbb{R}^{qT \times pT}$ and $\bar{R} \in \mathbb{R}^{qT \times qT}$ comprise T blocks of C and R, respectively. The vector $\bar{\mathbf{d}} \in \mathbb{R}^{qT \times 1}$ is a concatenation of T copies of $\mathbf{d}$. The covariance matrix

$$\bar{K} = \begin{bmatrix} \bar{K}_{11} & \dots & \bar{K}_{1T} \\ \vdots & \ddots & \vdots \\ \bar{K}_{T1} & \dots & \bar{K}_{TT} \end{bmatrix} \in \mathbb{R}^{pT \times pT} \tag{D.3}$$

comprises blocks $\bar{K}_{t_1 t_2} = \mathbf{diag}\left\{K_1(t_1, t_2), \dots, K_p(t_1, t_2)\right\} \in \mathbb{R}^{p \times p}$, where the **diag** operator returns a diagonal matrix whose non-zero elements are given by its arguments, $K_i(t_1, t_2)$ is defined in Eq. 5.3, and $t_1, t_2 = 1, \dots, T$. One can interpret $\bar{K}_{t_1 t_2}$ as the covariance of the neural states at timepoints t_1 and t_2. From Eqs. D.1 and D.2, the joint distribution of $\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$ can be written

$$\begin{bmatrix} \bar{\mathbf{x}} \\ \bar{\mathbf{y}} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \bar{\mathbf{d}} \end{bmatrix}, \begin{bmatrix} \bar{K} & \bar{K}\bar{C}' \\ \bar{C}\bar{K} & \bar{C}\bar{K}\bar{C}' + \bar{R} \end{bmatrix} \right). \tag{D.4}$$

Using the basic result of conditioning for jointly Gaussian random variables[1],

$$\bar{\mathbf{x}} \mid \bar{\mathbf{y}} \sim \mathcal{N}\left(\bar{K}\bar{C}' \left(\bar{C}\bar{K}\bar{C}' + \bar{R}\right)^{-1} (\bar{\mathbf{y}} - \bar{\mathbf{d}}),\ \bar{K} - \bar{K}\bar{C}' \left(\bar{C}\bar{K}\bar{C}' + \bar{R}\right)^{-1} \bar{C}\bar{K} \right). \tag{D.5}$$

Thus, the extracted neural trajectory is

$$E\left[\bar{\mathbf{x}} \mid \bar{\mathbf{y}}\right] = \bar{K}\bar{C}' \left(\bar{C}\bar{K}\bar{C}' + \bar{R}\right)^{-1} (\bar{\mathbf{y}} - \bar{\mathbf{d}}). \tag{D.6}$$

From Eq. D.4, the data likelihood $P(Y)$ can be easily computed since

$$\bar{\mathbf{y}} \sim \mathcal{N}\left(\bar{\mathbf{d}}, \bar{C}\bar{K}\bar{C}' + \bar{R}\right). \tag{D.7}$$

M-Step. The M-step involves maximizing $\mathcal{E}(\theta) = E\left[\log P(X, Y \mid \theta)\right]$ with respect to the parameters $\theta = \{C, \mathbf{d}, R, \tau_1, \dots, \tau_p\}$. The expectation in $\mathcal{E}(\theta)$ is taken with respect to the distribution $P(X \mid Y)$ found in the E-step, given in Eq. D.5. While this is a joint optimization with respect to all parameters in θ, it turns out that their optimal values are dependent on only a few or none of the other parameters, as shown

[1] Since $\bar{\mathbf{x}}$ is obtained by reshaping X, they contain the same elements. The same is true for $\bar{\mathbf{y}}$ and Y. Thus, $P(\bar{\mathbf{x}} \mid \bar{\mathbf{y}})$ is equivalent to $P(X \mid Y)$.

below. For clarity, we first define the following notation

$$\begin{aligned}
\langle \mathbf{x}_{:,t} \rangle &= E\left[\mathbf{x}_{:,t} \mid Y\right] \in \mathbb{R}^{p \times 1} && \text{for} \quad t = 1, \ldots, T \\
\langle \mathbf{x}_{:,t}\, \mathbf{x}'_{:,t} \rangle &= E\left[\mathbf{x}_{:,t}\, \mathbf{x}'_{:,t} \mid Y\right] \in \mathbb{R}^{p \times p} && \text{for} \quad t = 1, \ldots, T \\
\langle \mathbf{x}'_{i,:}\, \mathbf{x}_{i,:} \rangle &= E\left[\mathbf{x}'_{i,:}\, \mathbf{x}_{i,:} \mid Y\right] \in \mathbb{R}^{T \times T} && \text{for} \quad i = 1, \ldots, p
\end{aligned}$$

where these expectations can be obtained from Eq. D.5.

Maximizing $\mathcal{E}(\theta)$ with respect to C and $\mathbf{d}$ yields

$$\begin{bmatrix} C & \mathbf{d} \end{bmatrix} = \left(\sum_{t=1}^{T} \mathbf{y}_{:,t} \cdot \begin{bmatrix} \langle \mathbf{x}_{:,t} \rangle' & 1 \end{bmatrix} \right) \left(\sum_{t=1}^{T} \begin{bmatrix} \langle \mathbf{x}_{:,t}\, \mathbf{x}'_{:,t} \rangle & \langle \mathbf{x}_{:,t} \rangle \\ \langle \mathbf{x}_{:,t} \rangle' & 1 \end{bmatrix} \right)^{-1}, \tag{D.8}$$

which does not depend on any of the other parameters. The update for R is

$$R = \frac{1}{T} \cdot \mathbf{diag} \left\{ \sum_{t=1}^{T} \left(\mathbf{y}_{:,t} - \mathbf{d}\right) \left(\mathbf{y}_{:,t} - \mathbf{d}\right)' - \left(\sum_{t=1}^{T} \left(\mathbf{y}_{:,t} - \mathbf{d}\right) \langle \mathbf{x}_{:,t} \rangle' \right) C' \right\}, \tag{D.9}$$

where the **diag** operator zeros all off-diagonal elements of its argument. The new values of C and $\mathbf{d}$ found in Eq. D.8 should be used in Eq. D.9. Note that the updates for C, $\mathbf{d}$, and R have the same analytic form as for FA, except that the sums here are taken over different timepoints rather than different datapoints in the case of FA.

Although there is no analytic form for the timescale updates, they can be obtained using any gradient optimization technique. The gradient of $\mathcal{E}(\theta)$ with respect to τ_i $(i = 1, \ldots, p)$ is

$$\frac{\partial \mathcal{E}(\theta)}{\partial \tau_i} = \mathbf{tr} \left(\left[\frac{\partial \mathcal{E}(\theta)}{\partial K_i} \right]' \frac{\partial K_i}{\partial \tau_i} \right), \tag{D.10}$$

where

$$\begin{aligned}
\frac{\partial \mathcal{E}(\theta)}{\partial K_i} &= \frac{1}{2} \left(-K_i^{-1} + K_i^{-1} \langle \mathbf{x}'_{i,:}\, \mathbf{x}_{i,:} \rangle K_i^{-1} \right) \\
\frac{\partial K_i(t_1, t_2)}{\partial \tau_i} &= \sigma_{f,i}^2 \cdot \frac{(t_1 - t_2)^2}{\tau_i^3} \cdot \exp\left(-\frac{(t_1 - t_2)^2}{2 \cdot \tau_i^2} \right).
\end{aligned}$$

As in *Methods*, $K_i(t_1, t_2)$ denotes the (t_1, t_2)th entry of K_i and $t_1, t_2 = 1, \ldots, T$. Note that Eq. D.10 does not depend on the other $p - 1$ timescales, nor the other model

parameters. Thus, each of the timescales is can be optimized individually. Because the timescales must be non-negative, the optimization should be performed under the constraint that $\tau_i \geq 0$. This constrained optimization problem can be converted into an equivalent unconstrained optimization problem by optimizing with respect to $\log \tau_i$ (which can be positive or negative) rather than τ_i using a change of variable.

The derivations in this section assume a single time series (corresponding to a single experimental trial) with T timepoints. We typically want to learn the model parameters θ based on multiple time series, each with a possibly different T. The parameter update equations above can be extended in a straightforward way to accomodate multiple time series. Instead of optimizing $\mathcal{E}(\theta)$ for a single time series, we optimize their sum $\sum_n \mathcal{E}_n(\theta)$ across all time series indexed by n. Equations analogous to Eqs. D.8–D.10 can be derived by considering $\partial \left(\sum_n \mathcal{E}_n(\theta)\right) / \partial\theta$ rather than $\partial\mathcal{E}(\theta)/\partial\theta$. This assumes that the time series are independent, given the model parameters. In other words, there is no constraint built into the model that similar neural trajectories should be obtained on different trials. However, the neural trajectories are assumed to lie within the same low-dimensional state space with the same timescales.

Parameter Initialization and Local Optima. Because EM is an iterative algorithm that is guaranteed to converge to a local optimum, the values at which the parameters are initialized are important. Recall that the neural trajectories extracted by GPFA can be viewed as a compromise between the low-dimensional FA projection of each datapoint and the desire to string them together using a smooth function over time. Under this view, we initialized the parameters C, $\mathbf{d}$, and R using FA, which provides dimensionality reduction, but no smoothness over time. We then allowed GPFA to refine these estimates to obtain smooth neural trajectories. The degree of smoothness is defined by the timescales τ_i, which also need to be initialized. We fit the GPFA model starting at four different timescales: 50, 100, 150, 200 ms. In each case, all $p = 15$ timescales were initialized to the same value. Fig. D.1 shows the resulting learned timescales for each initialization. Although the learned timescales were initialization-dependent, their distributions were similar. In each case, there was one learned timescale around 525 ms, one or two around 300 ms, and the others in the range 40–180 ms. As indicated by the arrows in Fig. D.1, the mean of the 15

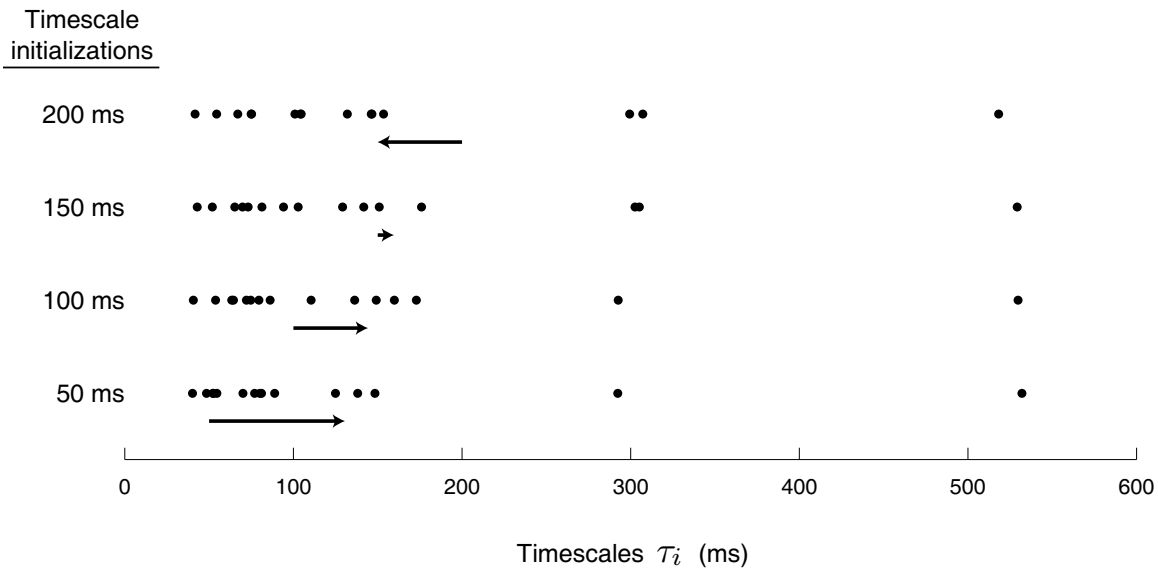


Figure D.1: Learned GPFA timescales τ_i $(i = 1, \ldots, 15)$ after 500 EM iterations starting at four different initial values: 50, 100, 150, 200 ms. Arrows denote how the mean of the 15 timescales changed between their initial and learned values. These results are based on the same data used in Figs. 5.5–5.8. For the 100 ms initialization, each of the learned timescales corresponds to a different panel in Fig. 5.6.

learned timescales ranged from 125 to 155 ms. Furthermore, the resulting training data likelihoods, as well as the extracted orthonormalized neural trajectories, were very similar in the four cases (results not shown). Unless otherwise specified, all results in this work are based on initializing the timescales to 100 ms and running EM for 500 iterations. We also reran the analysis in Fig. D.1 using 2000 EM iterations to verify that there are true local optima in the space of timescales. While other parameter initializations are possible (e.g., starting at random values with multiple restarts), we found that FA provided a sensible and effective initialization.

Because we seek to extract smooth neural trajectories, we fixed the GP noise variances $\sigma^2_{n,i}$ to a small value (10^{-3}) for all results shown in this work. Larger values of $\sigma^2_{n,i}$ generally yield neural trajectories that are less smooth. We also considered learning $\sigma^2_{n,i}$ from the data, where each state dimension indexed by i can have a different GP noise variance. This involves finding the gradient of $\mathcal{E}(\theta)$ with respect to $\sigma^2_{n,i}$ (similar to Eq. D.10) and taking gradient steps in the joint space of $\sigma^2_{n,i}$ and τ_i for each i during the M-step. If the $\sigma^2_{n,i}$ are initialized to 10^{-3}, their learned values (after 2000 EM iterations) remain on the order of 10^{-3}, yielding a similar training

data likelihood and prediction error as when the $\sigma^2_{n,i}$ are fixed to 10^{-3}.

Computational Requirements. This section details the computation time required to fit the GPFA model parameters and to extract neural trajectories. The computation time is a function of the state dimensionality p, the number of neurons q, the number of trials, the number of timesteps T in each trial, and the number of EM iterations. While fitting a GPFA model is generally computationally demanding, extracting a single neural trajectory can be very fast, as described below.

Table D.1: Time required for fitting GPFA model

Time bin width	20 ms	50 ms
$p = 3$	50 min	8 min
$p = 15$	9 hrs	45 min
T range	47–71	19–28

Results were obtained on a 2006-era Linux (FC4) 64-bit workstation with 2–4 GB of RAM running MATLAB (R14sp3, BLAS ATLAS 3.2.1 on AMD processor).

Table D.1 lists the computation time required for fitting a GPFA model for different state dimensionalities p and time bin widths. The values are based on $q = 61$ neurons, 56 trials, and 500 EM iterations. Because the absolute time duration of each trial is fixed, a larger time bin width means a smaller number of timesteps T, whose range across the 56 trials is shown in Table D.1. A naive implementation scales as $O(q^3T^3)$ due to the matrix inversion in Eq. D.5. The matrix inversion lemma can be applied to reduce the computational load to $O(p^3T^3)$. Overall, the most costly operations in fitting the GPFA model are the matrix inversion and multiplications in Eq. D.5, as well as the iterative gradient optimization (Eq. D.10) of the timescales.

There are several ways in which computation time can be reduced for the same state dimensionality p, number of neurons q, and number of trials. First, if different trials have the same T, the costly matrix inversion and multiplications in Eq. D.5 can be reused. The computation time can be drastically reduced if all or many trials have the same T. In Table D.1, many trials had different T and, therefore, did not take full advantage of this savings. Second, depending on how crucial time resolution

is, one might consider using a larger time bin width, thereby reducing T. Increasing the time bin width has the additional benefit that it increases the number of trials with the same T. As shown in Table D.1, increasing the time bin width from 20 ms to 50 ms reduced the computation time from 9 hours to 45 minutes for $p = 15$. Third, depending on how quickly the data likelihood Eq. D.7 converges, one may need fewer (or more) than 500 EM iterations. The computation time scales linearly with the number of EM iterations. Fourth, approximate techniques can be applied to reduce computation time (Teh et al., 2005; Cunningham et al., 2008a). In this work, we perform all computations exactly, without the potential speedups of approximate techniques.

Once the GPFA model parameters are learned, extracting a single neural trajectory (Eq. D.6) can be very fast, given the appropriate precomputation. In particular, the expensive matrix inversion and multiplications in Eq. D.6 can be precomputed for each T. Depending on the values of p and T, the time required for this precomputation ranges from a few milliseconds to a few seconds for each T. Once the precomputation is finished, extracting a single neural trajectory takes 2.5 ms for $p = 15$ and $T = 71$ (the most computationally-demanding trajectory in our dataset), and less time for smaller values of p and T. It is readily possible to envision having single-trial, low-dimensional visualizations (as extracted by GPFA) appear during the inter-trial interval of behaving animal experiments (< 1 second) using standard PC hardware and Matlab.

D.2 Computing Prediction Error

For GPFA, we first fit the model parameters $\theta = \{C, \mathbf{d}, R, \tau_1, \ldots, \tau_p\}$ using the EM algorithm to the training data. We show here how to evaluate model goodness-of-fit by applying these learned parameters to data not used for model fitting. As described in *Methods*, we seek to predict the activity of a neuron given the activity of all other $(q-1)$ recorded neurons. Let $\bar{\mathbf{y}}_j \in \mathbb{R}^{T\times 1}$ be the activity of neuron j and $\bar{\mathbf{y}}_{-j} \in \mathbb{R}^{(q-1)T\times 1}$ be the activity of the other $(q-1)$ neurons across all T timepoints, where $j = 1, \ldots, q$. In other words, $\bar{\mathbf{y}}_j$ is equal to the transpose of the jth row of Y, while $\bar{\mathbf{y}}_{-j}$ comprises all but the jth row of Y. The model prediction $\hat{\mathbf{y}}_j \in \mathbb{R}^{T\times 1}$ for neuron j is defined as $E\left[\bar{\mathbf{y}}_j \mid \bar{\mathbf{y}}_{-j}\right]$. Because $\bar{\mathbf{y}}_j$ and $\bar{\mathbf{y}}_{-j}$ are jointly Gaussian by

definition, the model prediction can be computed analytically.

We first define sparse binary matrices $B_j \in \{0,1\}^{T \times qT}$ and $B_{-j} \in \{0,1\}^{(q-1)T \times qT}$ such that $\bar{\mathbf{y}}_j = B_j \bar{\mathbf{y}}$ and $\bar{\mathbf{y}}_{-j} = B_{-j} \bar{\mathbf{y}}$. Multiplication by B_j and B_{-j} can be viewed as picking out the elements in $\bar{\mathbf{y}}$ corresponding to neuron j and to all other neurons, respectively. Using Eq. D.7,

$$\begin{bmatrix} \bar{\mathbf{y}}_j \\ \bar{\mathbf{y}}_{-j} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} B_j \bar{\mathbf{d}} \\ B_{-j} \bar{\mathbf{d}} \end{bmatrix}, \begin{bmatrix} B_j \Sigma B_j' & B_j \Sigma B_{-j}' \\ B_{-j} \Sigma B_j' & B_{-j} \Sigma B_{-j}' \end{bmatrix} \right), \tag{D.11}$$

where $\Sigma = \bar{C} \bar{K} \bar{C}' + \bar{R}$ is introduced for notational clarity. Applying the basic result of conditioning for jointly Gaussian random variables,

$$\hat{\mathbf{y}}_j = E\left[\bar{\mathbf{y}}_j \mid \bar{\mathbf{y}}_{-j}\right] = B_j \bar{\mathbf{d}} + \left(B_j \Sigma B_{-j}'\right) \left(B_{-j} \Sigma B_{-j}'\right)^{-1} \left(\bar{\mathbf{y}}_{-j} - B_{-j} \bar{\mathbf{d}}\right). \tag{D.12}$$

The prediction error is defined as the sum-of-squared differences between the model prediction and the observed square-rooted spike counts across all neurons and time-points

$$\text{Prediction error} = \sum_{j=1}^{q} \left\| \hat{\mathbf{y}}_j - \mathbf{y}_{j,:}' \right\|^2, \tag{D.13}$$

where $\mathbf{y}_{j,:}'$ is the transpose of the jth row of Y.

For the two-stage methods using PCA, PPCA, or FA, the model prediction is analogous to Eq. D.12, but has a simpler form because these static dimensionality reduction techniques have no concept of time. It is important to note that the training data and the data used to compute the model prediction must be pre-smoothed in the same way (e.g., using the same kernel) for the two-stage methods. However, when evaluating the prediction error, the model prediction must be compared to *unsmoothed* square-rooted spike counts, as in Eq. D.13 for GPFA. Thus, for both the two-stage methods and GPFA, a smooth model prediction is compared to unsmoothed square-rooted spike counts.

To compute the model prediction for the reduced GPFA model, we cannot simply apply Eq. D.12. Instead, we must consider an alternate approach to computing the model prediction via the orthonormalized state space. The basic idea is that a p-dimensional orthonormalized neural trajectory is first estimated using all but the jth

neuron. Then, the activity of the jth neuron is predicted using the only the top $\tilde{p}$ dimensions ($\tilde{p} = 1, \ldots, p$) of the orthonormalized neural trajectory. The following equations formalize these concepts

$$\hat{\mathbf{y}}_j = E\left[\mathbf{y}'_{j,:} \mid \bar{\mathbf{y}}_{-j}\right] \tag{D.14}$$

$$= E_X\Big[\, E\left[\mathbf{y}'_{j,:} \mid X, \bar{\mathbf{y}}_{-j}\right] \,\Big|\, \bar{\mathbf{y}}_{-j}\Big] \tag{D.15}$$

$$= E_X\Big[\, \left(\mathbf{c}'_j X + d_j \cdot \mathbf{1}_{1\times T}\right)' \,\Big|\, \bar{\mathbf{y}}_{-j}\Big] \tag{D.16}$$

$$= \left(\mathbf{c}'_j \; E_X[X \mid \bar{\mathbf{y}}_{-j}] + d_j \cdot \mathbf{1}_{1\times T}\right)' \tag{D.17}$$

$$= \left(\mathbf{u}'_j \; DV' \; E_X[X \mid \bar{\mathbf{y}}_{-j}] + d_j \cdot \mathbf{1}_{1\times T}\right)'. \tag{D.18}$$

Eq. D.15 is obtained from Eq. D.14 by conditioning on X. In Eq. D.16, we use the fact that $\mathbf{y}'_{j,:}$ is independent of $\bar{\mathbf{y}}_{-j}$ conditioned on X. Furthermore, $\mathbf{c}'_j \in \mathbb{R}^{1\times p}$ is the jth row of C, $d_j \in \mathbb{R}$ is the jth element of $\mathbf{d}$, and $\mathbf{1}_{1\times T}$ is a $1 \times T$ vector of all ones. Eq. D.18 uses the singular value decomposition of $C = UDV'$, as described in *Methods*. Note that $\mathbf{u}'_j \in \mathbb{R}^{1\times p}$ is the jth row of U, and that $DV' \; E_X[X \mid \bar{\mathbf{y}}_{-j}]$ is the orthonormalized neural trajectory estimated using all but the jth neuron.

Eq. D.18 is an alternate approach to computing the GPFA model prediction and yields the same result as Eq. D.12. Eq. D.18 says that the model prediction for neuron j can be obtained by projecting the orthonormalized neural trajectory estimated using all but the jth neuron onto the jth axis in the high-dimensional space. Although Eq. D.18 tends to be more computationally demanding than Eq. D.12, it allows us to compute the model prediction for the reduced GPFA model. For the reduced GPFA model, Eq. D.18 is computed using only the top $\tilde{p}$ elements of $\mathbf{u}_j$ (since its elements are ordered due to orthonormalization) and setting all other elements of $\mathbf{u}_j$ to zero.

D.3 Simulation with Error Floor

In Fig. 5.5, the benefit of GPFA over competing methods appears to be small (in percentage terms) if measured in terms of distance from zero prediction error. However, zero prediction error is unachievable due to the noise present in the data to be predicted. Thus, we conducted a simulation to determine the benefit of GPFA relative to a known error floor. This must be done in simulation, since the error floor

is unknown for real neural data.

The simulated data were generated using a three-dimensional state space, where each state dimension evolved in time according to a sinusoid with a different frequency. These sinusoids were then linearly combined and mapped out into a 61-dimensional observation space according to Eq. 5.1. The independent noise was assumed to be isotropic and Gaussian across the 61 dimensions. We simulated 56 trials, each with 50 timesteps. We then applied the two-stage methods and GPFA using 4-fold cross-validation, as we did for the real neural data.

As shown in Fig. D.2 (left column), all methods correctly indicated that the data are three-dimensional, since all curves reach their minimum at $p = 3$. Since the data were generated with isotropic noise, the results for two-stage methods using PPCA and FA were nearly identical. We then more densely sampled the kernel width for $p = 3$ to find the optimal smoothing kernel width, shown in Fig. D.2 (center column). Depending on the level of independent noise, we found that GPFA yielded prediction errors (black) that were tens of percent (A: 58.5%, B: 47.9%, C: 33.9%) lower than that of the best two-stage method (green dot), relative to the error floor (orange). The error floor was computed based on the level of activity of each neuron before noise was added, shown in Fig. D.2 (right column, orange curves). The orange curves provide the theoretical limit for how well the leave-neuron-out model prediction can come to the observed data on average.

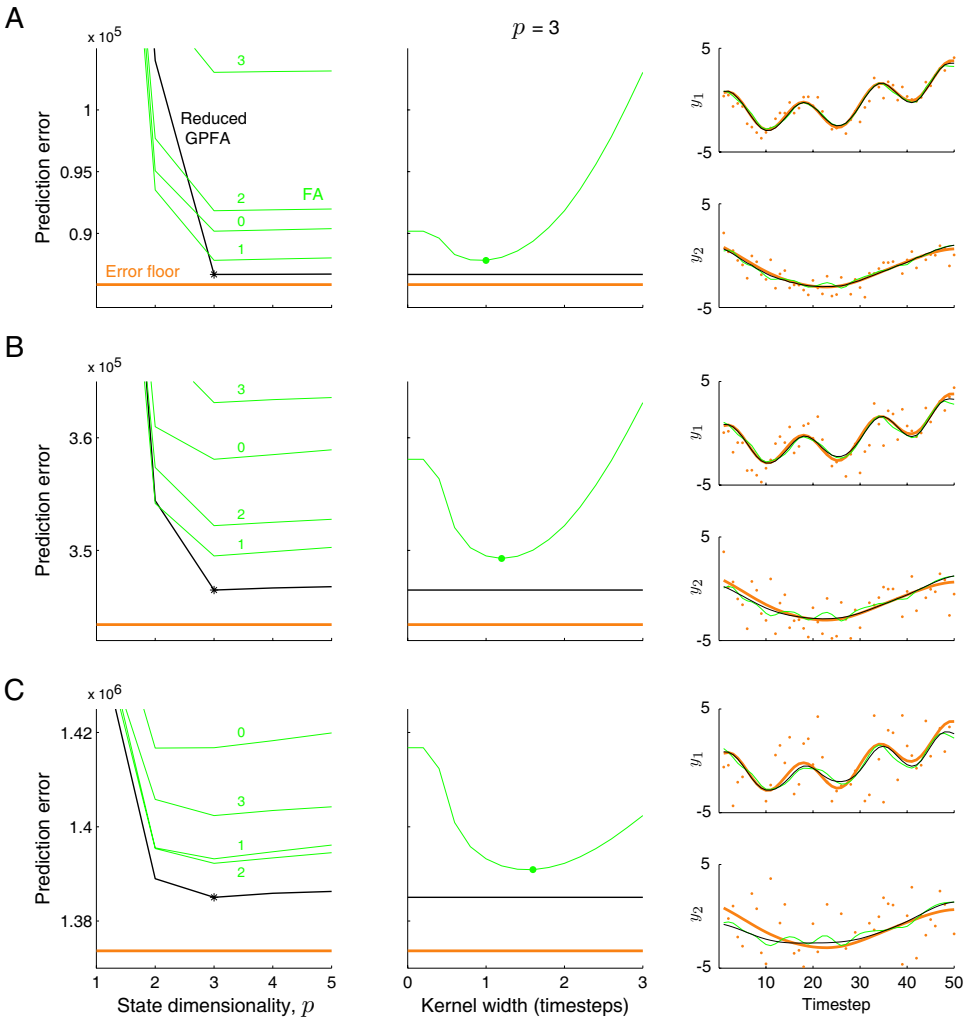


Figure D.2: Simulated data with known error floor. Each row corresponds to a different independent noise variance A: 0.5, B: 2, C: 8. Left column: Prediction errors for two-stage method with FA (green) and reduced GPFA (black), along with error floor (orange), at different state dimensionalities. Each green curve corresponds to a different kernel width (labeled are numbers of timesteps). Star indicates minimum of black curve. Center column: Denser sampling of kernel widths for $p = 3$. Minimum of green curved denoted by green dot. Right column: Each panel corresponds to an observed dimension. The same two observed dimensions are used in A, B, and C. Shown are the activity level of each neuron before noise was added (orange curves), noisy observations (orange dots), leave-neuron-out prediction using best two-stage method (green), and leave-neuron-out prediction using reduced GPFA (black).

Appendix E

Appendix Material for Optimal Target Placement

The following appendices give additional details for the work discussed in Chapter 6.

E.1 Derivation of KL Divergence for Poisson Neurons

We wish to show, for the Poisson spiking distribution $p(\mathbf{y}|m)$ (parameterized by target position $\mathbf{x}_m$, as given in Eq. 6.2), that the KL divergence has the simple closed form of Eq. 6.10. We begin by substituting Eq. 6.2 into Eq. 6.9:

$$
\begin{aligned}
KL(\mathbf{x}_m \parallel \mathbf{x}_{m'}) &= E_{\mathbf{y}|m}\left[\log\frac{p(\mathbf{y} \mid m)}{p(\mathbf{y} \mid m')}\right] && \text{(E.1)}\\
&= E_{\mathbf{y}|m}\left[\log\prod_{k=1}^{K}\frac{(f_k(\mathbf{x}_m)\Delta)^{y_k}e^{-(f_k(\mathbf{x}_m)\Delta)}y_k!}{(f_k(\mathbf{x}_{m'})\Delta)^{y_k}e^{-(f_k(\mathbf{x}_{m'})\Delta)}y_k!}\right] && \text{(E.2)}\\
&= E_{\mathbf{y}|m}\left[\sum_{k=1}^{K}\left(y_k\,\log\frac{f_k(\mathbf{x}_m)}{f_k(\mathbf{x}_{m'})} + \Delta f_k(\mathbf{x}_{m'}) - \Delta f_k(\mathbf{x}_m)\right)\right] && \text{(E.3)}
\end{aligned}
$$

where the third line follows using standard rules of exponents and logarithms (and cancelling redundant terms in both numerator and denominator). Note that all $f_k(\cdot)$ terms are constant with respect to the expectation (that is, $f_k(\cdot)$ does not depend on $\mathbf{y}$, since $\mathbf{x}_m$ or $\mathbf{x}_{m'}$ is given). Using this fact and the linearity of expectation (bringing

out the sum), we can simplify this KL divergence to:

$$KL(\mathbf{x}_m \parallel \mathbf{x}_{m'}) = \sum_{k=1}^{K}\left(\Delta f_k(\mathbf{x}_{m'}) - \Delta f_k(\mathbf{x}_m) + E_{\mathbf{y}|m}[y_k]\ \log\frac{f_k(\mathbf{x}_m)}{f_k(\mathbf{x}_{m'})}\right). \quad \text{(E.4)}$$

Finally, we note that $E_{\mathbf{y}|m}[y_k] = \Delta f_k(\mathbf{x}_m)$, and so we see:

$$KL(\mathbf{x}_m \parallel \mathbf{x}_{m'}) = \Delta\sum_{k=1}^{K}\left(f_k(\mathbf{x}_{m'}) - f_k(\mathbf{x}_m) + f_k(\mathbf{x}_m)\log\frac{f_k(\mathbf{x}_m)}{f_k(\mathbf{x}_{m'})}\right). \quad \text{(E.5)}$$

which is the form given in Eq. 6.10.

E.2 Notes on Sequential Quadratic Programming

Sequential Quadratic Programming (SQP) is a method for solving nonlinear constrained non-convex problems as in Eq. 6.13. The following gives only a brief overview of our implementation; the interested reader is referred to the excellent tutorial (Boggs and Tolle, 1996) and the general reference on convex optimization (Boyd and Vandenberghe, 2004). We note at first that SQP is a well known general method; the commonly used MATLAB (The MathWorks, Natick, MA) function for constrained optimization *fmincon* uses an SQP implementation (for medium-scale optimization problems). At low number of targets (M=2 or 4), we found this implementation to be effective. With more targets (M=16), this MATLAB implementation had convergence difficulties presumably associated with its numerical estimates of derivatives. Our implementation of this specific SQP problem, which calculates gradients and Hessians (second derivatives) in closed form, remains very effective to larger numbers of targets.

To begin, we must pose Eq. 6.13 as a standard optimization problem. To solve this *minimax* problem (*i.e.*, minimizing the maximum element of a finite set - here the $M(M-1)$ target pairs), it is common to introduce a *slack* variable t (Boyd and Vandenberghe, 2004; Gockenbach and Kearsley, 1999):

$$\begin{aligned} \underset{\boldsymbol{\chi},t}{\text{maximize}} \quad & t^2 \\ \text{subject to} \quad & KL(\mathbf{x}_m \parallel \mathbf{x}_{m'}) \geq t^2 \quad \forall\, m \neq m' \\ & \|\mathbf{x}_m\| \leq \gamma \quad \forall\, m = 1 \ldots M \end{aligned} \tag{E.6}$$

Maximizing t^2 subject to the KL constraints imposes that t^2 will have the value of the worst pairwise KL divergence. Introducing this slack variable only makes the problem algorithmically tractable; it does not change the result.

Newton's Method minimizes an (unconstrained) objective function by iterating through a series of minimizations of quadratic approximations to the objective function. Similarly, SQP minimizes a constrained objective function by iterating through a series of minimizations of constrained quadratic approximations to the original problem. These approximations are convex quadratic programs (QP) (see (Boyd and Vandenberghe, 2004) for extensive reading on QP). Each QP locally approximates the Lagrangian of Eq. E.6 at the current estimates of $\boldsymbol{\chi}$ and t ((Boggs and Tolle, 1996) justifies the choice of the Lagrangian instead of the objective itself). In our algorithm, we solve each QP quickly and accurately using the MATLAB solver *quadprog*. SQP requires a merit function to determine the length of steps that are taken in $\boldsymbol{\chi}$ and t; we used backtracking line search with an l_1 merit function. Beyond these particulars of our algorithm, the reader is again referred to (Boggs and Tolle, 1996) for many general implementation details and practical considerations.

Bibliography

Abeles, M., Bergman, H., Gat, I., Meilijson, I., Seidemann, E., Tishby, N., and Vaadia, E. (1995). Cortical activity flips among quasi-stationary states. *Proc Natl Acad Sci USA*, 92:8616–8620.

Afshar, A. (2008). Neural mechanisms of motor preparation and applications to prostheses. *Ph.D. Thesis, Stanford University*.

Aksay, E., Major, G., Goldman, M. S., Baker, R., Seung, H. S., and Tank, D. W. (2003). History dependence of rate covariation between neurons during persistent activity in an oculomotor integrator. *Cereb Cortex*, 13(11):1173–1184.

Anderson, K. D. (2004). Targeting recovery: Priorities of the spinal cord injured population. *J. Neurotrauma*, 21:1371–1383.

Arieli, A., Sterkin, A., Grinvald, A., and Aertsen, A. (1996). Dynamics of ongoing activity: Explanation of the large variability in evoked cortical responses. *Science*, 273(5283):1868–1871.

Artemiadis, P., Shakhnarovich, G., Vargas-Irwin, C., Donoghue, J., and Black, M. J. (2007). Decoding grasp aperture from motor-cortical population activity. In *Proc. of the IEEE EMBS*, pages 518–521.

Bar-Hillel, A., Spiro, A., and Stark, E. (2006). Spike sorting: Bayesian clustering of non-stationary data. *Journal of Neuroscience Methods*, 157:303–316.

Barbieri, R., Quirk, M., Frank, L., Wilson, M., and Brown, E. (2001). Construction and analysis of non-Poisson stimulus-response models of neural spiking activity. *J Neurosci Methods*, 105:25–37.

Basu, S. and Dassios, A. (2002). A Cox process with log-normal intensity. *Insurance: Mathematics and Economics*, 31:297–302.

Bathellier, B., Buhl, D. L., Accolla, R., and Carleton, A. (2008). Dynamic ensemble odor coding in the mammalian olfactory bulb: sensory information at different timescales. *Neuron*, 57:586–598.

Batista, A. P., Santhanam, G., Yu, B. M., Ryu, S. I., Afshar, A., and Shenoy, K. V. (2007). Reference frames for reach planning in macaque dorsal premotor cortex. *J Neurophysiol*, 98:966–983.

Beal, M. J., Ghahramani, Z., and Rasmussen, C. E. (2002). The infinite hidden Markov model. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Adv Neural Info Processing Sys 14*, pages 577–585. MIT Press, Cambridge, MA.

Behseta, S. and Kass, R. (2005). Testing equality of two functions using BARS. *Stats. Med*, 24:3523–3534.

Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer, New York.

Bisley, J. W. and Goldberg, M. E. (2003). Neuronal activity in the lateral intraparietal area and spatial attention. *Science*, 299(5603):81–86.

Blankertz, B., Muller, K.-R., Curio, G., Vaughan, T., Schalk, G., Wolpaw, J., Schlogl, A., Neuper, C., Pfurtscheller, G., Hinterberger, T., Schroder, M., and Birbaumer, N. (2004). The bci competition 2003: Progress and perspectives in detection and discrimination of eeg single trials. *IEEE Transactions on Biomedical Engineering*, 51:1044–1051.

Boggs, P. T. and Tolle, J. W. (1996). Sequential quadratic programming. *Acta Numerica*, pages 1–52.

Borst, A. and Theunissen, F. E. (1999). Information theory and neural coding. *Nat. Neurosci.*, 2.

Bowman, A. W. (1984). An alternative method of cross-validation for the smoothing of density estimates. *Biometrika*, 71:353–360.

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press, Cambridge.

Boyle, P. P., Lai, Y., and Tan, K. S. (2005). Pricing options using lattice rules. *N. Amer. Actuarial Journal*, 9(3).

Bradley, D. C., Chang, G. C., and Andersen, R. A. (1998). Encoding of three-dimensional structure-from-motion by primate area MT neurons. *Nature*, 392:714–717.

Briggman, K. L., Abarbanel, H. D. I., and Kristan Jr., W. B. (2005). Optical imaging of neuronal populations during decision-making. *Science*, 307(5711):896–901.

Briggman, K. L., Abarbanel, H. D. I., and Kristan Jr., W. B. (2006). From crawling to cognition: analyzing the dynamical interactions among populations of neurons. *Curr Opin Neurobiol*, 16(2):135–144.

Brockwell, A., Rojas, A., and Kass, R. (2004). Recursive Bayesian decoding of motor cortical signals by particle filtering. *J Neurophysiol*, 91(4):1899–1907.

Broome, B. M., Jayaraman, V., and Laurent, G. (2006). Encoding and decoding of overlapping odor sequences. *Neuron*, 51:467–482.

Brown, E., Barbieri, R., Ventura, V., Kass, R., and Frank, L. (2002). The time-rescaling theorem and its application to neural spike train data analysis. *Neural Comp*.

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

Brown, E. N., Kass, R. E., and Mitra, P. P. (2004). Multiple neural spike train data analysis: state-of-the-art and future challenges. *Nat Neurosci*, 7(5):456–461.

Brown, S. L., Joseph, J., and Stopfer, M. (2005). Encoding a temporally structured stimulus with a temporally structured neural representation. *Nat Neurosci*, 8(11):1568–1576.

Buccianti, A., Nardi, G., (eds, R. P., Serre, M. L., Bogaert, P., and Christakos, G. (1998). Computational investigations of bayesian maximum entropy spatiotemporal mapping. *Proc. of the Intl. Assoc. Math. Geology*.

Carmena, J., Lebedev, M., Crist, R., O'Doherty, J., Santucci, D., Dimitrov, D., Patil, P., Henriquez, C., and Nicolelis, M. (2003). Learning to control a brain-machine interface for reaching and grasping by primates. *PLoS Biology*, 1:193–208.

Carmena, J. M., Lebedev, M. A., Henriquez, C. S., and Nicolelis, M. A. (2005). Stable ensemble performance with single-neuron variability during reaching movements in primates. *J Neurosci*, 25(46):10712–10716.

Carrillo-Reid, L., Tecuapetla, F., Tapia, D., Hernández-Cruz, A., Galarraga, E., Drucker-Colin, R., and Bargas, J. (2008). Encoding network states by striatal cell assemblies. *J Neurophysiol*, 99:1435–1450.

Chapin, J. (2004). Using multi-neuron population recordings for neural prosthetics. *Nature Neuroscience*, 7:452–455.

Chestek, C., Batista, A., Santhanam, G., Yu, B., Afshar, A., Cunningham, J., Gilja, V., Ryu, S., Churchland, M., and Shenoy, K. (2007). Single-neuron stability during repeated reaching in macaque premotor cortex. *J Neurosci*, 27:10742–10750.

Chestek*, C. A., Cunningham*, J. P., Gilja, V., Nuyujukian, P., Ryu, S. I., and Shenoy, K. V. (2009). Neural prosthetic systems: Current problems and future directions. *Proc. of the 31st Annual Intl Conf of the IEEE EMBS*. In Press.

Chestek, C. A., Gilja, V., Nuyujukian, P., Kier, R. J., Solzbacher, F., Ryu, S. I., Harrison, R. R., and Shenoy, K. V. (2009). HermesC: Low-power wireless neural recording system for freely moving primates. *IEEE Trans Neural Syst Rehabil Eng*. In press.

Churchland, M. M., Santhanam, G., and Shenoy, K. V. (2006a). Preparatory activity in premotor and motor cortex reflects the speed of the upcoming reach. *J Neurophysiol*, 96(6):3130–3146.

Churchland, M. M., Yu, B. M., Ryu, S. I., Santhanam, G., and Shenoy, K. V. (2006b). Neural variability in premotor cortex provides a signature of motor preparation. *J Neurosci*, 26(14):3697–3712.

Churchland, M. M., Yu, B. M., Sahani, M., and Shenoy, K. V. (2007). Techniques for extracting single-trial activity patterns from large-scale neural recordings. *Curr Opin Neurobiol*, 17(5):609–618.

Cisek, P. and Kalaska, J. F. (2005). Neural correlates of reaching decisions in dorsal premotor cortex: Specification of multiple direction choices and final selection of action. *Neuron*, 45:801–814.

Cody, W. J. (1969). Rational Chebyshev approximations for the error function. *Math. Comp.*, pages 631–638.

Coleman, T. P. and Sarma, S. (2007). A computationally efficient method for modeling neural spiking activity with point processes nonparametrically. In *46th IEEE Conf. on Decision and Control*, pages 5800–5805.

Cook, E. P. and Maunsell, J. H. R. (2002). Dynamics of neuronal responses in macaque MT and VIP during motion detection. *Nat Neurosci*, 5(10):985–994.

Cover, T. M. and Thomas, J. A. (1991). *Elements of Information Theory*. John Wiley & Sons, New York.

Crammond, D. and Kalaska, J. (2000). Prior information in motor and premotor cortex: activity during the delay period and effect on pre-movement activity. *J Neurophysiol*, 84:986–1005.

Cranley, R. and Patterson, T. N. L. (1976). Randomization of number theoretic methods for multiple integration. *SIAM J. Numer. Anal.*, 13:904–914.

Csato, L., Opper, M., and Winther, O. (2002). TAP Gibbs free energy, belief propagation, and sparsity. In *Advances in Neural Information Processing Systems 14*. MIT Press, Cambridge, MA.

Cunningham, J. P., Gilja, V., Ryu, S. I., and Shenoy, K. V. (2009). Methods for estimating neural firing rates, and their application to brain-machine interfaces. *Neural Networks*. In Press.

Cunningham, J. P., Sahani, M., and Shenoy, K. V. (2008a). Fast Gaussian process methods for point process intensity estimation. In *Proceedings of the 25th International Conference on Machine Learning*.

Cunningham, J. P., Yu, B. M., Gilja, V., Ryu, S. I., and Shenoy, K. V. (2008b). Toward optimal target placement for neural prosthetic devices. *J Neurophysiol*, 100(6):3445–3457.

Cunningham, J. P., Yu, B. M., and Shenoy, K. V. (2006). Optimal target placement for neural communication prostheses. *Proc. of the 28th Annual Intl Conf of the IEEE EMBS*, pages 2912–2915.

Cunningham, J. P., Yu, B. M., Shenoy, K. V., and Sahani, M. (2008c). Inferring neural firing rates from spike trains using Gaussian processes. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in NIPS 20*. MIT Press, Cambridge, MA.

Czanner, G., Eden, U. T., Wirth, S., Yanike, M., Suzuki, W. A., and Brown, E. N. (2008). Analysis of between-trial and within-trial neural spiking dynamics. *J Neurophysiol*, 99:2672–2693.

Daley, D. and Vere-Jones, D. (2002). *An Introduction to the Theory of Point Processes*. Springer, New York.

Danóczy, M. and Hahnloser, R. (2006). Efficient estimation of hidden state dynamics from spike trains. In Weiss, Y., Schölkopf, B., and Platt, J., editors, *Adv Neural Info Processing Sys 18*, pages 227–234. MIT Press, Cambridge, MA.

Dayan, P. and Abbott, L. (2001). *Theoretical Neuroscience*. MIT Press, Cambridge.

de Klerk, E., Pasechnik, D. V., and Warners, J. P. (2000). On approximate graph colouring and MAX-k-CUT algorithms based on the ϑ-function.

de Lafuente, V. and Romo, R. (2005). Neuronal correlates of subjective sensory experience. *Nat Neurosci*, 8(12):1698–1703.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm (with discussion). *J R Stat Soc Ser B*, 39:1–38.

DiGiovanna, J., Mahmoudi, B., Fortes, J., Principe, J., and Sanchez, J. (2009). Coadaptive brain-machine interface via reinforcement learning. *IEEE Transactions on Biomedical Engineering*, 56(1):54–64.

DiMatteo, I., Genovese, C. R., and Kass, R. E. (2001). Bayesian curve-fitting with free-knot splines. *Biometrika*, 88(4):1055–1071.

Dodd, J. V., Krug, K., Cumming, B. G., and Parker, A. J. (2001). Perceptually bistable three-dimensional figures evoke high choice probabilities in cortical area MT. *J Neurosci*, 21(13):4809–4821.

Donoghue, J. P. (2008). Bridging the brain to the world: A perspective on neural interface systems. *Neuron*, 60:511–521.

Drezner, Z. (1994). Computation of the trivariate normal integral. *Math. of Comp.*, 63:289–294.

Drezner, Z. and Wesolowsky, G. O. (1989). On the computation of the bivariate normal integral. *J. Stat. Comp. Simul.*, 35:101–107.

Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification*. Wiley & Sons, New York.

Eden, U., Frank, L., Barbieri, R., Solo, V., and Brown, E. N. (2004). Dynamic analysis of neural encoding by point process adaptive filtering. *Neural Comp*, 16:971–998.

Eliades, S. J. and Wang, X. Q. (2008). Chronic multi-electrode neural recording in free-roaming monkeys. *J Neurosci Methods*, 172(2):201–214.

Endres, D., Oram, M., Schindelin, J., and Foldiak, P. (2008). Bayesian binning beats approximate alternatives: estimating peri-stimulus time histograms. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Adv Neural Info Processing Sys 20*. MIT Press, Cambridge, MA.

Evarts, E. (1968). Relation of pyramidal tract activity to force exerted during voluntary movement. *J Neurophysiol*, 31:14–27.

Everitt, B. S. (1984). *An introduction to latent variable models*. Chapman and Hill, London.

Faisal, A. A., Selen, L. P. J., and Wolpert, D. M. (2008). Noise in the nervous system. *Nat Rev Neurosci*, 9:292–303.

Fu, Q. G., Suarez, J. I., and Ebner, T. J. (1993). Neuronal specification of direction and distance during reaching movements in the superior precentral premotor area and primary motor cortex of monkeys. *J Neurophys*, 70:2097–2116.

Ganguli, S., Bisley, J. W., Roitman, J. D., Shadlen, M. N., Goldberg, M. E., and Miller, K. D. (2008). One-dimensional dynamics of attention and decision making in LIP. *Neuron*, 58:15–25.

Gao, Y., Black, M., Bienenstock, E., Shoham, S., and Donoghue, J. (2002). Probabilistic inference of hand motion from neural activity in motor cortex. *Advances in NIPS*, 14.

Gat, I., Tishby, N., and Abeles, M. (1997). Hidden Markov modelling of simultaneously recorded cells in the associative cortex of behaving monkeys. *Network*, 8(3):297–322.

Genz, A. (1992). Numerical computation of multivariate normal probabilities. *J. Comp. Graph. Stat.*, 1:141–149.

Genz, A. (2004). Numerical computation of rectangular bivariate and trivariate normal and t probabilities. *Stat. and Comput.*, 14:251–260.

Genz, A. and Brentz, F. (1999). Numerical computation of multivariate t probabilities with application to power calculation of multiple contrasts. *J. Stat. Comp. Simul.*, 63:361–378.

Genz, A. and Brentz, F. (2002). Comparison of methods for the computation of multivariate t probabilities. *J. Comp. Graph. Stats.*, 11:950–971.

Genz, A. and Kwong, K. S. (2000). Numerical evaluation of singular multivariate normal distributions. *J. Stat. Comp. Simul.*, 68:1–21.

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

Georgopoulos, A., Schwartz, A., and Kettner, R. (1986). Neuronal population coding of movement direction. *Science*, 233:1416–1419.

Gibbs, M. and MacKay, D. (1997). Efficient Implementation of Gaussian Processes. *Preprint*.

Gill, P., Golub, G., Murray, W., and Saunders, M. (1974). Methods for Modifying Matrix Factorizations. *Mathematics of Computation*, 28(126):505–535.

Gockenbach, M. S. and Kearsley, A. J. (1999). Optimal signal sets for non-Gaussian detectors. *SIAM Journal of Optimization*, 9(2):316–326.

Gray, A. and Moore, A. (2003). Nonparametric density estimation: Toward computationsl tractability. In *SIAM Int'l Conference on Data Mining*.

Hanes, D. P. and Schall, J. D. (1996). Neural control of voluntary movement initiation. *Science*, 274(5286):427–430.

Harrison, R. R., Watkins, P. T., Kier, R. J., Lovejoy, R. O., Black, D. J., Greger, B., and Solzbacher, F. (2007). A low-power integrated circuit for a wireless 100-electrode neural recording system. *IEEE J of Solid State Circuits*, 42:123–133.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The elements of statistical learning*. Springer.

Hatsopoulos, N., Joshi, J., and O'Leary, J. G. (2004). Decoding continuous and discrete motor behaviors using motor and premotor cortical ensembles. *J Neurophysiol*, 92:1165–1174.

Heskes, T. and Zoeter, O. (2002). Expectation propagation for approximate inference in dynamic bayesian networks. In *In Proceedings UAI*, pages 216–223.

Hickernell, F. J. and Hong, H. S. (1999). The asymptotic efficiency of randomized nets for quadrature. *Math. Comp*, 68:767–791.

Hochberg, L. R., Serruya, M. D., Friehs, G. M., Mukand, J. A., Saleh, M., Caplan, A. H., Branner, A., Chen, D., Penn, R. D., and Donoghue, J. P. (2006). Neuronal ensemble control of prosthetic devices by a human with tetraplegia. *Nature*, 442:164–171.

Horrace, W. (2005). Some results on the multivariate truncated normal distribution. *J Multivariate Analysis*, 94(1):209–221.

Horwitz, G. D. and Newsome, W. T. (2001). Target selection for saccadic eye movements: Prelude activity in the superior colliculus during a direction-discrimination task. *J Neurophysiol*, 86:2543–2558.

Hothorn, T., Hornik, K., Zeileis, A., and Wien, W. (2005). Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, 2006:651–674.

Jackson, A., Mavoori, J., and Fetz, E. E. (2007). Correlations between the same motor cortex cells and arm muscles during a trained task, free behavior, and natural sleep in the macaque monkey. *J Neurophysiol*, 97:360–374.

Jawitz, J. (2004). Moments of truncated continuous univariate distributions. *Advances in Water Resources*, 27:269–281.

Joe, H. (1995). Approximations to multivariate normal rectangle probabilities based on conditional expectations. *J. Am. Stat. Assoc.*, 90(431):957–964.

Johnson, D. (1996). Point process models of single-neuron discharges. *J Comput Neurosci*, 3:275–299.

Johnson, D. H., Gruner, C. M., Baggerly, K., and Seshagiri, C. (2001). Information-theoretic analysis of neural coding. *Journal of Computational Neuroscience*, 10:47–69.

Johnson, D. H. and Orsak, G. C. (1993). Relation of signal set choice to the performance of optimal non-Gaussian detectors. *IEEE Transactions on Communications*, 41(9):1319–1328.

Jones, L. M., Fontanini, A., Sadacca, B. F., Miller, P., and Katz, D. B. (2007). Natural stimuli evoke dynamic sequences of states in sensory cortical ensembles. *Proc Natl Acad Sci USA*, 104(47):18772–18777.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *J. Basic Engineering*, 82:35–45.

Kass, R. and Ventura, V. (2003). A spike-train probability model. *Neural Comp*, 14:5–15.

Kass, R., Ventura, V., and Brown, E. (2005). Statistical issues in the analysis of neuronal data. *J. Neurophysiol*, 94:8–25.

Kaufman, C., Ventura, V., and Kass, R. (2005). Spline-based non-parametric regression for periodic functions and its application to directional tuning of neurons. *Stat Med*, 24:2255–2265.

Kemere, C., Santhanam, G., Yu, B. M., Afshar, A., Ryu, S. I., Meng, T. H., and Shenoy, K. V. (2008). Detecting neural state transitions using hidden Markov models for motor cortical prostheses. *J Neurophysiol*, 100:2441–2452.

Kemere, C., Shenoy, K., and Meng, T. (2004). Model-based neural decoding of reaching movements: a maximum likelihood approach. *IEEE Transactions on Biomedical Engineering – Special issue on Brain-Machine Interfaces*, 51:925–932.

Kennedy, P. and Bakay, R. (1998). Restoration of neural output from a paralyzed patient by a direct brain connection. *NeuroReport*, 9:1707–1711.

Kennedy, P., Bakay, R., Moore, M., Adams, K., and Goldwaithe, J. (2000). Direct control of a computer from the human central nervous system. *IEEE Transactions on Rehabilitation Engineering*, 8:198–202.

Kerr, J. N. D. and Denk, W. (2008). Imaging in vivo: watching the brain in action. *Nat Rev Neurosci*, 9:195–205.

Kihlberg, J. K., Herson, J. H., and Schotz, W. E. (1972). Square root transformation revisited. *Appl Statist*, 21(1):76–81.

Kim, S., Sanchez, J. C., Rao, Y. N., Erdogmus, D., Carmena, J. M., Lebedev, M. A., Nicolelis, M. A. L., and Principe, J. C. (2006). A comparison of optimal mimo linear and nonlinear models for brain-machine interfaces. *J Neural Engineering*, 3:145–161.

Kim, S., Simeral, J. D., Hochberg, L. R., Donoghue, J. P., and Black, M. J. (2008). Neural control of computer cursor velocity by decoding motor cortical spiking activity in humans with tetraplegia. *J Neural Engineering*, 5:455–476.

Kipke, D. R., Shain, W., Buzsáki, G., Fetz, E., Henderson, J. M., Hetke, J. F., and Schalk, G. (2008). Advanced neurotechnologies for chronic neural interfaces: New horizons and clinical opportunities. *J Neurosci*, 28(46):11830–11838.

Koyama, S. and Kass, R. E. (2008). Spike train probability models for stimulus-drive leaky integrate-and-fire neurons. *Neural Comp*, 20.

Kulkarni, J. E. and Paninski, L. (2007). Common-input models for multiple neural spike-train data. *Network*, 18(4):375–407.

Kulkarni, J. E. and Paninski, L. (2008). State-space decoding of goal-directed movements. *IEEE Sig Proc Magazine*, 25(1):78–86.

Kuss, M. and Rasmussen, C. (2005). Assessing approximate inference for binary gaussian process classification. *Journal of Machine Learning Res.*, 6:1679–1704.

Lawrence, N. (2005). Probabilistic non-linear principal component analysis with Gaussian process latent variable models. *J Mach Learn Res*, 6:1783–1816.

Lawrence, N., Seeger, M., and Herbrich, R. (2003). Fast sparse gaussian process methods: The informative vector machine. In *Advances in Neural Information Processing Systems 15*, pages 609–616. MIT Press.

Lawrence, N. D. and Moore, A. J. (2007). The hierarchical Gaussian process latent variable model. In Ghahramani, Z., editor, *Proceedings of the 24th Annual International Conference on Machine Learning (ICML 2007)*, pages 481–488. Omnipress.

Lebedev, M., Carmena, J., O'Doherty, J., Zacksenhouse, M., Henriquez, C., Principe, J., and Nicolelis, M. (2005). Cortical ensemble adaptation to represent velocity of an artificial actuator controlled by a brain-machine interface. *J Neurosci*, 25(19):4681–4693.

Lebedev, M. A. and Nicolelis, M. A. L. (2006). Brain-machine interfaces: past, present, and future. *Trends in Neurosci*, 29.

Leopold, D. A. and Logothetis, N. K. (1996). Activity changes in early visual cortex reflect monkeys' percepts during binocular rivalry. *Nature*, 379:549–553.

Leuthardt, E. C., Miller, K. J., Schalk, G., Rao, R. P. N., and Ojemann, J. G. (2006). Electrocorticography-based brain computer interface - the seattle experience. *IEEE. Trans. on Neur. Sys. Rehabil. Eng.*, 14:194–198.

Levi, R., Varona, R., Arshavsky, Y. I., Rabinovich, M. I., and Selverston, A. I. (2005). The role of sensory network dynamics in generating a motor program. *J Neurosci*, 25(42):9807–9815.

Lewicki, M. S. (1998). A review of methods for spike sorting: the detection and classification of neural action potentials. *Network Comput. Neural Sys.*, 9.

Liao, X., Li, H., and Carin, L. (2007). Quadratically gated mixture of experts for incomplete data classification. In *Proceedings of the 24th International Conference on Machine Learning*.

Linderman, M. D., Santhanam, G., Kemere, C. T., Gilja, V., O'Driscoll, S., Yu, B. M., Afshar, A., Ryu, S. I., Shenoy, K. V., and Meng, T. H. (2008). Signal processing challenges for neural prostheses. *IEEE Signal Processing Magazine*, 25:18–28.

MacKay, D. (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press.

Mazor, O. and Laurent, G. (2005). Transient dynamics versus fixed points in odor representations by locust antennal lobe projection neurons. *Neuron*, 48:661–673.

Mehring, C., Rickert, J., Vaadia, E., deOliveira, S. C., Aertsen, A., and Rotter, S. (2003). Inference of hand movements from local field potentials in monkey motor cortex. *Nature Neuroscience*, 6(12).

Messier, J. and Kalaska, J. (2000). Covariation of primate dorsal premotor cell activity with direction and amplitude during a memorized-delay reaching task. *J Neurophysiol*, 84:152–165.

Minka, T. P. (2001a). Expectation Propagation for approximate Bayesian inference. *Uncertainty in AI*, pages 362–369.

Minka, T. P. (2001b). *A family of algorithms for approximate Bayesian inference*. MIT Ph.D. Thesis.

Minka, T. P. (2005). Divergence measures and message passing. *Microsoft Research Tech. Report MSR-TR-2005-173*.

Miura, K., Tsubo, Y., Okada, M., and Fukai, T. (2007). Balanced excitatory and inhibitory inputs to cortical neurons decouple firing irregularity from rate modulations. *J Neurosci*, 27:13802–13812.

Moller, J., Syversveen, A., and Waagepetersen, R. (1998). Log Gaussian Cox processes. *Scandanavian J. of Stats.*

Moran, D. and Schwartz, A. (1999a). Motor cortical representation of speed and direction during reaching. *J Neurophysiol.*, 82(5):2676–2692.

Moran, D. and Schwartz, A. (2000). One motor cortex, two different views. *Nature Neuroscience*, 3:963–963.

Moran, D. W. and Schwartz, A. B. (1999b). Motor cortical activity during drawing movements: Population representation during spiral tracing. *J Neurophysiol*, 82:2693–2704.

Mulliken, G. H., Musallam, S., and Andersen, R. A. (2008). Decoding trajectories from posterior parietal cortex ensembles. *J Neurosci*, 28:12913–12926.

Musallam, S., Corneil, B., Greger, B., Scherberger, H., and Andersen, R. (2004). Cognitive control signals for neural prosthetics. *Science*, 305:258–262.

Nawrot, M., Aertsen, A., and Rotter, S. (1999). Single-trial estimation of neuronal firing rates: From single-neuron spike trains to population activity. *J Neurosci Methods*, 94:81–92.

Nicolelis, M. A. L., Baccala, L. A., Lin, R. C. S., and Chapin, J. K. (1995). Sensorimotor encoding by synchronous neural ensemble activity at multiple levels of the somatosensory system. *Science*, 268(5215):1353–1358.

Niederreiter, H. (1972). On a number-theoretical integration method. *Aequationes Mathematicae*, 8:304–311.

Nirenberg, S., Carcieri, S. M., Jacobs, A. L., and Latham, P. E. (2001). Retinal ganglion cells act largely as independent encoders. *Nature*, 411.

Nuyens, D. and Cools, R. (2004). Fast component-by-component construction, a reprise for different kernels. In Niederreiter, H. and Talay, D., editors, *Monte Carlo and quasi-Monte Carlo methods*, pages 371–385. Springer-Verlag.

O'Driscoll, S., Meng, T. H., Shenoy, K. V., and Kemere, C. (2006). Neurons to silicon: Implantable prosthesis processor. *Int. Solid State Circuits Conference (ISSCC)*, pages 552–553.

Olson, C., Gettner, S., Ventura, V., Carta, R., and Kass, R. (2000). Neuronal activity in macaque supplementary eye field during planning of saccades in response to pattern and spatial cues. *J Neurophysiol*, 84:1369–1384.

Opper, M. and Winther, O. (2000). Gaussian processes for classification: Mean field algorithms. *Neural Comp.*, 12:2655–2684.

Opper, M. and Winther, O. (2001). From naive mean field theory to the TAP equations. In Opper, M. and Saad, D., editors, *Advanced Mean Field Methods: Theory and Practice*. MIT Press, Cambridge, MA.

Paciorek, C. and Schervish, M. (2003). Nonstationary covariance functions for gaussian process regression. *Advances in NIPS*, 15.

Paninski, L. (2004). Log-concavity results on Gaussian process methods for supervised and unsupervised learning. *Advances in NIPS*, 16.

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

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

Papoulis, A. and Pillai, S. U. (2002). *Probability, Random Variables, and Stochastic Processes*. McGraw Hill, Boston.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems*. Morgan Kaufman, San Mateo, CA.

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

Pillow, J. W., Paninski, L., Uzzell, V. J., Simoncelli, E. P., and Chichilinsky, E. J. (2005). Prediction and decoding of retinal ganglion cell responses with a probabilistic spiking model. *J Neurosci*, 25(47).

Pillow, J. W., Shlens, J., Paninski, L., Sher, A., Litke, A., Chichilnisky, E. J., and Simoncelli, E. P. (2008). Spatio-temporal correlations and visual signalling in a complete neural population. *Nature*.

Polikov, V. S., Tresco, P. A., and Reichert, W. M. (2005). Response of brain tissue to chronically implanted neural electrodes. *J Neurosci Methods*, 148:1–18.

Proakis, J. G. and Salehi, M. (1994). *Communication Systems Engineering*. Prentice Hall, New Jersey.

Quinonero-Candela, J. and Rasmussen, C. (2005). A Unifying View of sparse approximate Gaussian process regression. *J. Machine Learning*, 6:1939–1959.

Rasmussen, C. and Williams, C. (2006). *Gaussian Processes for Machine Learning*. MIT Press, Cambridge.

Raykar, V., Yang, C., Duraiswami, R., and Gumerov, N. (2005). Fast computation of sums of Gaussians in high dimensions. *University of Maryland Tech. Report CS-TR-4767/UMIACS-TR-2005-69*.

Richmond, B., Optican, L., and Spitzer, H. (1990). Temporal econding of two-dimensional patterns by single units in primate primary visual cortex. i. stimulus-response relations. *J Neurophysiol*, 64(2).

Riehle, A. and Requin, J. (1989). Monkey primary motor and premotor cortex: single-cell activity related to prior information about direction and extent of an intended movement. *J Neurophysiol*, 61:534–549.

Roitman, J. D. and Shadlen, M. N. (2002). Response of neurons in the lateral intraparietal area during a combined visual discrimination reaction time task. *J Neurosci*, 22(21):9475–9489.

Rosenbaum, D. (1980). Human movement initiation: specification of arm, direction, and extent. *J Exp Psychol Gen*, 109:444–474.

Roweis, S. and Ghahramani, Z. (1999). A unifying review of linear Gaussian models. *Neural Comput*, 11(2):305–345.

Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326.

Sahani, M. (1999). *Latent Variable Models for Neural Data Analysis*. PhD thesis, California Institute of Technology.

Sanchez, J. C., Principe, J. C., Nishida, T., Bashirullah, R., Harris, J. G., and Fortes, J. (2008). Technology and signal processing for brain-machine interfaces: The need for beyond the state-of-the-art tools. *IEEE Sig Proc Magazine*, 25(1):29–40.

Santhanam, G., Linderman, M. D., Gilja, V., Afshar, A., Ryu, S. I., Meng, T. H., and Shenoy, K. V. (2007). HermesB: A continuous neural recording system for freely behaving primates. *IEEE Trans Biomed Eng*, 54(11):2037–2050.

Santhanam, G., Ryu, S. I., Yu, B. M., Afshar, A., and Shenoy, K. V. (2006). A high-performance brain-computer interface. *Nature*, 442:195–198.

Santhanam, G., Sahani, M., Ryu, S. I., and Shenoy, K. V. (2004). An extensible infrastructure for fully automated spike sorting during online experiments. In *Proc. of the IEEE EMBS*, pages 4380–4384.

Sasaki, T., Matsuki, N., and Ikegaya, Y. (2007). Metastability of active CA3 networks. *J Neurosci*, 27(3):517–528.

Schwartz, A. B. (2004). Cortical neural prosthetics. *Ann. Rev. Neurosci*, 27:487–507.

Seeger, M. (2002). Notes on Minka's Expectation Propagation for Gaussian Process classification. *Technical Report, University of Edinburgh*.

Seeger, M. (2003). Bayesian Gaussian process models: PAC-Bayesian generalisation error bounds and sparse approximations. *Ph.D. Thesis, U. Edinburgh*.

Seidemann, E., Meilijson, I., Abeles, M., Bergman, H., and Vaadia, E. (1996). Simultaneously recorded single units in the frontal cortex go through sequences of discrete and stable states in monkeys performing a delayed localization task. *J Neurosci*, 16(2):752–768.

Serruya, M., Hatsopoulos, N., Paninski, L., Fellows, M., and Donoghue, J. (2002). Instant neural control of a movement signal. *Nature*, 416:141–142.

Shadmehr, R. and Wise, S. P. (2005). *The Computational Neurobiology of Pointing and Reaching*. MIT Press, Cambridge.

Shakhnarovich, G., Kim, S. P., and Black, M. J. (2006). Nonlinear physically-based models for decoding motor-cortical population activity. In *Advances in NIPS 18*. MIT Press, Cambridge, MA.

Shen, Y., Ng, A., and Seeger, M. (2006). Fast Gaussian Process Regression using KD-trees. *Advances in NIPS*, 18.

Shenoy, K., Meeker, D., Cao, S., Kureshi, S., Pesaran, B., Mitra, P., Buneo, C., Batista, A., Burdick, J., and Andersen, R. (2003). Neural prosthetic control signals from plan activity. *NeuroReport*, 14:591–596.

Shimazaki, H. and Shinomoto, S. (2007a). Kernel width optimization in the spike-rate estimation. *Neural Coding 2007, Montevideo, Uruguay*.

Shimazaki, H. and Shinomoto, S. (2007b). A method for selecting the bin size of a time histogram. *Neural Comput*, 19(6):1503–1527.

Shlens, J., Field, G., Gauthier, J., Grivich, M., Petrusca, D., Sher, A., Litke, A., and Chichilnisky, E. J. (2006). The structure of multi-neuron firing rate patters in primate retina. *J Neurosci*, 26.

Shoham, S., Paninski, L., Fellows, M., Hatsopoulos, N., Donoghue, J., and Normann, R. (2005). Statistical encoding model for a primary motor cortical brain-machine interface. *IEEE Transactions on TBME*, 52(7):1313–1322.

Silverman, B. (1982). Kernel density estimation using the fast fourier transform. *Journal of Royal Stat. Soc. Series C: Applied Stat.*, 33.

Smith, A. C. and Brown, E. N. (2003). Estimating a state-space model from point process observations. *Neural Comput*, 15(5):965–991.

Sollich, P. and Williams, C. K. I. (2005). Using the equivalent kernel to understand Gaussian process regression. In Saul, L. K., Weiss, Y., and Bottou, L., editors, *Adv Neural Info Processing Sys 17*, pages 1313–1320. MIT Press, Cambridge, MA.

Srinivasan, L. and Brown, E. N. (2007). A state-space framework for movement control to dynamic goals through brain-driven interfaces. *IEEE Transactions on Biomedical Engineering*, 54(3):526–535.

Srinivasan, L., Eden, U. T., Mitter, S. J., and Brown, E. N. (2007). General purpose filter design for neural prosthetic systems. *J Neurophys*, 98:2456–2475.

Srinivasan, L., Eden, U. T., Willsky, A. S., and Brown, E. N. (2006). A state-space analysis for reconstruction of goal-directed movements using neural signals. *Neural Computation*, 18:2465–2494.

Stark, E., Drori, R., and Abeles, M. (2006). Partial cross-correlation analysis resolves ambiguity in the encoding of multiple movement features. *J Neurophysiol*, 95:1966–1975.

Stopfer, M., Jayaraman, V., and Laurent, G. (2003). Intensity versus identity coding in an olfactory system. *Neuron*, 39:991–1004.

Strang, G. (1988). *Linear Algebra and Its Applications*. Saunders.

Suner, S., Fellows, M. R., Vargas-Irwin, C., Nakata, G. K., and Donoghue, J. P. (2005). Reliability of signals from a chronically implanted silicon-based electrode array in non-human primate primary motor cortex. *IEEE Trans. Neural Sys. Rehabil. Eng.*, pages 524–541.

Tanji, J. and Evarts, E. (1976). Anticipatory activity of motor cortex neurons in relation to direction of an intended movement. *J Neurophysiol*, 39:1062–1068.

Taylor, D., Tillery, S. H., and Schwartz, A. (2002). Direct cortical control of 3D neuroprosthetic devices. *Science*, 296:1829–1832.

Teh, Y. W. and Roweis, S. (2003). Automatic alignment of local representations. In Becker, S., Thrun, S., and Obermayer, K., editors, *Adv Neural Info Processing Sys 15*, pages 841–848. MIT Press, Cambridge, MA.

Teh, Y. W., Seeger, M., and Jordan, M. I. (2005). Semiparametric latent factor models. In Cowell, R. G. and Ghahramani, Z., editors, *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AISTATS)*. Society for Artificial Intelligence and Statistics.

Thiebaut, R. and Jacqmin-Gadda, H. (2004). Mixed models for longitudinal left-censored repeated measures. *Comput Methods Programs Biomed*, 74(3):255–260.

Tipping, M. E. and Bishop, C. M. (1999). Probabilistic principal component analysis. *J R Stat Soc Ser B*, 61(3):611–622.

Todorov, E. (2000). Direct cortical control of muscle activation in voluntary arm movements: a model. *Nature Neuroscience*, 3:391–398.

Todorov, E. and Jordan, M. (2002). Optimal feedback control as a theory of motor coordination. *Nature Neuroscience*, 5(11):1226–1235.

Tolhurst, D. J., Movshon, J. A., and Dean, A. F. (1983). The statistical reliability of signals in single neurons in cat and monkey visual cortex. *Vision Res*, 23(8):775–785.

Truccolo, W., Eden, U. T., Fellows, M. R., Donoghue, J. P., and Brown, E. N. (2005). A point process framework for relating neural spiking activity to spiking history, neural ensemble and extrinsic covariate effects. *J Neurophysiol*, 93:1074–1089.

Turner, R. E. and Sahani, M. (2007). A maximum-likelihood interpretation for slow feature analysis. *Neural Comput*, 19(4):1022–1038.

Velliste, M., Perel, S., Spalding, M. C., Whitford, A. S., and Schwartz, A. B. (2008). Cortical control of a prosthetic arm for self-feeding. *Nature*, 453.

Ventura, V. (2008). Spike train decoding without spike sorting. *Neural Comp*, 20:923–963.

Ventura, V., Cai, C., and Kass, R. E. (2005). Trial-to-trial variability and its effect on time-varying dependency between two neurons. *J Neurophysiol*, 94:2928–2939.

Ventura, V., Carta, R., Kass, R., Gettner, S., and Olson, C. (2002). Statistical analysis of temporal evolution in single-neuron firing rates. *Biostatistics*, 3(1):1–20.

Verdu, S. (1986). Asymptotic error probability of binary hypothesis testing for Poisson point-process observations. *IEEE Transactions on Information Theory*, IT-32(1):113–115.

Wahnoun, R., He, J., and Tillery, S. I. H. (2006). Selection and parameterization of cortical neurons for neuroprosthetic control. *J Neural Eng.*, 3:162–171.

Wang, J., Fleet, D., and Hertzmann, A. (2006). Gaussian process dynamical models. In Weiss, Y., Schölkopf, B., and Platt, J., editors, *Adv Neural Info Processing Sys 18*, pages 1441–1448. MIT Press, Cambridge, MA.

Weber, A. P. and Hahnloser, R. H. R. (2007). Spike correlations in a songbird agree with a simple Markov population model. *PLoS Comput Biol*, 3(12):2520–2531.

Weinrich, M., Wise, S., and Mauritz, K. (1984). A neurophysiological study of the premotor cortex in the rhesus monkey. *Brain*, 107:385–414.

Weiss, Y. and Freeman, W. T. (2001). Correctness of belief propagation in Gaussian graphical models of arbitrary topology. *Neural Comp.*, 13:2173–2200.

Wessberg, J., Stambaugh, C., Kralik, J., Beck, P., Laubach, M., Chapin, J., Kim, J., Biggs, S., Srinivasan, M., and Nicolelis, M. (2000). Real-time prediction of hand trajectory by ensembles of cortical neurons in primates. *Nature*, 408:361–365.

Wise, K., Anderson, D., Hetke, J., Kipke, D., and Najafi, K. (2004). Wireless implantable microsystems: High-density electronic interfaces to the nervous system. *Proceedings of the IEEE*, 92:76–97.

Wolpaw, J. R. and McFarland, D. J. (2004). Control of a two-dimensional movement signal by a noninvasive brain-computer interface in humans. *Proc Natl Acad Sci USA*, 101(51):17849–17854.

Wood, F. and Black, M. J. (2008). A nonparametric bayesian alternative to spike sorting. *J Neurosci Methods*, 173:1–12.

Wood, F., Black, M. J., Vargas-Irwin, C., Fellows, M., and Donoghue, J. P. (2004). On the variability of manual spike sorting. *IEEE Trans. Biomed. Eng.*, 51:912–918.

Wu, W., Black, M., Gao, Y., Bienenstock, E., Serruya, M., and Donoghue, J. (2002). Inferring hand motion from multi-cell recordings in motor cortex using a Kalman filter. In *SAB'02-Workshop on Motor Control in Humans and Robots: On the Interplay of Real Brains and Artificial Devices*, pages 66–73.

Wu, W., Black, M., Mumford, D., Gao, Y., Bienenstock, E., and Donoghue, J. (2004). Modeling and decoding motor cortical activity using a switching Kalman filter. *IEEE Transactions on TBME*, 51(6):933–942.

Wu, W., Gao, Y., Bienenstock, E., Donoghue, J., and Black, M. (2006). Bayesian population decoding of motor cortical activity using a Kalman filter. *Neural Comp*, 18(1):80–118.

Wu, W. and Hatsopoulos, N. G. (2008). Target-included model and hybrid decoding of stereotyped hand movement in the motor cortex. In *Intl Conf on Biomed Robotics and Biomech, 2008. (IEEE RAS/EMBS)*, pages 55–60.

Wu, W., Kulkarni, J. E., Hatsopoulos, N. G., and Paninski, L. (2008). Neural decoding of goal-directed movements using a linear state-space model with hidden states. *COSYNE Abstract*.

Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2002). Understanding belief propagation and its generalizations. *Mitsubishi ERL Tech. Report TR-2001-22*.

Yu, B. M., Afshar, A., Santhanam, G., Ryu, S. I., Shenoy, K. V., and Sahani, M. (2006). Extracting dynamical structure embedded in neural activity. In Weiss, Y., Schölkopf, B., and Platt, J., editors, *Adv Neural Info Processing Sys 18*, pages 1545–1552. MIT Press, Cambridge, MA.

Yu, B. M., Cunningham, J. P., Santhanam, G., Ryu, S. I., Shenoy, K. V., and Sahani, M. (2008). Gaussian process factor analysis for low-dimensional single-trial analysis of neural population activity. *Soc Neurosci Abstr*, (319.9).

Yu, B. M., Cunningham, J. P., Santhanam, G., Ryu, S. I., Shenoy, K. V., and Sahani, M. (2009a). Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity. *J Neurophysiol*, 102(1):614–635.

Yu, B. M., Cunningham, J. P., Santhanam, G., Ryu, S. I., Shenoy, K. V., and Sahani, M. (2009b). Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Adv Neural Info Processing Sys 21*, pages 1881–1888. MIT Press, Cambridge, MA.

Yu, B. M., Kemere, C., Santhanam, G., Afshar, A., Ryu, S. I., Meng, T. H., Sahani, M., and Shenoy, K. V. (2007). Mixture of trajectory models for neural decoding of goal-directed movements. *J Neurophysiol*, 97:3763–3780.

Zar, J. (1999). *Biostatistical Analysis*. Prentice Hall, New Jersey.

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

Zhao, Y., Grambsch, P. M., and Neaton, J. D. (2005). Comparison of numerical algorithms for bivariate sequential tests based on marginal criteria. *Comp. Stat. and Data Anal.*, 49:631–641.

Zhu, H. and Rohwer, R. (1995). Information geometric measurements of generalisation. *Aston Univ. Tech. Report NCRG/4350*.

Zumsteg, Z. S., Kemere, C., O'Driscoll, S., Santhanam, G., Ahmed, R. E., Shenoy, K. V., and Meng, T. H. (2005). Power feasibility of implantable digital spike sorting circuits for neural prosthetic systems. *IEEE TNSRE*, 13:272–279.