Methods for Building Network Models of Neural Circuits

Brian DePasquale

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
under the Executive Committee
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2016

ABSTRACT

Methods for Building Network Models of Neural Circuits

Brian DePasquale

Artificial recurrent neural networks (RNNs) are powerful models for understanding and modeling dynamic computation in neural circuits. As such, RNNs that have been constructed to perform tasks analogous to typical behaviors studied in systems neuroscience are useful tools for understanding the biophysical mechanisms that mediate those behaviors. There has been significant progress in recent years developing gradient-based learning methods to construct RNNs. However, the majority of this progress has been restricted to network models that transmit information through continuous state variables since these methods require the input-output function of individual neuronal units to be differentiable. Overwhelmingly, biological neurons transmit information by discrete action potentials. Spiking model neurons are not differentiable and thus gradient-based methods for training neural networks cannot be applied to them.

This work focuses on the development of supervised learning methods for RNNs that do not require the computation of derivatives. Because the methods we develop do not rely on the differentiability of the neural units, we can use them to construct realistic RNNs of spiking model neurons that perform a variety of benchmark tasks, and also to build networks trained directly from experimental data. Surprisingly, spiking networks trained with these non-gradient methods do not require significantly more neural units to perform tasks than their continuous-variable

model counterparts. The crux of the method draws a direct correspondence between the dynamical variables of more abstract continuous-variable RNNs and spiking network models. The relationship between these two commonly used model classes has historically been unclear and, by resolving many of these issues, we offer a perspective on the appropriate use and interpretation of continuous-variable models as they relate to understanding network computation in biological neural circuits.

Although the main advantage of these methods is their ability to construct realistic spiking network models, they can equally well be applied to continuous-variable network models. An example is the construction of continuous-variable RNNs that perform tasks for which they provide performance and computational cost competitive with those of traditional methods that compute derivatives and outperform previous non-gradient-based network training approaches.

Collectively, this thesis presents efficient methods for constructing realistic neural network models that can be used to understand computation in biological neural networks and provides a unified perspective on how the dynamic quantities in these models relate to each other and to quantities that can be observed and extracted from experimental recordings of neurons.

Table of Contents

List of Figures

Acknowledgments

One of the most satisfying components of being a scientist is that work is inherently collaborative. This thesis would not have been possible without the support of many people over many years, whom I would like to thank and acknowledge.

First and foremost, I would like to thank my wife and partner, Akemi Martin, for her patience, support and inspiration. Akemi is an incredibly courageous woman whose fearlessness inspires me to live life the way I want, no matter how confusing my choices might seem to others. She has an uncanny ability to see things as they are, not as they seem, which has profoundly influenced my work and life, both scientifically and outside of science. I would also like to thank our dog, Caesar, who reminds me that life is a joyful thing and that everyday has something new to offer.

I would like to thank my parents, Michael and Kathleen DePasquale, for their constant love and support. I can only imagine what it must be like to have me as a son. They have always stood behind me and tried to understand whatever crazy idea I felt compelled to pursue. Growing up, I never considered them renegades, or boundary-pushers, two qualities I now view as synonymous with being a scientist. Age has brought me perspective on this, and it is clear to me now that the relentless desire I feel for exploring the world around me was quietly imbued in me by them. I would also like to thank my siblings, Michael and Anthony DePasquale, for politely nodding along when I tried to explain my research to them.

I would like to thank prior research collaborators from MIT, in particular Joey Feingold, Chris Moore and Mike Long, for aiding my transition into neuroscience and for guidance while applying to graduate school. I shared many joyful moments with them recording from monkeys

and playing softball, and miss their cheerful disposition in my everyday life. Additionally, I would like to thank Vassilios Fessatidis, an undergraduate professor at Fordham University, who supported me in my choice to leave physics and move into neuroscience. Much of my early mathematical knowledge is due to his exceptional teaching.

I would like to thank early research supervisors and thesis committee members—Randy Bruno, Liam Paninski, Stefano Fusi, Misha Tsodyks, Jonathan Pillow, John Cunningham, and (briefly) Mike Shadlen—for formative and guiding research projects and discussions during my PhD. Each one contributed, at least in some small way, to the ideas produced in this work, and for that I am grateful. In particular, I would like to thank Stefano, for being around for advice when (sometimes) Larry was not.

I would like to thank members of the Theory Center, the Neurobiology program and the Churchland Lab, in particular Matthew Lovett-Barron, Charlotte Barkan, Christine Constantinople, Saul Kato, Sean Escola, Armen Enikolopov, Xaq Pitkow, Merav Stern, Yashar Ahmadian and Hagai Lalazar, for years of fun, guidance and riveting scientific interaction. I would like to thank Omri Barak and Greg Wayne specifically for their tutelage in the early stages of my PhD, when I could barely write a single line of code.

I would like to thank good friends from Cambridge, MA and from my Fordham days, in particular David Craft, Alex Ince-Cushman and Nathan Paluck, for pushing me to pursue a PhD and for keeping me distracted from doing it.

I would like to express my deep gratitude to my research advisors, Larry Abbott and Mark Churchland. Many of the more philosophical points regarding neural coding contained in this thesis are directly inspired by the work of and conversations with Mark. My discussions with him throughout my PhD have always been lively, challenging and ultimately have made me a

more thoughtful scientist. For this, I am greatly indebted to him.

I know that Larry does not possess a deep love for jazz, but I was struck by a Miles Davis quote while completing my thesis that I think rings true to my student-mentor relationship with him: "Sometimes you have to play a long time to be able to play like yourself." To me, the two main challenges a student must face when completing a PhD is to learn to play, and then to learn to play like yourself. Larry gave me the attention I needed to learn to play, and the space, time, inspiration and support I needed to learn to play like myself. He is a truly exceptional scientist, and more generally, a magnanimous and genial human being. It has been a great pleasure to spend time working and thinking with him.

Additionally, Larry taught me the deep importance of clarity when sharing scientific ideas. At this point, to me, the primary function of theoretical neuroscience within the greater neuroscience community is to provide clarity, not confusion. I hope this thesis does him justice.

for Akemi and Caesar

Chapter 1

Introduction

Levels of abstraction in modeling and analysis

Every behavior that an organism performs is generated by the coordinated activity of a neural circuit— an interconnected network of neurons. The primary focus of experimental systems neuroscience is to understand what aspects of neural circuit activity give rise to a specific behavior. To aide this effort, theoretical neuroscientists construct artificial neural circuit models that can perform analogous tasks while replicating the structural properties and activity patterns observed in biological neural circuits. Such models can themselves be studied with the hope that they may shed light on the processes at play in biological neural circuits.

In order for these models to be useful tools for understanding biological neural circuits, it is imperative that they capture as many salient features of biological neural circuits as possible. Furthermore—to avoid constructing a model of a system that is as complicated as the system itself—it is critical that the neuroscience community identify which aspects of biological circuits are truly critical to their function and therefore need to be modeled explicitly. At this juncture in our understanding of neural circuits, perhaps the most important question to be addressed when considering neural coding and models of neural circuits is what level of abstraction is appropriate to capture the operation of a neural circuit, and thus, understand it. Given a particular level

of abstraction, it is imperative that models of neural data and theoretical neural network models firmly connect these abstractions to measurable biophysical quantities, so that their interpretation yields insight into neural circuit operation.

This thesis seeks to offer a unified perspective on the relationship between general principles of neural coding, variables extracted from biological neural circuits and network models that seek to replicate these principles. In order to do this, we must achieve two goals: 1) we must overcome the technical difficulties faced when constructing artificial recurrent neural networks that can perform tasks akin to behaviors studied within the experimental neuroscience community, and 2) we must develop an interpretation of what the signals in different classes of network models represent, how they relate to each other and how they relate to neural activity observed in biological neural circuits.

Overview of dissertation

In this Introduction, I lay the groundwork for integrating these ideas. First, I introduce the hypotheses and assumptions of standard models of neural coding. I provide a general overview of these concepts and experimental evidence relating these concepts to observable features in neural data. Then, I introduce existing network models that capture some of these features, and I outline modifications that need to be made to these models to bring them in line with a more modern perspective on neural coding. I formally state the problem of constructing artificial recurrent networks—both spiking and non-spiking networks—and introduce methods to overcome these difficulties. I introduce and distinguish the two classes of network model that we study, how they relate to each other, features of biological neural circuits they can replicate and how they can be used as models of computation in biological neural circuits.

Chapter Two presents an overview of state-of-the-art methods for training recurrent spiking neural networks. We review previous methods and relate these to the main work of this thesis, to be presented in the following chapter.

Chapter Three describes the main body of research that comprises this thesis. I present the full technical details of our training method for building recurrent spiking networks that was reviewed in Chapter Two. I present several examples of functioning spiking networks performing tasks of relevance to systems neuroscience that adhere to many of the most salient constraints of biological neural circuits. Furthermore, I illustrate how this procedure can be used to model neuroscience data by constructing a spiking network to match electromyograms (EMG) recorded during a reaching task. I introduce methods for including additional biological realism in these models, including constructing sparse and Dale's Law obeying networks. I conclude by discussing the relationship between the dynamical variables in spiking and continuous-variable models with the hope of elucidating the connection between the two classes of models as well as their relationship to dynamical variables extracted from neural data.

Chapter Four re-visits the question of training recurrent continuous-variable networks, aided by lessons learned during the process of constructing recurrent spiking networks. I show that, just as in spiking networks, deriving targets for the full recurrent connectivity matrix results in superior learning compared to previously presented methods. Additionally, I illustrate how this improved learning comes about from analysis of the learned connectivity matrix and how these modifications lead to increased noise robustness.

Chapter Five offers some concluding remarks, implications of the presented work and future directions. I offer additional perspective of the role of continuous-variable and spiking networks in modeling biological neural circuits with a focus on what can be learned from these networks

regarding neural function and computation.

Neural coding

While the majority of this thesis is concerned with building functional network models, it is important to review some history of neural coding, since one of the primary functions of a network model is to instantiate a hypothesis about how neural circuits operate. The history of neural coding is rich and its relationship to concepts in theoretical neuroscience can be subtle. I therefore devote significant attention to making sure these concepts are clear, so that how they relate to the modeling choices I make later will be clear.

Firing rate codes

Perhaps the most salient feature of neural activity in higher organisms is that neurons predominantly represent information and communicate that information to other neurons by action potentials: a nearly discontinuous spike in voltage across the membrane of a neuron. This simple fact has made the process by which neural circuits encode information profoundly elusive. Adding to our confusion, it has been widely observed across many brain regions that neural spiking activity is surprisingly variable even when an animal performs a nominally identical behavior or experiences identical stimuli [Churchland et al., 2010b; Goris et al., 2014; Shadlen and Newsome, 1994, 1998; Softky and Koch, 1993]. This poses a vexing question: how can a system of unreliable elements give rise to reliable output? To overcome the challenges that spiking neurons present, both theoretical and experimental neuroscientists have done the most sensible thing possible: simply get rid of them and replace them with more agreeable functions and concepts.

Of course, this choice is not made flippantly, but based on far-reaching and deep assumptions about the nature of neural coding.

In many neural systems, numerous studies have found that the frequency of spike discharges averaged over an appropriately defined temporal window or group of neurons (which will be defined more precisely momentarily) carries the most useful information for understanding what relationship individual neurons or groups of neurons might play in generating behavior [Britten et al., 1992; Churchland et al., 2012; Georgopoulos et al., 1993; Romo and Salinas, 2001]. This averaged quantity—typically referred to as a "firing rate" when considering a single neuron or a group of neurons that respond in a similar way—translates cumbersome discontinuous spikes trains into more tractable continuous quantities or temporal functions while mostly retaining the original data's explanatory power. The principle justification for this assumption is that the variable nature of spiking (either across a given time interval, across repetitions of a task or across a group of similarly responding neurons) represents a source of noise that should rightly be suppressed through averaging to undercover the latent signal—the firing rate—present within the neuron's or neurons' activity (Figure 1.1).

Once the relevant signal of interest is uncovered by an averaging process, these signals can be *decoded*, which is to say related in some way to a stimulus or performed behavior to elucidate the response's relationship to it. Parametrically defining the relationship between a neuron's firing rate (either observed across a sufficient time window, across repetitions of a behavior or across a group of similarly responsive neurons) as a function of the value of a presented stimulus or performed behavior gives rise to the concept of *neural tuning* [Hubel and Wiesel, 1959]: that a neuron's or group of neurons' response will exhibit some degree of reproducibility to a presented stimulus (its firing rate), and that its firing rate varies systematically for different stimuli. Understanding the tuning properties of neurons allows their activity to be used to decode stimuli or

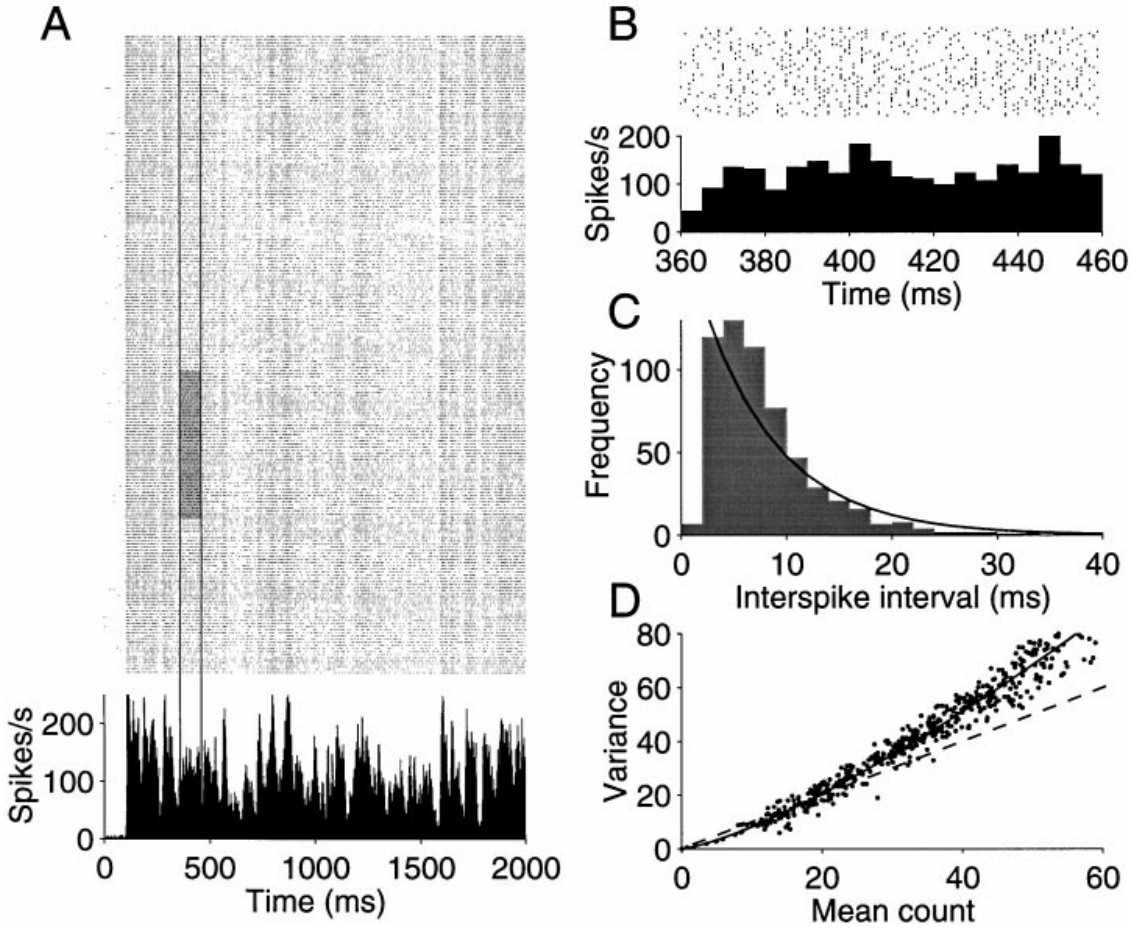

Figure 1.1: The variable nature of spiking and recovery of the firing rate.

(**a**) (**top**) Response of an area MT neuron across 210 presentations of an identical visual stimulus. (**bottom**) The peri-stimulus time histogram (PSTH) of this neuron, computed by averaging the spiking response across trials, binning the spikes in 2 ms intervals. The vertical lines delineate a time window over which the firing rate was relatively constant, and the gray box identifies a subset of 50 trials plotted in (**b**). (**b**) Response across 50 trials, illustrating the trial-to-trial variability in spiking. (**c**) The distribution of time between successive spikes for this neuron (inter-spike interval, ISI) is closely approximated by an exponential distribution, characteristic of a Poisson point process. (**d**) The spike count mean plotted against the spike count variance from 50 to 200 adjacent trials in an epoch from 100 to 500 ms long. The dotted-line indicates what data generated from a Poisson point process would look like. (adapted from Shadlen and Newsome [1998])

intended behaviors.

The primary consideration with regard to a firing-rate code is precisely what to average over.

For many simple behaviors—for example classifying a stimulus as being greater in magnitude or lesser in magnitude than a prior, remembered stimulus [Romo and Salinas, 2001]—simply counting the number of spikes an individual neuron generates while the stimulus is presented (and thus averaging over time) may be sufficient to correctly decode the decision the animal ultimately made.

When considering more complex behaviors, it's worth asking *who* is doing the averaging and across what collection of neural activity. Two perspectives are worth considering here: the view of an experimental neuroscientist and the view of a neuron in the brain. In many ways, neurons in the brain are faced with the identical problem that we are, namely trying to estimate a signal from noisy spiking activity. When trying to understand how a neuron's response relates to a behavior, an experimental neuroscientist will have access to a perspective that a neuron in the brain will not. A typical experiment will observe the activity of a neuron over multiple repetitions of a behavior. The inquisitive neuroscientist can average across these repetitions (as was done in Figure 1.1) thereby suppressing the variable component of the response and uncovering the neuron's reproducible firing rate. Clearly, neurons in the brain do not have this luxury.

Assuming that a neuron in the brain observes as much noise (in the form of spiking irregularity) as we do, then it too must identify a collection of spikes (as we did, using the collection of spikes over repeated trials of one neuron) over which to average so that it may extract the meaningful signal it requires to participate in computation. For example, if a neuron was presented with only a single-trial's worth of activity from an experimentalist's recording, this signal would be far too noisy for the neuron to respond appropriately. This realization led to the hypothesis that *groups* of similarly tuned neurons form functional units and that these groups of neurons can collectively estimate (from each others' activity) and propagate the signal that they are tasked with encoding [Shadlen and Newsome, 1994]. This hypothesis was bolstered by the observation

that anatomically clustered neurons tended to have similar tuning properties and that identical tuning properties seemed to be represented by groups of neurons [Georgopoulos et al., 1993; Shadlen and Newsome, 1994].

This yields a picture as to how a rate code can be instantiated in a functional group of spiking neurons and how the continuous firing-rate variables of interest to us can be extracted from recordings of one member of this group or many members of the group. If we consider a group of similarly tuned neurons encoding a stimulus and the perspective of one neuron in that group, then, provided that this group is connected in such a way that the input into our chosen neuron is an average of the output of other neurons in the group, its output will reflect the encoded variable (albeit in its own, noisy manner). In this way, a correspondence between the perspective that a neuroscientist has and the perspective that a neuron has can be drawn: assuming that spiking noise is isotropic with respect to space (across neurons in the group) and time (across repetitions), then the collection generated from one neuron viewed across multiple repetitions of a behavior should constitute a surrogate data set to the collection of spikes one neuron will view on a single trial. In order for an experimenter—or a neuron—to estimate the firing rate of the group, they need only average over a collection of identical repeats of a behavior (if they have access to that data) or across the group itself.

Static population codes

As tasks become more complicated, the activity of a single neuron or a group of similarly tuned neurons may not be sufficient to decode a stimulus or behavioral response. Additionally, neural tuning itself (i.e. the reproducible way in which a neuron responds) becomes more complicated and thus more complex averaging is needed to decode relevant signals [Georgopoulos et al.,

1993]. This led to the development of *population codes* [Averbeck et al., 2006; Salinas and Abbott, 1994] based on a weighted average across an appropriately defined *ensemble* of neural activity (and possibly across time). From a population coding perspective, behavior can be decoded from an ensemble of spiking neurons, where each neuron exhibits a particular form of tuning and reports that information in a noisy way. With increased task complexity, it is only possible to decode behavior with the information represented in the neural ensemble collectively.

We take a moment here to distinguish our definitions of ensemble and group, as they are quite close to each other, but diverge in a very important way. Previously, we defined a group as a collection of identically tuned spiking neurons that collectively encode a firing-rate. Each neuron reports this firing-rate in a noisy way, and thus an average must be taken to accurately estimate it. Here, ensemble also refers to a collection of spiking neurons that collectively encode a signal or signals in a noisy way, except that we do not require these neurons to be similarly tuned. We draw this distinction here because the ambiguity with which these concepts have been treated in the past has led to additional ambiguity with regard to theoretical models based on these concepts (as we will discuss later).

One of the most successful applications of population codes to understanding behavior is the population vector approach for decoding, used to decode the direction of movement from recordings in the motor system [Georgopoulos et al., 1993]. In this approach, an ensemble of firing rates (where spiking activity is averaged across time to compute a firing rate) is collected from an animal performing a three-dimensional reaching task, and the goal is to decode the movement direction from the activity of the ensemble. We represent the population firing rate of the ensemble by the N-dimensional vector $\boldsymbol{r}$, and the direction of the movement as the 3-dimensional vector $\boldsymbol{V}$. Each neuron is assumed to have a preferred direction, which is to say that it is tuned to a certain direction and will respond most rigorously when the animal makes a movement in that

direction. We denote each neuron's preferred direction by the 3-dimensional vector C_i, where $C_i = V$ for the V that causes the i^{th} neuron to respond maximally. From this, an estimate of the value of a movement direction can be constructed as a rate-weighted sum of the preferred direction vectors: $V_{est} = \sum_{i=1}^{N} r_i C_i$. An example of this decoding is shown in Figure 1.2.

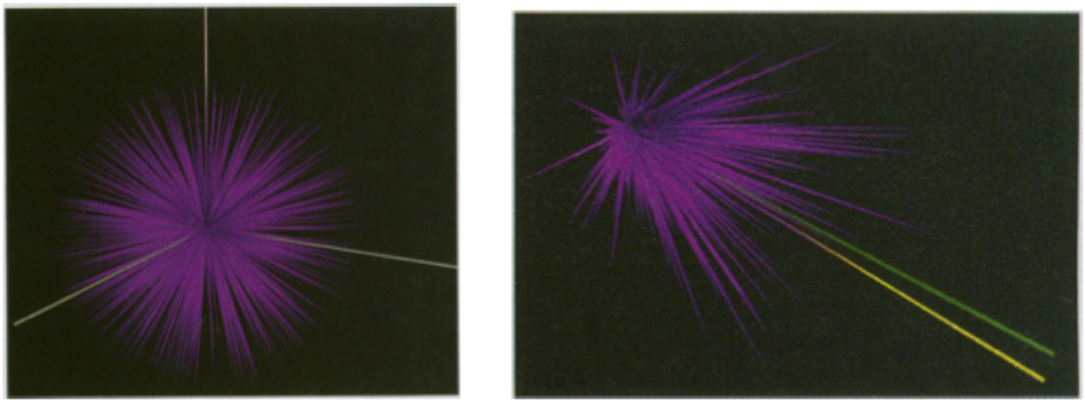

Figure 1.2: Population decoding in the motor system.
(left) Preferred direction vectors C_i from 634 motor cortical cells **(right)** Rate-weighted preferred direction vectors (purple) and the sum of these vectors V_{est} (green). The actual direction of movement V is shown in yellow. (adapted from Georgopoulos et al. [1993])

The population vector approach performs decoding across an ensemble of neurons, and thus doesn't explicitly confine them to functional groups (as we defined group earlier). Thus, this form of analysis can successfully decode a relatively complex behavior from an ensemble of neurons that exhibit a diverse set of tuning properties. While this perspective is certainly in line with our view on the best way to describe the coordinated activity of a population of neurons in the context of a network model and with more modern perspectives that will be discussed, it is worth noting that the more restrictive concept of a neural group can still be seen in this work. For example, the authors note the presence of similarly tuned neurons in their dataset, a concept that is more in line with the idea of functional groups than with ideas we introduce in future

sections.

Dynamic representation and heterogeneity of response

While population coding advanced our understanding of how complex neural responses relate to behavior, new experimental evidence requires this perspective to be updated further. With the ability to record from more neurons over longer time windows, and while animals are performing more complex tasks, the definitions of the continuous variables that should be defined to link to behavior (such as firing rates, and population firing rates) has become murkier; although, many of the concepts previously introduced can still be used to identify them. Specifically, while it remains true that the majority of spiking activity reflects a form of network noise that should be suppressed through averaging, the way this average is computed requires even further relaxing of restrictions, making it an even more abstract concept. Analysis methods in line with this view have been developed and used effectively to uncover low-dimensional, dynamical [Churchland et al., 2012] or static [Machens, 2010] continuous quantities with strong predictive power over behavior.

Two important findings have emerged from recent experimental studies, specifically in the motor cortex, that have highlighted a need for this refinement [Churchland and Shenoy, 2007; Churchland et al., 2010a]. First, as more and more neuron response profiles have been collected in behaving animals, the heterogeneity of tuned responses exhibited in these recordings has grown (Figure 1.3) in contradiction to the findings of Georgopoulos et al. [1993]. This heterogeneity has grown so large that very few recorded neurons appear to have similar response profiles. These findings demand an update to the perspective on what constitutes a neural group, as I have defined it, since, evidence is emerging that each neuron would constitute the only member of a

group. In other words, no two neurons fire in the same way. These findings have a profound impact on the interpretation of continuous-variable models used in theoretical neuroscience (as will be discussed) and important implications for theories of neural coding [Rigotti et al., 2013].

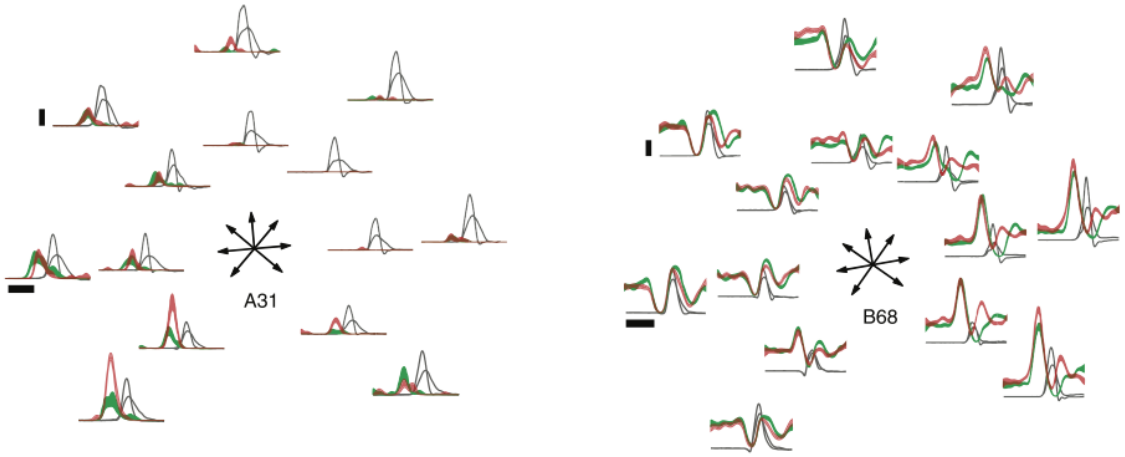

Figure 1.3: Neural responses in motor cortex are heterogeneous.
Example firing rates from two motor cortical neurons recorded during a reaching behavior. Each subplot shows the average firing rate for a neuron for one reach direction, two instructed reach speeds (red and green) and two reach distances (near and far). **(left)** A "classical" motor cortical neuron, displaying standard cosine tuning with respect to reach direction and scaling in magnitude with velocity. **(right)** A less classical neuron showing temporal complexity in its response and insensitivity to reach speed. (adapted from Churchland and Shenoy [2007])

Additionally, as our ability to collect data over extended time periods during complex tasks has increased, evidence has emerged that our traditional definition of static neural tuning is not adequate. For example, recordings from motor cortex during a reaching task have shown that, on average, a neuron's tuning to a reach direction during a preparatory time epoch preceding the reach is not significantly correlated with that neuron's tuning during the reach (Figure 1.4) [Churchland et al., 2010a]. This indicates that the concept of neural tuning needs to be updated to reflect these dynamic properties and that these modifications need to be fully appreciated in artificial network models.

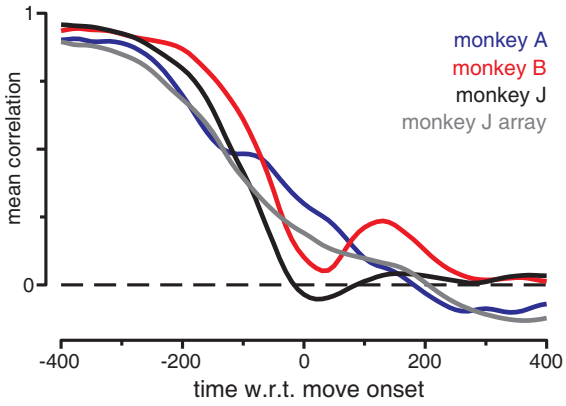

Figure 1.4: Neural tuning in motor cortex changes with time.

Average correlation (across the population of recorded neurons) between pre-movement firing rates and peri-movement firing rates across 4 datasets. At the time of movement, the average correlation of pre-movement firing rate with peri-movement firing rate is close to zero, indicating that the tuning has changed. (adapted from Churchland et al. [2010a])

What are the continuous quantities of interest then?

I take the position here—and go on to build models to support this position—that the relevant quantities of interest when considering the relationship between neural activity and behavior still share many of the features of the continuous variables—such as firing rate and population firing rate—that have already been defined. However, two important changes must be made. First, each individual continuous variable can no longer be considered an average across a homogeneous group of identically tuned neurons, since support for this view is weakening based on current evidence, and (as we will discuss) constructing artificial networks based on this principle leads us down an unrealistic and unhelpful path. Second, these variables must represent dynamic continuous quantities—such as the variables of a low-dimensional, continuous dynamical system—not conventionally defined firing-rates. These changes need to be made in light of the fact that these continuous variables continue to be instantiated by a higher-dimensional, noisy spiking ensemble, consistent with the basic tenets of rate-coding.

The goal of my network modeling efforts in this thesis is to update these concepts correspondingly and to fully investigate their implications in an artificial network model. By doing so, I hope to bolster support for the adoption of these analysis approaches and the assumptions they embody about neural coding by the experimental neuroscience community at large.

Artificial network models

Thus far, I have discussed hypotheses about how neural circuits encode information and experimental evidence supporting these hypotheses. I now turn to the larger focus of this thesis: constructing artificial network models that embody these assumptions, with the hope that these models can shed some light on various points of these hypotheses.

The narrative presented in the preceding section highlights the contentious question of the importance of spikes in neural coding. This question has dominated work on network models and coding theory in the field of theoretical neuroscience as well [Brette, 2015]. While there have been successes constructing networks of spiking model elements and non-spiking elements, the underlying subtext to these studies has been an uncertainty as to how important explicitly accounting for spiking activity is. This uncertainty has led to the development of two classes of neural network models: networks that include spiking neurons and networks that do not.

The justification for including spiking neurons in a model is rather obvious (since neurons spike); but the history of models that have not included spikes, and the justification for not doing so, is worth discussing here.

f-I curves and mean field theories of spiking neurons

At its basic core, a neuron is a nonlinear input-output device. The way that a neuron's input is mapped to its output can be characterized by its response function, commonly referred to as its frequency-current (f-I) curve. *In vitro* experiments have established that, to a first approximation, this function is saturating-threshold-linear, which is to say the neuron does not respond to inputs weaker than a certain value, will respond with a linearly increasing number of spikes for inputs stronger than this threshold, and eventually saturate at very high levels of input [La Camera et al., 2004; McCormick et al., 1985]. *In vitro* experiments are commonly performed under conditions that do not accurately reflect a neuron's input *in vivo*; neurons *in vivo* experience constant and substantial synaptic input that causes their output to become more irregular [Softky and Koch, 1993]. Under more realistic input conditions, experiments have shown that a neuron's f-I curve will become smoother around the threshold and can more accurately be described by a smooth, sigmoidal function [Rauch et al., 2003]. Individual spiking neuron models that receive external stochastic input accurately reflect these findings [Rauch et al., 2003]. Examples of these results are shown in Figure 1.5.

However, in real neural circuits, neurons do not receive truly stochastic input, but rather the collective output of other connected neurons. Recurrent models of spiking neurons—commonly referred to as "balanced networks"—have been developed that address these issues [Amit and Brunel, 1997; Brunel, 2000; van Vreeswijk and Sompolinsky, 1996]. Studied analytically using mean-field approaches [Renart et al., 2004] the activity of these networks can self-consistently replicate the variable nature of neural spiking. In simplified terms, mean-field approaches consider a recurrent network of spiking neurons and assess what firing states of the network are stable given the recurrent dynamics, under specific assumptions about the nature of the recurrent

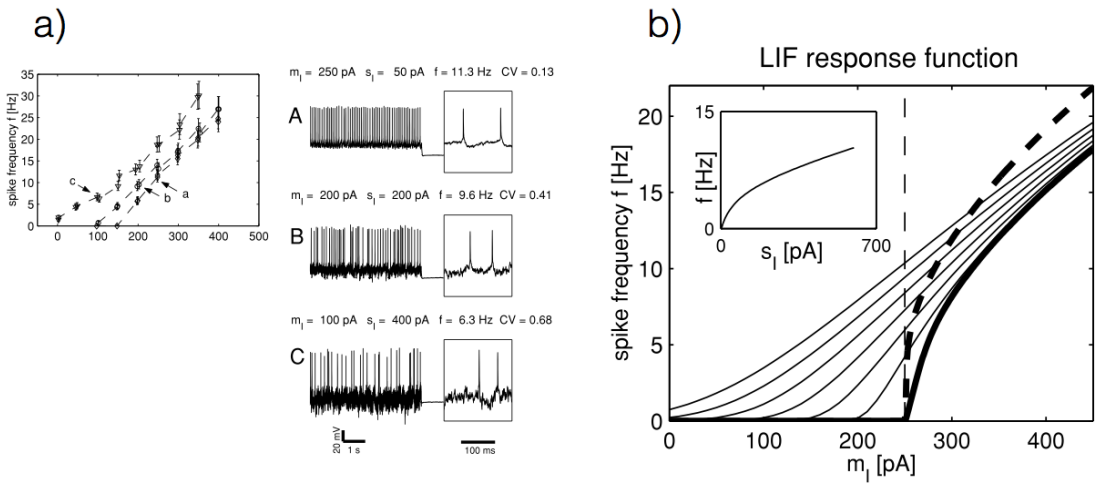

Figure 1.5: *In vivo*-like and LIF f-I curves.

(**a**) f-I curve for a rat cortical S1 neuron, recorded *in vitro*. Currents with varying levels of noise generated from an Ornstein-Uhlenbeck process were applied to the cell, thus approximating synaptic input *in vivo*. As the standard deviation of the noise was increased, the f-I curve became smoother near the threshold. (**b**) f-I curve for a simulated LIF neuron, driven with noise of various amplitudes. (adapted from Rauch et al. [2003])

connectivity and the statistics of firing that arise due to this connectivity. Since, in a recurrent circuit, the input to each neuron is defined by the output of other neurons in the network, provided that the mean and variance of the total recurrent input into each network neuron gives rise to output activity that is consistent with the statistics of its input, a stable network state will exist. Mathematically, this self-consistency equation can be summarized in the following way: $\nu = \phi(\mu(\nu), \sigma^2(\nu))$, where ϕ is the f-I curve of a neuron, μ is the firing-rate dependent mean input into each neuron and σ^2 is the variance of rate-dependent fluctuations of the input into each neuron. In sparsely connected networks, when the recurrent connections are strong (scale like $1/\sqrt{C}$, where C is the number of inputs a neuron receives) [van Vreeswijk and Sompolinsky, 1996] and when the mean excitatory and inhibitory input balance or for finite and more densely connected networks where the coupling strength between neurons is small [Amit and Brunel, 1997; Brunel, 2000], these networks exhibit stable, asynchronous dynamics that replicate the

irregular spiking observed *in vivo*.

The development of balanced networks was a major step forward in our ability to describe and model the irregular spiking observed in neural activity. We will return to the issue of generating irregular spiking and models of this general type in later sections.

Firing-rate models

A separate, contemporaneous line of research that considered the input-output function of an individual neuron more heuristically led to the development of a class of non-spiking neuron models commonly referred to as firing-rate models [Wilson and Cowan, 1972]. In this neuron model, the input-output function is heuristically taken to be a smooth sigmoidal function since this function captures the basic features of an individual neuron's or group of irregularly spiking neurons' f-I curve, as discussed above. These model neurons do not produce action potentials, but rather signal their output with a continuous function, commonly called "the firing rate", which reflects the approximate or averaged spiking activity a biological neuron would generate given a particular input. These models were developed with the perspective that the reduced set of continuous variables that were used represented an approximation to the firing rate of a noisy spiking neuron [Abbott and Kepler, 1990; Amit and Tsodyks, 1991a,b; Ermentrout, 1994; Gerstner, 1995; Ostojic and Brunel, 2011; Shriki et al., 2003] or the collective activity of a homogeneous group of spiking neurons [Knight, 1972; Wilson and Cowan, 1972]. This heuristic assumption was generally consistent with more principled, mean-field approaches where the noisy firing rate of an individual neuron or the collective firing rate of a group of irregularly spiking neurons was reduced to a smaller number of continuous, nonlinear equations whose transfer function takes the form of a smooth sigmoidal function.

Reducing the complex dynamics of spiking networks down to a small number of more mathematically manageable equations greatly simplified the task of building and studying neural network models and has been a boon to theoretical neuroscience. Network models of this form have been developed to account for many response properties in sensory systems [Dayan and Abbott, 2001], as memory systems based on attractor dynamics [Hopfield, 1982] and to generate more complex dynamic properties necessary for performing various neural functions [Vogels et al., 2005]. More recent work using these models has aided in understanding complex neural response properties related to decision-making and generating motor commands [Barak et al., 2013; Druckmann and Chklovskii, 2012; Hennequin et al., 2014; Mante et al., 2013; Sussillo et al., 2015]. These recent network models embraced a more abstract definition of the individual units in these networks, and, as I will now discuss, it was because of this re-interpretation that these models were able to fully tackle the challenge of accounting for the complex neural responses they sought to capture.

Reinterpreting firing-rate models

At this point, it is appropriate to readdress the distinction between groups of neurons and an ensemble of neurons. Within the context of rate-coding, we defined a group of neurons as a collection of noisy, similarly tuned neurons and an ensemble as a collection of noisy neurons with varied tuning. These concepts have re-emerged within our discussion of the development of firing-rate models and it is here that a main motivation for this work can be stated.

The early development and use of rate models has had a lasting impact on their interpretation which is out-of-step with their modern use. Originally conceived of as approximations to single spiking neurons or groups of similarly-tuned neurons, they became synonymous with the

concept of a neural group as defined from an experimental perspective. The scale of these early models was modest, owing to the comparatively modest neural phenomena they sought to describe. Furthermore, the interpretation of models developed at this scale remained consistent with the features of the neural data they sought to describe. As neural data sets have become more complex—owing to the study of more complex behavior—these models and the methods used to fit them to data have likewise become more complex. But the interpretation of the elements of the model—namely that each continuous,"firing-rate" variable corresponded to a group of similarly tuned neurons, or approximated a single neuron—has remained the same. The failure to re-interpret these models in light of this increasing complexity has exposed the shortcoming of this interpretation that we seek to highlight and address.

Modern approaches to modeling complex neural phenomena with firing-rate models can more accurately be considered an exercise in function approximation than an extension of neural network modeling as practiced in the early days of neural networks. Ultimately, when constructing a network model that seeks to explain a behavior, what one is trying to approximate is an abstract, complex, nonlinear function that describes the many aspects of that behavior. When a complex behavior is considered, this function is likely too complex to guess. Instead, we approximate its form using a generic function approximation device—a neural network—that can capture this functional form. We choose to approximate the functional form of behavior in this way because we have developed methods to do so. When firing-rate models are used in this way, the interpretation of each model unit—and thus the relationship of that unit to more primitive, biophysical quantities—will require a careful re-examination.

Exactly how the scale of the network model used to account for the behavior will grow as the complexity of the behavior being modeled grows is not clear. Modern approaches to modeling complex neural phenomena with firing-rate models typically require hundreds to thousands of

firing-rate neurons to be successful. While it is not believed that the underlying structure present in a complex data set requires hundreds to thousands of variables to be described, we simply do not have a method for best guessing what functional form the behavior takes, and therefore fit it instead with an over-parameterized form.

Our work illustrates that this shortcoming of interpreting firing-rate models as groups of spiking neurons becomes fully exposed when one attempts to construct a spiking model consistent with a modern firing-rate model. Attempting to construct a spiking network model equivalent to a modern day rate model based on the traditional interpretation of these models very quickly becomes intractable [Hennequin et al., 2014]. Constructing a spiking model and relating it to the activity of a firing-rate model requires an interpretation of the quantities of a more abstract model to a more primitive, biophysical model. If the interpretation is not sound, complications or inconsistencies should arise, and a revision of the interpretation is required.

The revision of rate-models that I support shares many of the features of their original interpretation but has profound implications on it as well. The variables in a firing-rate network can more accurately be considered abstract approximations to ensembles of spiking neurons, instead of approximations to individual spiking neurons or groups of similarly-tuned spiking neurons. Interpreted in this way, these models are more sensibly related to low-dimensional continuous signals that are decoded from ensembles of spiking neurons with modern analysis approaches [Churchland et al., 2012; Machens, 2010]. Each variable is not strictly related to a specific group of spiking neurons, but rather to an ensemble of spiking neurons that collectively generate a particular continuous signal. It was by re-interpreting firing-rate models in this way that I was able to construct reasonable-sized networks of spiking neurons that were consistent with firing-rate network models and more recent experimental results, as will be discussed in future chapters.

Building network models that perform tasks

With a thorough review of the features of neural activity we'd like to capture in a network model and the shortcomings of existing theoretical frameworks behind us, I now move on to the main focus of this dissertation.

Our modeling goals

In this work, I am interested in building functional networks of neurons (in the case of spiking networks) or neural "units" (in the case of continuous-variable networks) that perform *tasks* while embodying salient features of biological neural circuits and assumptions about how neurons encode information. Task, in this context, means a computation that a biological neural circuit might perform. Examples include producing complex dynamical outputs (as would need to be generated during movement), classifying inputs into proper classes (such as classifying visual objects into abstract categories) or remembering input over behaviorally relevant timescales (a common computation in working memory tasks).

Here we outline exactly what properties we'd like the elements of these networks to have.

1. **Network units should communicate with spikes.** As I outlined above, general methods for building spiking networks has been a lingering problem that we seek to address here. Without methods for building functional spiking models, more abstract models and analysis methods remain on shaky conceptual ground.

2. **Network units should exhibit realistic levels of spiking irregularity across time and**

variability across trials. As I will discuss, continuous-variable models can be constructed to perform impressive tasks, are useful tools for understanding computation in biological neural circuits, and are conceptually grounded based on experimental observations. However, they fail to capture and model the spiking noise which biological networks exhibit. Showing that realistic model networks of spiking networks can be constructed to perform similar tasks ensures their (continuous-variable models') relevance as models of neural circuits.

3. **Network units should exhibit fairly general, dynamic tuning.** As I have discussed, experimental evidence is emerging that old frameworks for considering neural tuning have become too constrictive. While network models built with continuous-variable models have addressed these issues, much remains to be addressed in spiking networks.

4. **Network units should exhibit extreme heterogeneity of response profiles.** Experimental evidence is growing thin for the existence of homogeneous groups of neurons, and spiking network models built on this assumption require an impractical and unrealistic number of neurons.

5. **Network units should be amenable to some form of learning, so that they can perform functions.** Interesting spiking models that perform tasks have been built in the past, but advanced forms of supervised learning have permitted the construction of even more impressive continuous-variable models while progress training spiking networks has lagged far behind. Without a practical and general procedure for constructing networks that can perform tasks, these models cannot help shed light on the problems of neural coding.

6. **Network units should generate ongoing, spontaneous activity that is spatiotemporally rich.** Giving rise to behavior is just one aspect of neural network dynamics; neurons

exhibit rich, ongoing activity in the absence of stimuli [Arieli et al., 1996; Tsodyks et al., 1999] and understanding these network dynamics will shed light on their operation when they *are* performing tasks.

7. **Network units should exhibit systematic fluctuations in their firing rate as a basis for task performance.** As I outlined above, our position is that coordinated fluctuations in neural firing are the dominant substrate of behavior. This property should be captured in any network model.

8. **Behavioral output should be able to be decoded in a simple way.** Ultimately, signals must be decoded by other neurons, or by neuroscientists. I presume this form of decoding is not overly complicated and require our models to exhibit this property.

Many of these conditions are already satisfied by existing models, as I will discuss, but no model yet addresses all of them. This is my goal in this work.

Recurrently connected networks

I focus my attention exclusively on *recurrently connected* networks (opposed to multi-layer feed-foward networks) for two reasons: 1) biological neural circuits are highly recurrent, with feed-back loops at multiple spatial scales; and 2) recurrent networks generally excel at tasks for which temporal structure or dynamics are relevant (which are the types of tasks that interest us).

The artificial neural circuits we study can be described by the following equation:

$$\tau \dot{\boldsymbol{x}} = -\boldsymbol{x} + \boldsymbol{J}\boldsymbol{h}. \tag{1.1}$$

Here $\boldsymbol{x}$ is an N-dimensional vector describing the state of the system, $\boldsymbol{J}$ is an $N \times N$ matrix describing the connections between units and $\boldsymbol{h}$ is a N-dimensional vector describing the output of each network unit. τ for all of these models sets the time scale of the dynamics. The relationship between $\boldsymbol{x}$ and $\boldsymbol{h}$ will be specific to the neuron model we are studying, as will be described below, but generally we can describe this function as a nonlinear mapping $H : \boldsymbol{x} \rightarrow \boldsymbol{h}$. (Note on mathematical notation: the notation I introduce here is meant to be general to encompass a fairly general class of neural circuits. Notation will be specific to each chapter given the details of the models being discussed and will not generally be identical to the notation introduced here). The specific interpretation of each mathematical variable will depend on the details of each model.

Continuous-variable models

The first network model we introduce—traditionally called "firing-rate" networks—are composed of neural "units". These models were introduced previously, from a more historical perspective. In the past, models of this form were generally low-dimensional (they had small numbers of neurons) or were not constructed to generate overly complicated dynamics. Here we introduce a more modern perspective on their mathematical form and function.

The dynamics of these models can be described by Equation 1.1 where $\boldsymbol{x}$ is typically referred to as an activation variable, coarsely analogous to the subthreshold activation of neuron. The input-output function $H(\cdot)$ nonlinearly maps the N-dimensional continuous state variable $\boldsymbol{x}(t)$ to a N-dimensional continuous output variable $\boldsymbol{h}(t)$ which is commonly referred to as the *firing-*

rate:

$$\boldsymbol{h}(t) = H(\boldsymbol{x}(t)). \tag{1.2}$$

Throughout this work $H(\cdot)$ for this class of network model is a hyperbolic tangent function (although this is not a necessary condition of these models). Models of this form, although extremely complex, are convenient from the viewpoint of mathematical analysis and supervised learning due to the differentiable nature of $H(\cdot)$. Modern models of this form will generally have upwards of hundreds to thousands of network units and given enough neurons, can approximate arbitrary dynamical systems [Sussillo, 2014].

At this point, we will refrain from referring to these models by their more traditional name of firing-rate models, instead adopting the term *continuous-variable networks*. The reasons behind this choice will become clear as we proceed.

Spiking models

Spiking networks are comprised of spiking model neurons that communicate solely by discontinuous events called action potentials. The model of an individual neuron can incorporate great biophysical detail including a diverse array of voltage activated conductances, sodium and potassium channels for biophysically realistic spike generation and intrinsic currents that give rise to spike frequency adaptation and facilitation effects [La Camera et al., 2004]. Since our primary interest in this work is to understand specifically how modifications in network connectivity can give rise to networks that can perform computations, we restrict our attention to the simplest

model neuron that generates spikes—the current-based leaky integrate-and-fire (LIF) neuron [Abbott, 1999; Brunel and van Rossum, 2007].

Like continuous-variable networks, the dynamics of a generic LIF network can be described by Equation 1.1 where $\boldsymbol{x}$ represents the subthreshold voltage of a neuron. However, unlike continuous-variable neuron models, the input-output function $H(\cdot)$ of LIF model neurons cannot be described by an analytic function; instead it is replaced by a spiking rule:

$$\begin{aligned} &\text{if} \quad && x_i \geq x_{\text{threshold}} \\ &\quad && x_i \rightarrow x_{\text{reset}} \\ &\quad && h_i \rightarrow h_i + 1 \\ &\text{else} \\ &\quad && \tau_h \dot{h}_i = -h_i \end{aligned} \tag{1.3}$$

For spiking networks, $\boldsymbol{h}$ corresponds to a synaptic current and the time-scale of its dynamics is determined by τ_h

Random connections

For both of the models we study, we initially choose the entries of the recurrent connectivity, $\boldsymbol{J}$, randomly. There are two reasons that this choice is commonly made.

First, practically speaking, we do not presently have access to sufficient connectivity data to

merit precise choices about connections. Therefore we need to make a modeling choice with regard to how they are selected. Although we generally assume that biological neural circuits are not *truly* randomly connected, it is safe to assume that they exhibit a fair amount of *degeneracy* [Marder and Taylor, 2011] which is to say that the specific wiring of a neural circuit could be changed drastically (for example, imagine multiplying $\boldsymbol{J}$ by an orthogonal matrix) without a drastic change to its function.

Also, it turns out that randomly connected networks give rise to many convenient dynamical properties for performing supervised learning as well as producing many of the most salient and desirable features of biological neural circuit dynamics. I discuss these properties below for the different network models we study. I will also return to the idea of random connectivity and its role in network modeling in Chapter Five.

Continuous-variable models

The seminal work of Sompolinsky et al. [1988] was perhaps the first significant study to address the role of random connectivity in continuous-variable networks. It was found that when the entries of $\boldsymbol{J}$ were distributed according to a Gaussian distribution with zero mean and variance g^2/N, the term g had a profound impact on the network dynamics. For $g > 1$, the dynamics of these networks are chaotic. The rich spatiotemporal dynamics exhibited by these networks is illustrated in Figure 1.6 (as well as Figure 1.8, before learning commences). This finding posed an opportunity and a complication with respect to performing supervised learning with these models; the chaotic dynamics implies that these networks are extremely rich and can potentially be used to generate complex output functions but their chaotic quality causes initially nearby state variables to diverge, meaning that generating reproducible outputs from these networks is

not possible.

The work of Rajan et al. [2010] addressed these shortcomings by noting the appearance of a phase transition from chaotic to stable dynamics when a strong external input was applied to these networks. We can augment our network model from Equation 1.1 to include this input:

$$\tau \dot{\boldsymbol{x}} = -\boldsymbol{x} + \boldsymbol{J}\boldsymbol{h} + \boldsymbol{u}_{\text{out}} f_{\text{out}}. \tag{1.4}$$

This discovery opened the door for using these networks as rich temporal basis sets for constructing complex outputs. These ideas will be addressed more fully in Training recurrent networks.

Before moving on, some points are worth making here with regard to our list of desirable network properties. Models of the form described above meet conditions 3-8 from our list above: they exhibit rich, complex spontaneous activity (6); heterogenous, dynamic responses when driven with an external input (3&4); and as will be discussed exhibit fluctuations in their firing-rates (7) that can be linearly decoded to solve tasks (8), making them amenable to supervised learning (5). Additionally, these networks exhibit a decrease in firing-rate fluctuations when a stimulus is applied, consistent with experimental observations [Churchland et al., 2010b].

Based on these findings, models of this form will be quite useful as we proceed in our efforts to include spiking neurons into a general model framework.

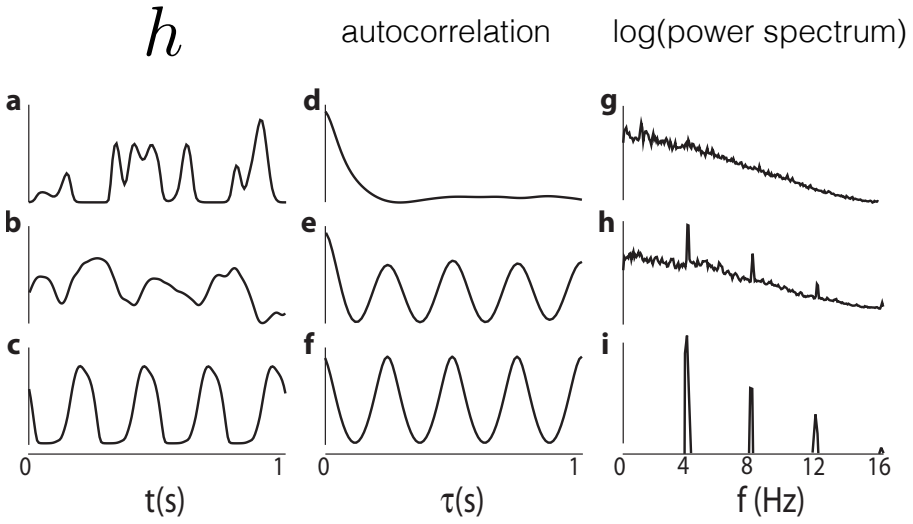

Figure 1.6: Input dependent suppression of chaos.
(**top row**) Firing-rate activity **(left)** of one unit in a chaotic neural network. **(middle)** The average autocorrelation function, computed across all units of the network, shows a typical slow decay, giving these models long memory. **(right)** Average power spectrum across the network shows fluctuations at a continuum of timescales. (**middle row**) Same as above, but for a network with a weak external stimulus applied. Signs of the external input can be observed in the average autocorrelation function, but some residual chaotic fluctuations remain. (**bottom row**) Same as above, but for a network with a strongly applied external stimulus. The chaotic fluctuations have been suppressed and the network is fully entrained to the stimulus. (adapted from Rajan et al. [2010])

Spiking models

As previously discussed, several studies have examined the properties of randomly connected spiking networks loosely collected into a framework called *balanced networks* [Brunel, 2000; Litwin-Kumar and Doiron, 2012; Renart et al., 2010; van Vreeswijk and Sompolinsky, 1996]. The principal motivation of these studies was to replicate the high degree of spiking irregularity seen from *in vivo* cortical recording [Shadlen and Newsome, 1994; Softky and Koch, 1993], and these networks achieved that goal with considerable success. The activity from a chaotic, balanced network is illustrated in Figure 1.7.

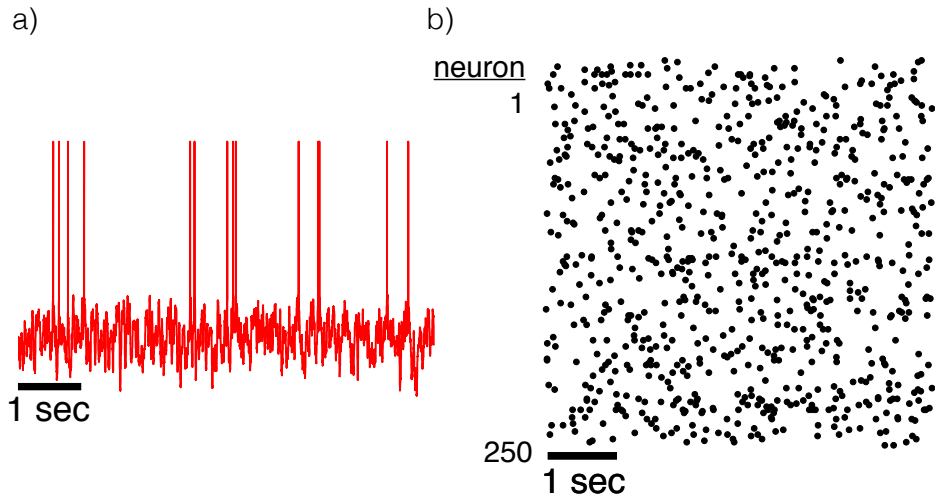

Figure 1.7: Irregular spiking in a balanced network.

(a) Irregular voltage fluctuations are generated, self-consistently, from the recurrent network dynamics. The Fano factor of this network is approximately one, consistent with data generated from a Poisson point-process and in agreement with experimental observations. **(b)** A raster of 250 LIF neurons (of 1,000) from a balanced LIF network exhibiting irregular spiking.

These networks, although impressive in their ability to capture the irregularity of spiking observed *in vivo*, fall short in many ways. While they do function as useful models of spontaneous activity (goal 6), and communicate with irregular spikes (1 & 2), they are unable to perform tasks (5 & 8) and their network dynamics more accurately reflect fixed point dynamics of a single continuous variable (thereby falling short of goals 3, 4 & 7). Although the work of Litwin-Kumar and Doiron [2012] addressed goals 3, 4 & 7 by introducing clustered connections into these networks, effectively creating groups of competing attractors, additional steps are needed to bring them in line with our goals.

As in randomly connected continuous models, some aspect of randomly connected spiking networks are desirable for the networks we seek to construct. Moving forward, we will exploit these properties to generate realistic-looking spiking variability, while seeking to augment these networks to address their shortcomings.

Training recurrent networks

I have thus far introduced models that exhibit interesting dynamics that relate to observations in biological neural circuits but that do not have any functionality. I now augment these models to incorporate learning on their connectivity so that they can perform tasks.

To do so, we introduce a target output function $f_{\text{out}}(t)$ which is a way of defining what we would like this neural circuit to "do". In order to define the output of our system, we introduce N-dimensional vector of output connections $\boldsymbol{w}$ and define the total output of the system as a linear sum of the output of each network unit, weighted by $\boldsymbol{w}$: $\boldsymbol{w}\boldsymbol{h}(t)$. The goal of learning is to modify $\boldsymbol{w}$ (and $\boldsymbol{J}$, if we can) so that the collective output of the network is as close as possible (in a least-squares sense) to the target output function $f_{\text{out}}(t)$. This statement can be described mathematically by the following equation:

$$\boldsymbol{w}\boldsymbol{h}(t) \approx f_{\text{out}}(t) \tag{1.5}$$

Given an appropriate set of functions $\boldsymbol{h}(t)$ this task is trivial. However, arriving at a functioning network model through learned modifications of the recurrent connections $\boldsymbol{J}$ that gives rise to a self-consistent set of functions $\boldsymbol{h}(t)$ *is not* trivial and presents the main challenge when attempting to construct recurrent neural networks.

The principal method for addressing the issue of training recurrent networks has historically been use the of the back-propagation algorithm, or, in the case of recurrent neural networks, back-propagation through time. Since our primary interest is constructing functional spiking

networks, as will become apparent, the use of back-propagation can pose problems. Specifically, in cases where the input-output function $H(\cdot)$ is not differentiable, as is the case in spiking neurons, this algorithm will fail. These methods have been summarized more completely elsewhere [Rumelhart and McClelland, 1986; Sutskever, 2013; Werbos, 1990] and constitute the bulk of a thesis in their own right, so we only refer to them in passing.

Target-based methods

The problem of constructing a recurrent system with gradient-based methods can be circumvented by the methods we introduce here. Generally, we refer to these methods as *target-based* methods.

The principle difficulty of performing supervised learning in recurrent networks is ensuring that $\boldsymbol{h}(t)$ meet two conditions: they must be 1) sufficient for generating the desired output function via the output connections $\boldsymbol{w}$; and 2) sufficient for supporting the activity of other neurons in the network and future activity states $\boldsymbol{h}(t+dt)$ through the recurrent connections $\boldsymbol{J}$. Gradient-based methods propagate errors backwards in time to identify an appropriate set of output functions for the recurrent units. In target-based learning the approach of propagating error gradients backwards in time is replaced by deriving a set of self-consistent target functions, which we call *auxiliary target functions* $\boldsymbol{f}_{\mathrm{J}}(t)$ for performing learning on $\boldsymbol{J}$. These targets can rightly be called a good "guess" at a set of dynamical activity patterns that a recurrent system could self generate through learned modifications to $\boldsymbol{J}$. It is interesting to note that a mixture of these two ideas, called target-propagation, has recently been developed [Bengio, 2014; Lee et al., 2015].

The principal question one needs to address for the success of target-based learning is to identify an appropriate set of targets that allows the network to collectively solve the task. Methods for

identifying these targets is the primary result of this work with regard to learning.

Training continuous-variable networks

For continuous-variable networks, Echo State Learning [Jaeger and Haas, 2004] and FORCE learning [Sussillo and Abbott, 2009] were introduced as alternative methods for training recurrent networks, and fall under the category of target-based learning. In these methods, learning was restricted only to $\boldsymbol{w}$ but these modifications ultimately led to a modification of $\boldsymbol{J}$ due to a feedback loop of the learned signal. These methods exploit the interaction of an initially chaotic system with a strong input (as found in Rajan et al. [2010]). The interaction of this input with the initial recurrent connectivity $\boldsymbol{J}$ gives rise to network output sufficient for target based learning.

Given a sufficient initial guess for $\boldsymbol{J}$, Equation 1.5 can be solved to a sufficient degree of accuracy, and we can replace the last term in Equation 1.4 with the solution from Equation 1.5. Doing so, (and then applying a bit of algebra) yields:

$$\tau \dot{\boldsymbol{x}} = -\boldsymbol{x} + (\boldsymbol{J} + \boldsymbol{u}_{\text{out}} \boldsymbol{w}) \boldsymbol{h}. \tag{1.6}$$

While at first sight, it may appear that FORCE learning is not altering the recurrent connectivity, we can re-define the recurrent connectivity as:

$$\hat{\boldsymbol{J}} = \boldsymbol{J} + \delta \boldsymbol{J} = \boldsymbol{J} + \boldsymbol{u}_{\text{out}} \boldsymbol{w}. \tag{1.7}$$

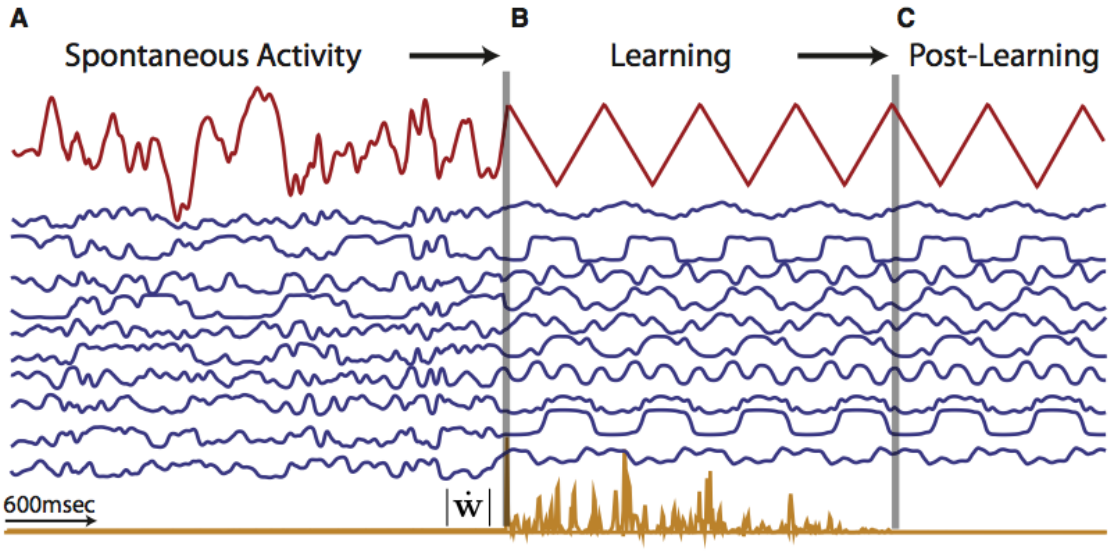

Figure 1.8: FORCE learning

(**a**) The rich, chaotic dynamics of $\boldsymbol{h}$ (blue) gives rise to a non-repeating network output $\boldsymbol{wh}$ (red). **b**) Learning is initiated and applied recursively. Modifications to $\boldsymbol{w}$ occurring rapidly (orange) and drive the network via the feedback projections $\boldsymbol{u}_{\text{out}}$. This feedback suppresses the chaotic dynamics within the network giving rise to a network output suitable for solving Equation 1.5. (**c**) Learning is halted, and the effective modifications to $\boldsymbol{J}$, $\delta\boldsymbol{J}$, have altered the recurrent connectivity so that f_{out} is produced stably without need for further modification. (adapted from Sussillo and Abbott [2009])

Defined in this way, it is apparent that FORCE learning has identified a recurrently connected network with connectivity $\hat{\boldsymbol{J}}$ that is a rank-1 perturbation to the initial connectivity $\boldsymbol{J}$. An example of FORCE learning is illustrated in Figure 1.8.

Chapter Four addresses additional steps that can be taken to improve learning in continuous-variable models using target-based learning and a random initialization for $\boldsymbol{J}$.

Training spiking networks

It is precisely because of the non-differential nature of the process H for spiking networks that constructing functional networks through supervised learning has been so challenging. With the aid of target-based learning, as we show in later chapters, the full power of supervised learning can be brought to the problem of building functional spiking networks.

Despite the complexity of training recurrent spiking networks, there have been notable successes in the past which we now discuss. These methods can coarsely be considered target-based methods since they often rely on approximating another system with a group of spiking neurons, and occasionally achieve this approximation through a least-square fitting procedure. Since we devote substantial attention to the technical details of training spiking networks in Chapter Two, here we only review the studies themselves, their achievements and the progress those achievements brought to the field.

Historically, the problem of constructing recurrent spiking neural networks that perform tasks was solved by first identifying a closely related (often reduced form) continuous model that embodied the structure of the task and then approximating that system with a spiking system. Early examples include memory systems that rely on attractor states [Amit and Brunel, 1997] or decision-making networks that rely on inhibition-mediated competition between attractors [Wang, 2002]. Since these systems performed tasks using fixed-point dynamics, static mean-field theory could be applied to transform the problem from the domain of spiking neurons to, essentially, a one-dimensional continuous equation. Later, these ideas were extended to continuous sets of fixed points, called *line attractors*, to model temporal integration and memory of continuous external variables like the position of a motor effector [Seung et al., 2000] or the value of a sensory stimulus [Machens et al., 2005]. Here again, a continuous dynamical model

was first invoked and then approximated by a group of spiking neurons. Ultimately, these approaches were further extended to arbitrary linear continuous dynamical systems, encompassing dampened and persistent oscillator and integrators, [Boerlin et al., 2013] or nonlinear dynamical systems with closed form continuous dynamics [Eliasmith, 2005] such as the Lorenz attractor.

As I will discuss in Chapter Three, our approach to building spiking networks follows this same line of research, where a continuous system is first identified that can be used as a helpful tool for building a more complex, spiking system.

Connecting continuous-variable and spiking models

Here I summarize two approaches we adopted in the early days of attempting to understand the relationship between continuous-variable networks and spiking networks. While these two approaches did not yield inspiring or terribly useful results, they were instructive about the operation of these models, the difficultly of performing supervised learning in spiking networks, and ultimately guided us to more useful approaches that we discuss later.

Hybrid firing-rate/spiking network

The first approach we adopted to connect firing rate networks to spiking networks took the form of a "hybrid" approach, where we considered the variables $\boldsymbol{h}$ of a rate network as the rate parameter of a Poisson process that gave rise to spike counts. This is a primitive way of including spiking noise into these models to examine what effect spikes might have on their dynamical properties and to see if learning could be achieved in the face of this noise. Methods similar

to this have more recently been studied by other research groups [Harish and Hansel, 2015; Kadmon and Sompolinsky, 2015].

To define the firing rate, we introduced a parameter $R_{\max}$ which determined exactly what the relative firing rate of $\boldsymbol{h}$ meant, since normally these quantities are scaled between -1 and 1. To generate Poisson distributed random variables from elements of $\boldsymbol{h}$ that took negative values, we would compute the absolute value of these quantities first, and then adjust for their sign after, in a sense computing "negative spike counts". The dynamics for the systems we studied took the following form:

$$\tau \dot{\boldsymbol{x}}_i = -\boldsymbol{x}_i + \sum_{j=1}^{N} \boldsymbol{J}_{ij} \frac{sgn(\boldsymbol{h}_j)}{R_{\max}\Delta t} Poiss(|\boldsymbol{h}_j| R_{\max}\Delta t) \tag{1.8}$$

We found that the introduction of Poisson spiking noise into these models was destructive to the slow chaotic fluctuations that can be observed in normal firing rate networks and that make them useful for supervised learning. This destructive quality can most clearly be seen by computing the average autocorrelation function of $\boldsymbol{h}$, which was plotted for a standard firing-rate network in Figure 1.6. For a network of 1,000 firing-rate units, with $g = 3$, $R_{\max}$ had to take on a value of 10,000 Hz (meaning that each firing rate unit is comprised of 100 spiking neurons, each firing at a maximum rate of 100 Hz, implying the entire network is comprised of 100,000 spiking neurons) before the decay of the average autocorrelation function resembled results seen in standard firing-rate networks. Furthermore, if we attempted standard FORCE learning in a network of this type, even for a trivial task, learning failed for smaller values of $R_{\max}$ (Figure 1.9).

These results indicated that the inclusion of spiking noise in continuous-variable networks is detrimental to the slow chaotic fluctuations that allow them to be useful for supervised learning.

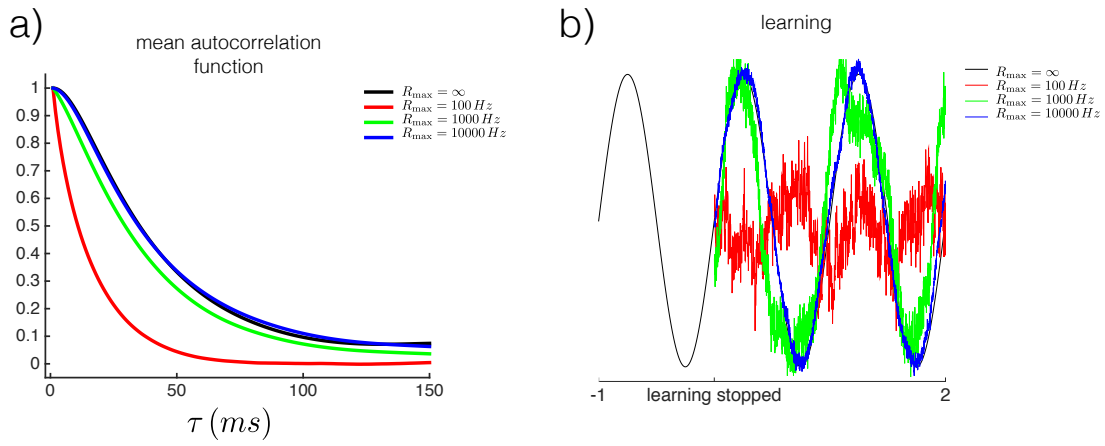

Figure 1.9: Hybrid rate/spiking model.

(**a**) The mean autocorrelation function for a variety of $R_{\max}$ values. (**b**) Attempting to learn a 1 Hz sine wave with networks of different $R_{\max}$ values. Black is the target and the output of a trained rate network.

Furthermore, these results raised several red flags about the connection of these models to spiking networks, which directly inspired my future work. As a note on this approach, the learning we attempted on these models was standard FORCE learning; new approaches to performing FORCE-like learning, to be introduced in Chapter Four could be more successful in these types of models.

The failure of reservoir approaches in LIF networks

Our second approach was to attempt to exploit randomly connected spiking networks as a reservoir of rich activity in the same way as had been done in continuous-variable networks. Naïvely, one might assume that the same tricks of reservoir computing can be exploited in spiking networks (in fact, we followed this path, fruitlessly, for some time). Unfortunately (or perhaps rather, fortunately, since it led us to better methods) this approach failed. Here we provide some intuition as to why reservoir computing methods work for continuous-variable models, and why

this is not a sensible thing to do in spiking networks (Figure 1.10).

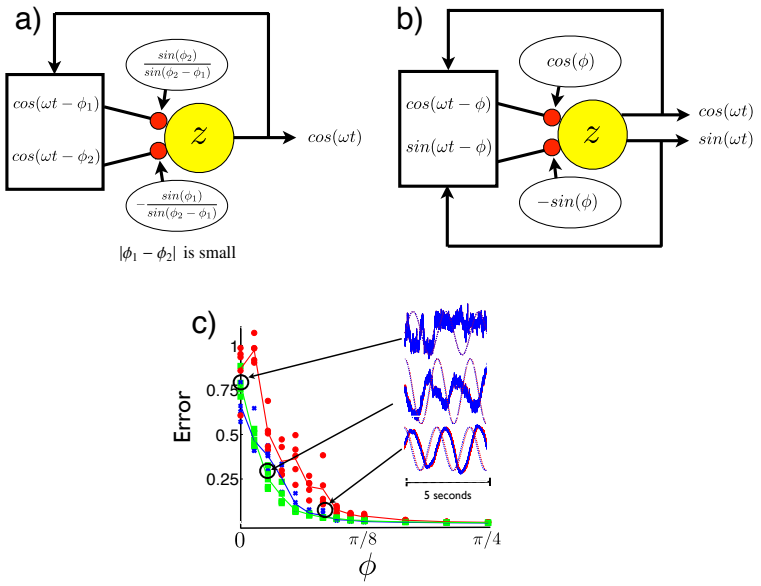

Figure 1.10: Intuition into reservoir methods, and why they fail in LIF networks.

A simplified diagram of a recurrent network acting as a reservoir and an external feedback neuron on which supervised learning is being performed. (**a**) When the output is a single cosine function, the reservoir must generate the phase-shifted signal. (**b**) Introducing a phase-shifted signal as a target provides the phase-shifted signal automatically. This signal is usually available for free in continuous-variable networks (for example, it can be read out explicitly from a network driven by the target function, without learning). (**c**) Test error after training a spiking network with a pair of periodic target functions with phases 0 to $\pi/4$. Each color shows a network with different parameter settings for the synaptic time constant. The inset shows the generated and desired output at various phase shifts. By a phase shift of $\pi/8$, the network can perform the task.

Consider a case in which the output is a single sine function. For this to happen, the interaction of the external feedback and the recurrent dynamics must be sufficient to create a phase-shifted version of the target function. If this phase shift is too small, the learned solution for $\boldsymbol{w}$ will grow large and lead to unstable dynamics. Firing-rate reservoirs can generate appropriate phase shifts, while randomly connected spiking networks do not. If we introduce a phase-shifted version of the target as a second target of learning, then, by definition, the phase-shifted signal is present

in the network activity and learning will succeed. This approach can be adopted in both spiking models and rate models, although it is not necessary in the latter.

From this initial intuition, we were able to construct various oscillating networks (including high-dimensional oscillators) (Figure 1.11a). From there, we went on to study the possibility of using firing-rate networks as sources of targets (which we discuss later). One tangent of research in this area that is left unresolved is to what degree the chaotic state of firing rate networks can be replicated in spiking networks by training them with chaotic rate trajectories. We found that doing so creates a spiking network that replicates some of the slow dynamics of the rate network, but includes additional high-frequency content, due to the spiking noise (Figure 1.11b).

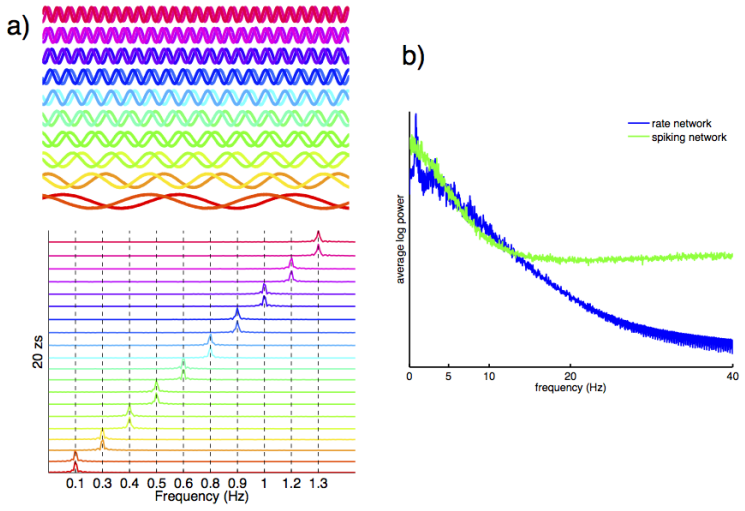

Figure 1.11: Training LIF networks with Fourier bases and chaotic rate networks.

(**a**) 10 sine-cosine pairs are used to train a spiking network. The signals during testing are shown, as well as the power spectrum of each signal, indicating that the correct target frequency is being generated. (**b**) The log of the average power spectrum of $\boldsymbol{h}$ for a chaotic rate network and a spiking network trained from signals from the rate network. The spiking network generates some of the slow fluctuations of the rate network, but has increased power in high frequencies from the spikes.

Since the days of studying these methods, other developments within the research community may warrant re-investigating them. Specifically, work has been done examining how the choice of model neuron (using a quadratic integrate-and-fire neuron for example [Brunel and Latham, 2003; Huh, 2015]) may effect a randomly connected spiking network's capacity to function as a reservoir. Another, contemporary approach, for training recurrent spiking networks is similar to the early approach we tried here, except that a nonlinearity is included in the synaptic dynamics, which may make it possible to use randomly connected spiking networks as reservoirs [Thalmeier et al., 2015]. Additionally, research into understanding the effects of synaptic timescales and connectivity strength on the dynamical properties of randomly connected LIF networks is an on-going field of study [Harish and Hansel, 2015; Kadmon and Sompolinsky, 2015; Ostojic, 2014]. New insights here could shed light on permitting the use of reservoir techniques in spiking networks.

Chapter 2

Building Functional Networks of Spiking Model Neurons [1]

Abstract

Most of the networks used by computer scientists and many of those studied by modelers in neuroscience represent unit activities as continuous variables. Neurons, however, communicate primarily through discontinuous spiking. We review methods for transferring our ability to construct interesting networks that perform relevant tasks from the artificial continuous domain to more realistic spiking network models. These methods raise a number of issues that warrant further theoretical and experimental study.

---

[1] This chapter was published as *Building functional networks of spiking model neurons*, Nature Neuroscience (2016). co-authored by L.F. Abbott[†,‡], Brian DePasquale[†] and Raoul-Martin Memmesheimer[†,§]. Copyright is held by Nature Publishing Group.

[†] Department of Neuroscience, Columbia University College of Physicians and Surgeons, New York, NY USA. [‡] Department of Physiology and Cellular Biophysics, Columbia University College of Physicians and Surgeons New York, NY USA. [§] Department of Neuroinformatics, Donders Institute for Brain Cognition and Behavior, Radboud University, the Netherlands.

Introduction

The world around us is described by continuous variables—distances, angles, wavelengths, frequencies—and we respond to it with continuous motions of our bodies. Yet the neurons that represent and process sensory information and generate motor acts communicate with each other almost exclusively through discrete action potentials. The use of spikes to represent, process and interpret continuous quantities and to generate smooth and precise motor acts is a challenge both for the nervous system and for those who study it. A related issue is the wide divergence between the timescales of action potentials and of perceptions and actions. How do millisecond spikes support the integration of information and production of responses over much longer times? Theoretical neuroscientists address these issues by studying networks of spiking model neurons. Before this can be done, however, network models with functionality over behaviorally relevant timescales must be constructed. Here, we review a number of methods that have been developed for building recurrent network models of spiking neurons.

Constructing a network requires choosing the models used to describe its individual neurons and synapses, defining its pattern of connectivity, and setting its many parameters (Figure 2.1a). The networks we discuss are based on model neurons and synapses that are, essentially, as simple as possible. The complexity of these networks resides in the patterns and strengths of the connections between neurons (although we consider dendritic processing toward the end of this Perspective article). This should not be interpreted as a denial of the importance or the complexity of the dynamics of membrane and synaptic conductances, or of phenomena such as bursting, spike-rate adaptation, neuromodulation and synaptic plasticity. These are undoubtedly important, but the simplified models we discuss allow us to assess how much of the dynamics needed to support temporally extended behaviors can be explained by network connectivity. Furthermore,

such models provide a foundation upon which more complex descriptions can be developed.

The problem we are addressing is this: a network receives an input $f_{\text{in}}(t)$, and its task is to generate a specified output $f_{\text{out}}(t)$ (Figure 2.1a; we discuss below how this output is computed). Our job is to configure the network so that it does this task, where by 'configure' we mean set the weights (that is, strengths) of the network synapses to appropriate values. For a network of N neurons, these weights are given by the elements of an $N \times N$ matrix, denoted by J, that describes the modifiable connections between network neurons (although some of these elements may be constrained to 0, corresponding to non-existent connections). We note here that we are constructing recurrently connected networks which pose unique challenges not faced when constructing feedforward networks. Given our interest in spanning the temporal gap between spikes and behavior, tasks of interest often involve integrating an input over time [Boerlin and Denève, 2011; Boerlin et al., 2013; Burak and Fiete, 2009; Eliasmith, 2005; Hansel and Sompolinsky, 1998; Lim and Goldman, 2013; Maass et al., 2007; Renart et al., 2003; Schwemmer et al., 2015; Seung et al., 2000; Song and Wang, 2005; Wang, 2002], responding to particular temporal input sequences [Buonomano and Merzenich, 1995; Gütig and Sompolinsky, 2006; Pfister et al., 2006], responding after a delay or with an activity sequence [Buonomano and Merzenich, 1995; DePasquale et al., 2016; Diesmann et al., 1999; Jahnke et al., 2012; Liu and Buonomano, 2009; Maass et al., 2002; Memmesheimer et al., 2014; Reutimann et al., 2004; Thalmeier et al., 2015; Vogels and Abbott, 2005], responding with a temporally complex output [Brea et al., 2013; DePasquale et al., 2016; Eliasmith et al., 2012; Florian, 2012; Hennequin et al., 2014; Maass et al., 2007; Memmesheimer et al., 2014; Ponulak and Kasiński, 2010; Thalmeier et al., 2015], or autonomously generating complex dynamics [Boerlin et al., 2013; DePasquale et al., 2016; Eliasmith, 2005; Memmesheimer et al., 2014; Thalmeier et al., 2015]. In this Perspective, we focus on general approaches that extend our ability to construct spiking networks capable of performing a wide variety of tasks or that make spiking networks perform these tasks more accurately.

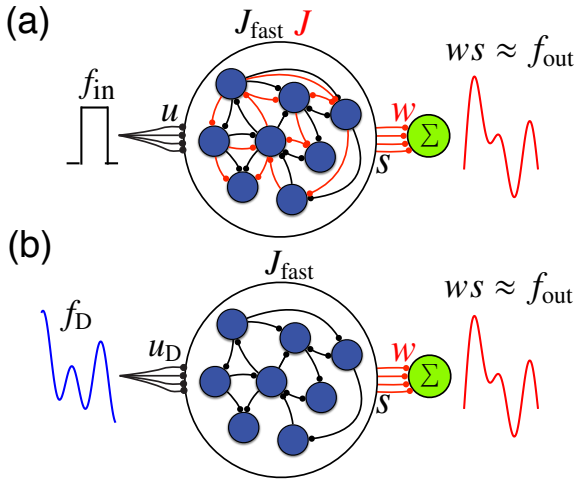

Figure 2.1: Structure of autonomous and driven networks.

(**a**) The autonomous network. In this diagram, black lines and dots denote fixed connections and red lines and dots are connections that are adjusted to make the network function properly. A defined input f_{in} is provided to the network through connections characterized by weights u. Neurons in the network are connected by two types of synapses, parameterized by J_{fast} (in black) and J (in red). The problem is to choose the strengths of the synapses defined by J, and the weights w, so that the output of the network, ws, approximates a given target output f_{out}. (**b**) The driven network. In this case, the network is driven by input f_{D}, through weights u_{D}, that forces it to produce the desired output. Only the fixed synapses denoted by J_{fast} are included. Output weights are adjusted as in the autonomous network.

Determining the connection matrix required to make a network perform a particular task is difficult because it is not obvious what the individual neurons of the network should do to generate the desired output while supporting each others' activities. This is the classic credit assignment problem of network learning: what should each individual neuron do to contribute to the collective cause of performing the task? The field of machine learning has addressed credit assignment by developing error gradient-based methods, such as back-propagation, that have been applied with considerable success [LeCun et al., 2015]. This approach has also been used to construct abstract network models known as rate models in which neurons communicate with each other through continuous variables [Mante et al., 2013; Sussillo et al., 2015]. Unfortunately, the application of gradient-based methods to spiking networks [Bohte et al., 2002; Florian, 2012; Sporea and Grüning, 2013; Tino and Mills, 2006] is problematic because it has not been clear

how to define an appropriate differentiable error measure for spike trains. The methods that we review can all be thought of as ways to solve the credit assignment problem without resorting to gradient-based procedures.

Defining the input, output and network connections

Before beginning the general discussion, we need to explain how neurons in the network interact, how they receive an input, and how they produce an output. This, in turn, requires us to define what we call the normalized synaptic current, $s(t)$, that arises from a spike train (Figure 2.2a, top and middle traces). In the synapse model we use, each presynaptic spike causes the normalized synaptic current to increase instantaneously by 1, $s \rightarrow s + 1$. Between spikes, s decays exponentially toward 0 with a time constant τ, which is set to 100 ms in the examples we show. There is one normalized synaptic current for each network neuron, so s is an N-component vector. The normalized synaptic current is used to construct both the output of the network and the inputs that each neuron receives through the synapses described by the matrix J (Figure 2.1a). The synaptic current generated in a postsynaptic neuron by a particular presynaptic neuron is given by the appropriate synaptic weight times the normalized synaptic current for that presynaptic neuron. The synaptic currents for all the network neurons are given collectively by Js.

All of the models we present have, in addition to the connections described by J, a second set of synapses with time constants considerably faster than τ, described by J_{fast}. We consider two arrangements for these fast synapses: random or set to specific values (see below). In either case, the fast synapses are not modified as part of the adjustments made to J to get the network to perform a particular task. It is tempting to equate the fast (J_{fast}) and slower (J) synapses in these models to fast AMPA and slower NMDA excitatory synapses or to fast GABA_A and slower

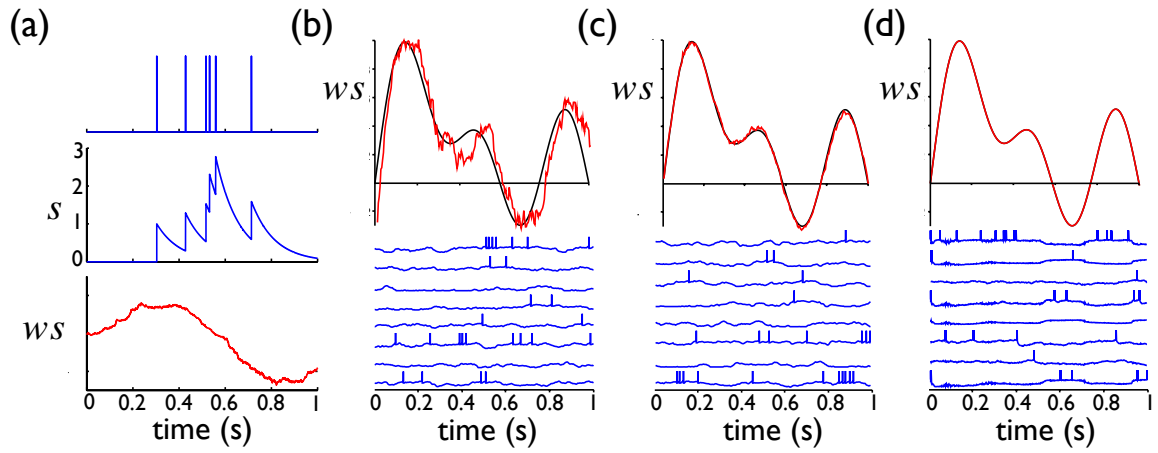

Figure 2.2: Driven networks approximating a continuous target output.

(**a**) Spike train from a single model neuron (top), the normalized synaptic current $s(t)$ that it generates (middle), and the output ws computed from a weighted sum of the normalized synaptic currents from this neuron and 99 others (bottom). (**b-d**) Results from driven networks with optimally tuned readout weights. In each, the upper plot shows the actual output ws in red and the target output f_{out} in black, and the lower plot shows representative membrane potential traces for 8 of the 1000 integrate-and-fire model neurons in each network. Neurons in the driven network are connected by fast synapses with random weights for **b** and **c** and with weights adjusted according to the spike-coding scheme for **d**. The three panels show the outputs in response to a driving input $f_{\text{D}}=f_{\text{out}}$ (**b**), a driving input $f_{\text{D}}=f_{\text{out}}+\tau df_{\text{out}}/dt$ in a rate-coding network (**c**), and a driving input $f_{\text{D}}=f_{\text{out}}+\tau df_{\text{out}}/dt$ in a spike-coding network (**d**).

GABA_{B} inhibitory synapses. Although this is correct as far as timescales are concerned, there are issues with this interpretation due to the different way these two classes of synapses are treated and modified in the models. These remain to be resolved.

The input to the network, f_{in}, takes the form of a current injected into each neuron. This current is $f_{\text{in}}(t)$ times a neuron-dependent weight. The vector formed by all N of these weights is denoted by u (Figure 2.1a; we could also extend these networks to include multiple inputs f_{in} but, for conciseness, we restrict our examples to single-input cases.)

The network output is a weighted sum of the normalized synaptic currents generated by all the neurons in the network (Figure 2.2a, bottom trace; we consider a single output here, but extend

this to multiple outputs later). Each network neuron has its own output weight in this sum and, collectively, these weights form an N-component row vector w (Figure 2.1a). Output weights are adjusted to minimize the average squared difference between the actual output, ws, and the desired output f_{out}.

In all of the examples we show, the firing rates of all the network neurons are constrained to realistic values. Another important element in producing realistic-looking spike trains is trial-to-trial variability. Irregular spiking can be generated internally through the random fast synapses we include [Brunel, 2000; van Vreeswijk and Sompolinsky, 1996], or by injecting a noise current into each network neuron. We do both here.

The spiking networks we discuss come in two varieties that we call rate-coding and spike-coding. At various points we also discuss what are called rate networks, more abstract models in which network units communicate through continuous variables, not spikes. It is important to keep in mind that the rate-coding case we discuss refers to spiking, not rate, networks.

Driven networks

In any construction project, it is useful to have a working example. As circular as it sounds, one way to construct a network that performs a particular task is by copying another network that does the task. This approach avoids circularity because the example network involves a cheat; it is driven by an input $f_{\text{D}}(t)$ that forces it to produce the desired output (Figure 2.1b). We call the original network—the one we are constructing (Figure 2.1a)—the autonomous network (even though it receives the external input f_{in}) and call the example network the driven network (Figure 2.1b). If there is a single driving input, it is injected into the network neurons through

weights described by a vector u_D. Later we will discuss situations in which $P > 1$ driving inputs are used. In this case, u_D is an $N \times P$ matrix. Although the driven network does not receive the original input f_{in} directly, the driving input f_D typically depends on f_{in}, as discussed below. The autonomous and the driven networks contain the same set of fast synapses, but the slower synapses described by the matrix J are absent in the driven network.

The role of the driven network is to provide targets for the autonomous network. In other words, we will construct the autonomous network so that the synaptic inputs to its neurons match those in the driven network. In machine-learning a related scheme is known as target propagation [Bengio, 2014; LeCun, 1986], and interesting neural models have been built by extracting targets from random networks [Laje and Buonomano, 2013] or from experimental data [Fisher et al., 2013; Rajan et al., 2016].

Obviously, a critical issue here is how to determine the driving input that forces the driven network to perform a task properly. We address this below but, for now, just assume that we know what the driving input should be. Then, the driven network solves the credit assignment problem for us; we just need to examine what the neurons in the driven network *are* doing to determine what the neurons in the autonomous network *should* do. Even better, the driven network tells us how to accomplish this: we just need to arrange the additional connections described by J so that, along with the term uf_{in}, they produce an input in each neuron of the autonomous network equal to what it receives from the external drive in the driven network. However, there are significant challenges in seeing this program through to completion: 1) We have to figure out what f_D is—in other words, determine how to drive the driven network so that it performs the task. 2) We must assure that this input can be self-generated by the autonomous network with a reasonable degree of accuracy. 3) We must determine the recurrent connection weights J that accomplish this task, and 4) We must assure that the solution we obtain is stable

with respect to the dynamics of the autonomous network. This Perspective covers significant progress that has been made in all four of these areas.

The driven network consists of nonlinear, spiking model neurons connected by either randomly chosen or specifically set (as discussed below) fast synapses (Figure 2.1b), and the spikes produced by these units are filtered (Figure 2.2a) and summed (Figure 2.1) to provide the output. The transformation from the input f_D to the output f_{out} might seem to be quite complex, but it turns out that the effects of nonlinearities and fast network connections can largely be compensated by appropriate choice of the output weights w. In light of this, a first guess for the driving input might be to set $f_D = f_{out}$—that is, to treat the network as if it simply passes a signal from the input to a properly extracted output. This approach can generate good results in rate-based networks [Jaeger and Haas, 2004; Lukosevicius et al., 2012; Sussillo, 2014; Sussillo and Abbott, 2009, 2012], and it has been tried in spiking networks [Maass et al., 2007], but in these it only works in limited cases and, in general, poorly (Figure 2.2b).

A significant advance [Eliasmith, 2005; Eliasmith and Anderson, 2003] was the realization that the element of the network input-output transformation that cannot be compensated by the choice of output weights is the synaptic filtering at the output, characterized by the time-constant τ. Correcting for this synaptic low-pass filtering and its phase delay is easy: we simply define the driving input as a high-pass-filtered, phase-advanced version of the desired output,

$$f_D = f_{out} + \tau \frac{df_{out}}{dt}. \tag{2.1}$$

Using this driving input to produce f_{out} works quite well (Figure 2.2c). Equation 2.1, which provides an answer to challenge 1, forms the basis for the work we discuss. Of course, this only gives

us a driven version of the network we actually want. In the following sections, we show how to make the transition from the driven network (Figure 2.1b) to the autonomous network (Figure 2.1a). Before doing this, however, we introduce an approach that allows the desired output to be produced with greatly enhanced accuracy.

Spike coding to improve accuracy

The network output shown in (Figure 2.2c, red line, top panel) matches the target output (Figure 2.2c, f_{out}, black line, top panel) quite well, but deviations can be detected, for example, at the time of the second peak of f_{out}. Some deviations are inevitable because we are trying to reproduce a smooth function with signals $s(t)$ that jump discontinuously every time there is a spike (Figure 2.2a). In addition, deviations may arise from irregularities in the patterns of spikes produced by the network. The driven network in Figure 2.2c approximates the desired output function because its neurons fire at rates that rise and fall in relation to changes in the function f_{out}. For this reason, we refer to networks of this form as rate coding. Deviations between the actual and desired outputs occur in these networks when a few more or a few less spikes are generated than the precise number needed to match the target output. The spike-coding networks that we now introduce [Boerlin and Denève, 2011; Boerlin et al., 2013; Schwemmer et al., 2015] work on the same basic principle of raising and lowering the firing rate, but they avoid generating excessive or insufficient numbers of spikes by including strong fast interactions between neurons. These interactions replace the random fast connections used in the network of Figure 2.2b,c with specifically designed and considerably stronger connections [Denève and Machens, 2016]. In general, both excitatory and inhibitory strong fast synapses are required. These synapses cause the neurons to spike in a collectively coherent manner and assure near-optimal performance. For

a rate-coding network of N neurons, the deviations between the actual and desired output are of order $1/\sqrt{N}$. In spike-coding networks, these deviations are of order $1/N$, a very significant improvement (as can be seen by comparing the outputs in Figure 2.2c,d). The values of the fast strong connections needed for spike coding were derived as part of a general analysis of how to generate a desired output from a spiking network optimally [Boerlin and Denève, 2011; Boerlin et al., 2013]. The use of integrate-and-fire neurons, equation 2.1 for the optimal input, a determination of the optimal output weights w, and the idea and form of the fast connections are all results of this interesting analysis.

The strength of the fast synapses used in the spike-coding scheme is reflected in the way they scale as a function of the number of synapses that the network neurons receive. Denoting this number by K, one way of assuring a fixed level of input onto a neuron as K increases is to make synaptic strengths proportional to $1/K$. The inability of this scheme to account for neuronal response variability [Softky and Koch, 1993] led to the study of networks [Brunel, 2000; van Vreeswijk and Sompolinsky, 1996] in which the synaptic strengths scale as $1/\sqrt{K}$. Maintaining reasonable firing rates in such networks requires a balance between excitation and inhibition. The fast synapses in spiking-coding networks have strengths that are independent of K, imposing an even tighter spike-by-spike balance between excitation and inhibition to keep firing levels under control.

Spike-coding networks implement the concept of encoding information through precise spiking in a far more interesting way than previous proposals. The spike trains of individual neurons in spike-coding networks can be highly variable (through the injection of noise into the neurons, for example) without destroying the remarkable precision of their collective output. This is because, if a spike is missed or a superfluous one is generated by one neuron, other neurons rapidly adjust their spiking to correct the error.

In the following sections, we discuss both spike-coding and rate-coding variants of networks solving various tasks. All of the networks contain fast synapses, but for the rate-coding networks these are random and relatively weak, and their role is to introduce irregular spiking, whereas for the spike-coding networks they take specifically assigned values, are strong and produce precise spiking at the population level. Another important difference is that the elements of the input vector u and recurrent synaptic weights given by J are considerably larger in magnitude for spike-coding networks than in the rate-coding case.

Autonomous networks

It is now time to build the autonomous network and, to do this, we must face challenges 2-4: how can we arrange the network connections so that the *external* signal f_{D} that allows the driven network (Figure 2.1b) to function properly can be produced *internally* and stably by the autonomous network (Figure 2.1a)? One way to assure that the autonomous network can generate the driving input needed to produce f_{out} is to place restrictions on f_{D}. Because $f_{\mathrm{D}} = f_{\mathrm{out}} + \tau d f_{\mathrm{out}}/dt$, this also restricts f_{out} and thus limits the complexity of the tasks that the network can perform. We discuss these restrictions and ways to get around them in the following sections.

Because the autonomous network receives the input f_{in} and, if it works properly, produces a good approximation of the desired output f_{out}, one sensible restriction on f_{D} is to require it to be a linear combination of f_{in} and f_{out}. This imposes the requirement that

$$f_{\mathrm{out}} + \tau \frac{d f_{\mathrm{out}}}{dt} = B f_{\mathrm{out}} + u_{\mathrm{R}} f_{\mathrm{in}}, \tag{2.2}$$

where B and u_R are constants. Because $ws \approx f_{out}$, we can write the current that each neuron in the driven network receives from the driving input, using Equation 2.2, as $u_D f_D \approx u_D B w s + u_D u_R f_{in}$. For the autonomous network to work properly, these currents must be reproduced in the absence of the driving input by the combination of recurrent and input currents $Js + uf_{in}$. Equating the driving and autonomous currents, we see that the autonomous network can be constructed by setting $u = u_D u_R$ and $J = u_D B w$. This solves challenge 3 [Boerlin and Denève, 2011; Boerlin et al., 2013; Eliasmith, 2005; Eliasmith and Anderson, 2003].

If $B = 1$, the two terms involving f_{out} in Equation 2.2 cancel, and f_{out} is then proportional to the time integral of f_{in}. The construction we have outlined thus produces, in this case, a spiking network that integrates its input, fairly accurately in the rate-coding case (Figure 2.3a) and very accurately for the spike-coding version (Figure 2.3b). Integrating networks have been constructed prior to the development of the approaches we are presenting [Burak and Fiete, 2009; Hansel and Sompolinsky, 1998; Maass et al., 2007; Renart et al., 2003; Seung et al., 2000; Song and Wang, 2005; Wang, 2002]; the key advances are that the same methods can be used for more complex tasks and that, in the case of spike-coding, accuracy is greatly improved.

For a single function f_{out}, Equation 2.2 can only describe a low-pass filter or an integrator, but a somewhat broader class of functions can be included by extending f_{out} from a single function to a vector of P different functions, while maintaining the restriction that f_D depends linearly on f_{out}. In this extension, B in Equation 2.2 is a $P \times P$ matrix, u_R is a P-component vector, u_D is an $N \times P$ matrix, and w is a $P \times N$ matrix. The same approach discussed above, but extended to $P > 1$, allows us to build networks that generate a set of P free-running, damped and/or driven oscillations [Boerlin et al., 2013; Eliasmith, 2005; Eliasmith and Anderson, 2003].

Even with the extension to oscillations, the networks we have discussed thus far are highly lim-

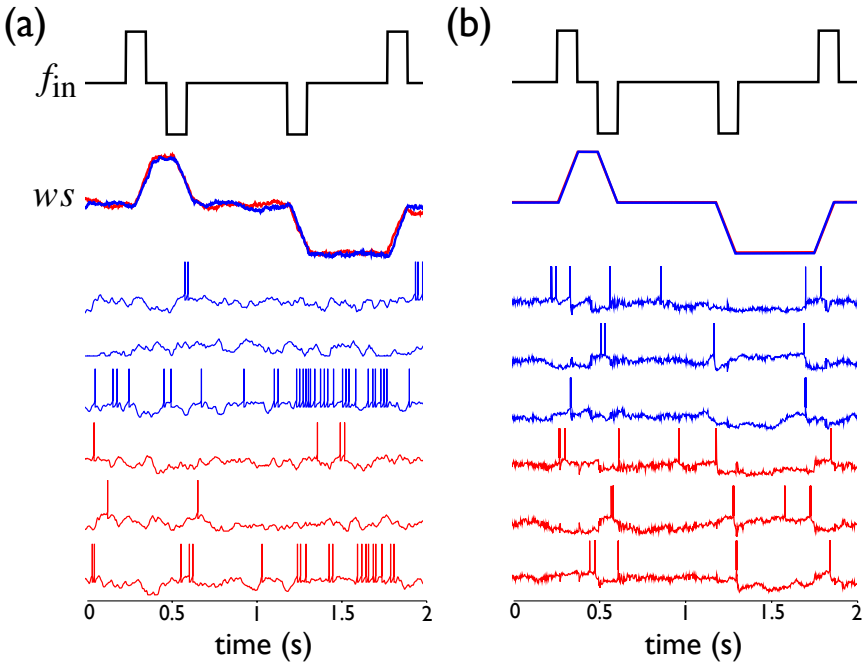

Figure 2.3: Two autonomous networks of spiking neurons constructed to integrate the input f_{in} (top, black traces). (**a**) A rate-coding network. (**b**) A spike-coding network. For each network, the results from two trials are shown. The upper red and blue traces marked ws show the output of the networks on these two trials (they overlap almost perfectly in panel **b** and are therefore difficult to distinguish) and the bottom blue and red traces show the membrane potentials of 3 neurons in the networks on the two trials. Note the trial-to-trial variability in the spiking patterns. Each network consists of 1000 model neurons.

ited. This is due to the restriction we placed on f_D by requiring it to be linear in f_{out}. To expand functionality, we must loosen this restriction while continuing to ensure that the autonomous network can generate the signals comprising f_D. Suppose we allow f_D, instead, to be a nonlinear function of f_{out}. In this case, Equation 2.2 is replaced by

$$f_{out} + \tau \frac{df_{out}}{dt} = BH(f_{out}) + u_R f_{in}, \tag{2.3}$$

where H is a nonlinear function (tanh for example). As in the linear case, we know that $f_{out} \approx ws$, so the driving current into each neuron of the driven network in the nonlinear case is $u_D f_D \approx$

$u_D BH(ws) + u_D u_R f_{in}$. Equating this to the analogous current in the autonomous network, $Js + uf_{in}$, we find that the input weights of the autonomous network are again given by $u = u_D u_R$, but the recurrent circuitry of the network must reproduce the currents given by $u_D BH(ws)$, which, unlike the expression Js, are not linear in s. There are two approaches for dealing with this problem.

The first approach is to modify the spiking neuron model used in the network to include dendritic nonlinearities, meaning that the recurrent input to the neurons of the autonomous network is given by a more complex expression than Js. We implement this by considering the different pieces from which the current $u_D BH(ws)$ is constructed. The term ws can be interpreted as N inputs weighted by the components of w summed on P nonlinear dendritic processes. The function H is then interpreted as a dendritic nonlinearity associated with these processes, and the remaining factor, $u_D B$, describes how the P dendrites are summed to generate the total recurrent synaptic input into the soma of each network neuron. Modifying the neuron model in this way and using a spike-coding scheme, this approach has been developed as a general way to build spiking network models that can be modified easily to perform a wide variety of tasks [Thalmeier et al., 2015].

The second approach sticks with the original neuron model, uses a rate-coding approach, and solves the condition $Js \approx u_D BH(f_{out})$ by a least-squares procedure [DePasquale et al., 2016; Eliasmith, 2005; Seung et al., 2000]. This can work in the nonlinear case because, although the expression Js is linear in s, the normalized synaptic current is generated by a nonlinear spike-generation process. In particular, this process involves a threshold, which supports piecewise approximations to nonlinear functions [Seung et al., 2000]. To avoid stability problems that may prevent such a solution from producing a properly functioning network, the least-squares procedure used to construct J should be recursive and run while the network is performing the

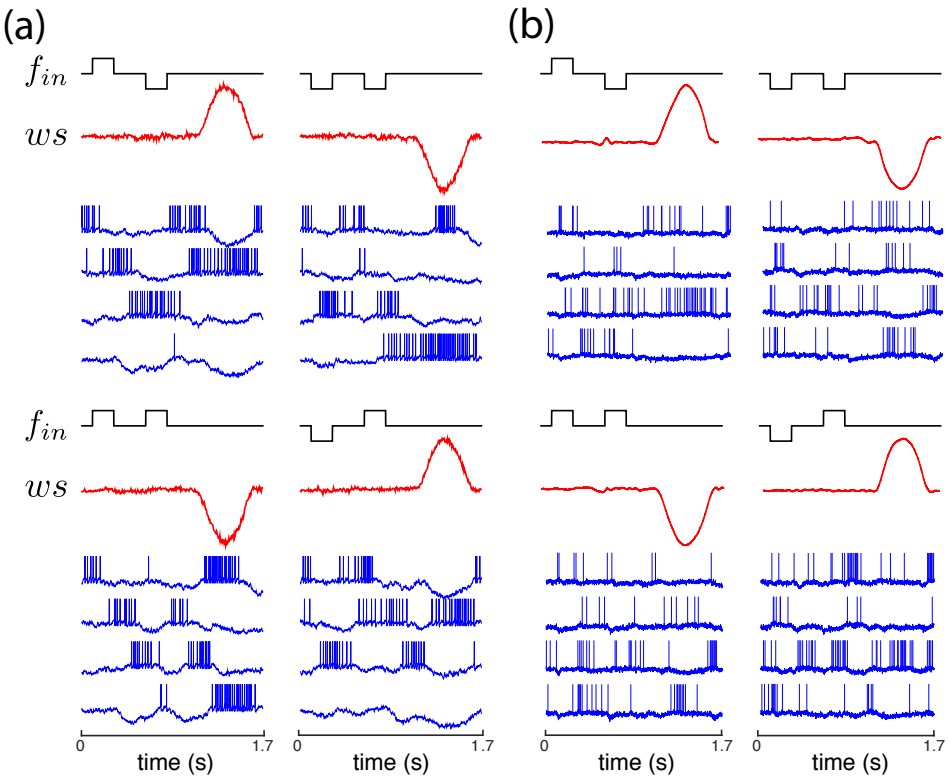

Figure 2.4: Autonomous networks solving a temporal XOR task.

(**a**) A rate coding network with linear neuronal input integration. (**b**) A spike coding network with nonlinear neuronal input integration. In both cases, the network output (red traces) is a delayed positive deflection if two successive input pulses have different signs and is a negative deflection if the signs are the same. Blue traces show the membrane potentials of 4 neurons in the networks.

task, using an RLS or FORCE algorithm [DePasquale et al., 2016; Sussillo and Abbott, 2009, 2012]. Although stability is not guaranteed [Rivkind and Barak, 2015], this approach works well in practice, effectively resolving challenge 4.

Figure 2.4 shows examples of a rate-coding network with linear integrate-and-fire neurons (Figure 2.4a) and a spike-coding network that includes dendritic nonlinearities (Figure 2.4b) built according to the procedures discussed in this section and performing a temporal XOR task.

The connection to more general tasks

At this point, the reader may well be wondering what Equation 2.3 has to do with the tasks normally studied in neuroscience experiments. Tasks are typically defined by relationships between inputs and outputs (given this input, produce that output) not by differential equations. How can we use this formalism to construct spiking networks that perform tasks described in a more conventional way? The answer lies in noting that Equation 2.3 defines a P-unit rate model, that is, a model in which P nonlinear units interact by transmitting continuous signals (not spikes) through a connection matrix B. Continuous-variable (rate) networks can perform a variety of tasks defined conventionally in terms of input-output maps if P is large enough [Jaeger and Haas, 2004; Lukosevicius et al., 2012; Sussillo, 2014; Sussillo and Abbott, 2009, 2012]. This observation provides a general method for constructing spiking networks that perform a wide range of tasks of interest to neuroscientist [DePasquale et al., 2016; Thalmeier et al., 2015]. In this construction, the continuous-variable (rate) network plays the role of a translator, translating the conventional description of a task in terms of an input-output map into the differential equation description (Equation 2.3) needed to construct a spiking network [DePasquale et al., 2016]. For the spike-coding network with nonlinear dendrites [Thalmeier et al., 2015], the continuous variable (rate) model is built into the spiking network, and this allows the network to be quickly and easily re-adjusted to perform a variety of tasks. The rate-coding networks with linear integrate-and-fire neurons do not require precise dendritic targeting or dendritic nonlinearities, but their recurrent connectivity requires more radical readjustment to allow the networks to perform a new task [DePasquale et al., 2016]. In both cases, the power of recurrent continuous variable (rate) networks is used to enhance the functionality of a spiking network.

Discussion

We have reviewed powerful methods for constructing network models of spiking neurons that perform interesting tasks [Boerlin et al., 2013; DePasquale et al., 2016; Eliasmith, 2005; Thalmeier et al., 2015]. These models allow us to study how spiking networks operate despite high degrees of spike-train variability and, in conjunction with experimental data, they should help us identify the underlying signals that make networks function.

We have outlined several steps that may be used in the construction of functioning spiking networks, and it is interesting to speculate whether these have analogs in the development of real neural circuits for performing skilled tasks. One step was to express the rules and goals of the task in terms of the dynamics of a set of interacting units described by continuous variables. In other words, the rules of the task are re-expressed in terms of a system of first-order differential equations (Equation 2.3). It is interesting to ask whether task rules are represented in real neural circuits in the language of dynamics; finding such a representation in experimental data would provide a striking confirmation of the principles of network construction we have discussed. Continuous-variable (rate) networks not only play a key role in the construction of these spiking networks, they also describe the fundamental dynamic signals by which the spiking networks operate. This makes them well-suited for describing how neural circuits operate, not mechanistically (spiking networks are closer to this), but at a basic functional level.

Our discussion also introduced a driven network that could be used to guide the construction of an autonomous network, and it is interesting to ask whether this step has any biological counterpart. A possible parallel between the driven and autonomous networks we have discussed is the transition from labored and methodical initial performance of a task to automatic and virtually

effortless mastery. In the spiking network models, this transformation occurs when an external driving input is reproduced by an internally generated signal. After this transformation takes place, the external signal can be either removed or ignored. Plasticity mechanisms acting within neural circuits may, in general, act to assure that irrelevant signals are ignored and predictable signals are reproduced internally [Bourdoukan and Denève, 2015; Bourdoukan et al., 2012; Hosoya et al., 2005; Kennedy et al., 2014; Vogels et al., 2011]. The nature and mode of action of such mechanisms should help us replace the least-squares adjustment of synaptic weights we have discussed with more biophysically realistic forms of plasticity. An alternative might be provided by reward-based learning rules such as reward modulated synaptic plasticity [Friedrich and Senn, 2012; Hoerzer et al., 2014; Potjans et al., 2009; Vasilaki et al., 2009].

The spike-coding variants that we have discussed [Boerlin et al., 2013; Thalmeier et al., 2015] are unlikely to operate over a brain-wide scale. Instead, such networks may exist as smaller special purpose circuits operating with high accuracy. Their experimental signatures are strong and dense inter-connectivity. The challenge will be to identify the set of neurons that are part of such a circuit. Finally, the nonlinear version of spike-coding networks that we discussed [Thalmeier et al., 2015] involves both functional clustering of synapses and dendritic nonlinearities. Synaptic clustering has been reported [Branco and Häusser, 2011; Druckmann et al., 2014; Kleindienst et al., 2011], but it remains to be seen if this has the precision needed to support the required dendritic computations. Dendritic nonlinearities of various sorts abound [London and Häusser, 2005; Major et al., 2013] and, in this regard, it is important to note that a wide variety of nonlinear functions H can support the computations we have discussed.

The ability to construct spiking networks that perform interesting tasks opens up many avenues for further study. These range from developing better methods for analyzing spiking data to studying how large neuronal circuits operate and how different brain regions communicate and

cooperate. We hope that future reviewers will be able to cover exciting developments in these areas.

Acknowledgments

We thank C. Machens, M. Churchland and D. Thalmeier for helpful discussions. Research supported by the NIH grant MH093338 and by the Gatsby Charitable Foundation through the Gatsby Initiative in Brain Circuitry at Columbia University, the Swartz Foundation, the Harold and Leila Y. Mathers Foundation, the Kavli Institute for Brain Science at Columbia University, the Max Kade Foundation, and the German Federal Ministry of Education and Research BMBF through the Bernstein Network (Bernstein Award 2014).

Chapter 3

Using Firing-Rate Dynamics to Train Recurrent Networks of Spiking Model Neurons [1]

Abstract

Recurrent neural networks are powerful tools for understanding and modeling computation and representation by populations of neurons. Continuous-variable or “rate" model networks have been analyzed and applied extensively for these purposes. However, neurons fire action potentials, and the discrete nature of spiking is an important feature of neural circuit dynamics. We show how continuous-variable models can be used as a basis for training spiking network models to perform relevant tasks. We present a procedure for training such networks to generate complex dynamical patterns, to produce complex temporal outputs based on integrating network input, and to model physiological data. Our procedure makes use of a continuous-variable network to identify targets for training the inputs to the spiking model neurons. Surprisingly, we

---

[1] Portions of this chapter were published as *Using Firing-Rate Dynamics to Train Recurrent Networks of Spiking Model Neurons*, arXiv pre-print (2016). co-authored by Brian DePasquale[†], Mark M. Churchland[†,‡] and L.F. Abbott[†,§]

[†] Department of Neuroscience, Columbia University College of Physicians and Surgeons, New York, NY USA. [‡] Grossman Center for the Statistics of Mind, Columbia University College of Physicians and Surgeons, New York NY USA. [§] Department of Physiology and Cellular Biophysics, Columbia University College of Physicians and Surgeons, New York NY USA.

are able to construct spiking networks that duplicate tasks performed by continuous-variable networks with only a relatively minor expansion in the number of neurons. Our approach provides a novel view of the significance and appropriate use of “firing-rate" models, and it is a useful approach for building model spiking networks that can be used to address important questions about representation and computation in neural systems.

Introduction

A fundamental riddle of nervous system function is the disparity between our continuous and comparatively slow sensory percepts and motor actions and the neural representation of those percepts and actions by brief, discrete and spatially distributed action potentials. A related puzzle is the reliability with which these signals are represented despite the variability of neural spiking across nominally identical performances of a behavior. A useful approach to addressing these issues is to build spiking model networks that perform relevant tasks, but this has proven difficult to do. Here we develop a method for constructing functioning networks of spiking model neurons that perform a variety of tasks while embodying the variable character of neuronal activity. In this context, “task" refers to a computation performed by a biological neural circuit.

There have been previous successes constructing spiking networks that perform specific tasks [Hennequin et al., 2014; Machens et al., 2005; Seung et al., 2000; Wang, 2002]. In addition, more general procedures have been developed [Abbott et al., 2016] that construct spiking networks that duplicate systems of linear [Boerlin and Denève, 2011; Boerlin et al., 2013; Eliasmith, 2005] and nonlinear [Eliasmith, 2005; Thalmeier et al., 2015] equations. However, most tasks of interest to neuroscientists, such as action choices based on presented stimuli, are not expressed in terms of systems of differential equations. Our work uses continuous-variable network models

[Sompolinsky et al., 1988], typically called “rate" networks, as an intermediary between conventional task descriptions in terms of stimuli and responses and spiking network construction. This results in a general procedure for constructing spiking networks that perform a wide variety of tasks of interest to neuroscience [Abbott et al., 2016; Thalmeier et al., 2015]. We apply this procedure to example tasks and show how constraints on the sparseness and sign (Dale’s law) of network connectivity can be imposed. We also build a spiking network model that matches multiple features of data recorded from neurons in motor cortex and from arm muscles during a reaching task.

Results

The focus of our work is the development of a procedure for constructing recurrently connected networks of spiking model neurons. We begin by describing the model-building procedure and then present examples of its use.

Network architecture and network training

The general architecture we consider is a recurrently connected network of N leaky integrate-and-fire (LIF) model neurons that receives task-specific input $f_{\text{in}}(t)$ and, following training, produces an approximation of a specified “target" output signal $f_{\text{out}}(t)$ (Figure 3.1a). f_{in} can be thought of as external sensory input or as input from another neural network, and f_{out} as the input current into a downstream neuron or as a more abstractly defined network output (for example a motor output signal or a decision variable).

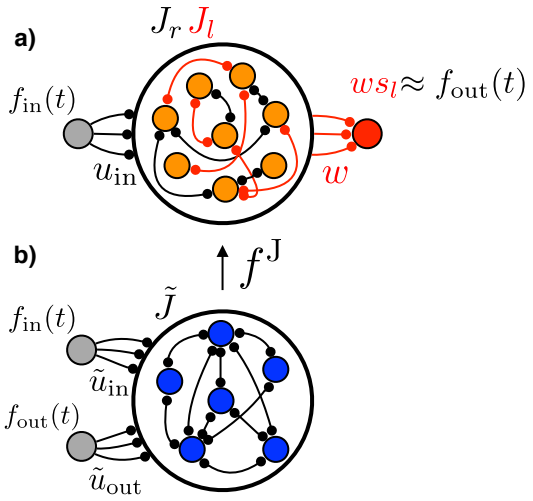

Figure 3.1: Network architectures.

a) Spiking network. A network of N recurrently connected leaky integrate-and-fire neurons (orange circles) receives an input f_{in} (grey circle) through synapses $\boldsymbol{u}_{\text{in}}$, and generates an output f_{out} (red circle) through synapses $\boldsymbol{w}$. Connections marked in red (recurrent connections $\boldsymbol{J}_l$ and output connections $\boldsymbol{w}$) are modified by training, and black connections are random and remain fixed. **b)** Continuous-variable network. A network of $\tilde{N}$ recurrently connected "rate" units (blue circles) receive inputs f_{in} and f_{out} through synapses $\tilde{\boldsymbol{u}}_{\text{in}}$ and $\tilde{\boldsymbol{u}}_{\text{out}}$, respectively. All connections are random and held fixed. The sum of $\tilde{\boldsymbol{u}}_{\text{out}} f_{\text{out}}$ and the recurrent input determined by $\tilde{\boldsymbol{J}}$ defines the auxiliary targets $\boldsymbol{f}_{\text{J}}(t)$ for the spiking network.

When a neuron in the network fires an action potential, it contributes both fast and slow synaptic currents to other network neurons. These currents are described by the two N-dimensional vectors, $\boldsymbol{f}$ and $\boldsymbol{s}$, respectively. When neuron i in the network fires an action potential, component i of both $\boldsymbol{s}$ and $\boldsymbol{f}$ is incremented by 1, otherwise

$$\tau_{\text{s}} \frac{d\boldsymbol{s}}{dt} = -\boldsymbol{s} \quad \text{and} \quad \tau_{\text{f}} \frac{d\boldsymbol{f}}{dt} = -\boldsymbol{f}\,. \tag{3.1}$$

The two time constants determine the decay times of these slow and fast synaptic currents, and we set $\tau_{\text{s}} = 100$ ms and $\tau_{\text{f}} = 5$ ms.

The neurons in the network are connected to each other by synapses with strengths denoted

by the $N \times N$ matrix $\boldsymbol{J}^{\mathrm{f}}$ for the fast synapses and the $N \times N$ matrix $\boldsymbol{J}^{\mathrm{s}}$ for the slow synapses. The elements of these matrices are chosen randomly from a Gaussian distribution with mean $\mu_{\mathrm{f}}/N\tau_{\mathrm{f}}\nu$ and variance $g_{\mathrm{f}}^2/N\tau_{\mathrm{f}}\nu$ and mean $\mu_{\mathrm{s}}/N\tau_{\mathrm{s}}\nu$ and variance $g_{\mathrm{s}}^2/N\tau_{\mathrm{s}}\nu$ respectively. This random connectivity produces chaotic spiking in the network [Brunel, 2000; vanVreeswijk and Sompolinsky, 1998], which we use as a source of spiking irregularity and trial-to-trial variability. We use the parameters g_{f} and g_{s} to control the level of this variability and the parameters μ_{f} and μ_{s} to control the firing rate of the network. These parameters take the following values in all of the examples we show (unless stated otherwise): $\mu_{\mathrm{f}} = -4.5$ mV, $g_{\mathrm{f}} = 5.5$ mV, $\mu_{\mathrm{s}} = 0$ mV, $g_{\mathrm{s}} = 1.0$ mV, $\nu = 15Hz$.

During network training a subset n of the connections of $\boldsymbol{J}^{\mathrm{f}}$ and a subset n of the connections of $\boldsymbol{J}^{\mathrm{s}}$ are modified. To keep our notation simple, we collect all the synapses that undergo learning, from both fast ($\boldsymbol{J}^{\mathrm{f}}$) and slow ($\boldsymbol{J}^{\mathrm{s}}$) synapses, into a $N \times 2n$ dimensional matrix $\boldsymbol{J}_l$. Likewise, both the slow ($\boldsymbol{s}$) and fast ($\boldsymbol{f}$) synaptic currents that project to other neurons via these synapses are collected into a $2n$-dimensional vector $\boldsymbol{s}_l$. The remaining random synapses and synaptic currents (both slow and fast) that will not undergo learning are also collected, denoted by the $N \times (N-2n)$ matrix $\boldsymbol{J}_r$ and a $(N-2n)$-dimensional vector $\boldsymbol{s}_r$.

The network output is constructed only from the synaptic currents that undergo learning. Connections between the network and the output have strengths given by an $N_{\mathrm{out}} \times 2n$ matrix $\boldsymbol{w}$, where N_{out} is the number of outputs (either 1 or 2 in the examples we provide).

The membrane potentials of the model neurons, collected together into a N-component vector $\boldsymbol{V}$, obey the equation

$$\tau_{\mathrm{m}} \frac{d\boldsymbol{V}}{dt} = \boldsymbol{V}_{\mathrm{rest}} - \boldsymbol{V} + g\boldsymbol{J}_l\boldsymbol{s}_l + \boldsymbol{J}_r\boldsymbol{s}_r + \boldsymbol{u}_{\mathrm{in}}f_{\mathrm{in}} + I\,, \tag{3.2}$$

with $\tau_m = 10$ ms. For a case with N_{in} inputs, $\boldsymbol{u}_{in}$ is an $N \times N_{in}$ matrix (we consider $N_{in} = 1$ and 2) with elements drawn independently from a uniform distribution between -1 and 1. The parameter g controls the strength of the learned input into each neuron, and $g = 6$ for the examples we show. I is a time-independent bias current that is set for each neuron to keep its mean firing rate within a biological realistic level by controlling the overall mean input into each neuron. It is increased by an amplitude of 0.25 for all neurons between trials in the examples of Figure 3.3 and Figure 3.4, representing a "holding" input between trials. Each neuron fires an action potential when its membrane potential reaches a threshold $V_{th} = -55$ mV and is then reset to $V_{reset} = V_{rest} = -65$ mV.

We can now specify the goal and associated challenges of network training. The goal is to modify the entries of $\boldsymbol{J}_l$ and $\boldsymbol{w}$ so that the network performs the task specified by f_{in} and f_{out}, meaning that

$$\boldsymbol{w}\boldsymbol{s}_l \approx f_{out} \tag{3.3}$$

when the network responds to f_{in} (with the approximation being as accurate as possible). Equation 3.3 stipulates that $\boldsymbol{s}_l$ must provide a basis for the function f_{out}. If it does, it is straightforward to compute the optimal $\boldsymbol{w}$ by minimizing the squared difference between the two sides of Equation 3.3, averaged over time. This can be done either recursively [Haykin, 2002] or using a standard batch least-squares approach.

Determining the optimal $\boldsymbol{J}_l$ is significantly more challenging because of the recurrent nature of the network. $\boldsymbol{J}_l$ must be chosen so that the learned component of the input to the network neurons, $\boldsymbol{J}_l\boldsymbol{s}_l$, generates a pattern of spiking that produces $\boldsymbol{s}_l$. The circularity of this constraint is what makes recurrent network learning difficult. The difference between the easy problem of computing $\boldsymbol{w}$ and the difficult problem of computing $\boldsymbol{J}_l$ is that, in the case of $\boldsymbol{w}$, we have the target f_{out} in Equation 3.3 that specifies what signal $\boldsymbol{w}$ should produce. For $\boldsymbol{J}_l$, it is not obvious

what the input it generates should be.

Suppose that we *did* have targets analogous to f_{out}, but for computing $\boldsymbol{J}_l$ (we call them auxiliary target functions and denote them by the N-component vector $\boldsymbol{f}_{\text{J}}(t)$). Then, $\boldsymbol{J}_l$, like $\boldsymbol{w}$, could be determined by a least-squares procedure, that is, by minimizing the time-averaged squared differences between the two sides of

$$\boldsymbol{J}_l \boldsymbol{s}_l \approx \boldsymbol{f}_{\text{J}}(t). \tag{3.4}$$

There are stability issues associated with this procedure, that we discuss below, however the main challenge in this approach is to determine the appropriate auxiliary target functions. Our solution to this problem is to obtain them from a continuous-variable model. More generally, if we can train or otherwise identify another model that implements a solution to a task, we can use signals generated from that model to train our spiking network.

Using continuous-variable models to determine auxiliary target functions

Equation 3.4 and Equation 3.3, respectively, summarize two key features of the vector of functions $\boldsymbol{f}_{\text{J}}(t)$: 1) They should correspond to the inputs of a recurrently connected dynamic system, and 2) they should provide a basis for the network output f_{out}. To satisfy the first of these requirements, we identify $\boldsymbol{f}_{\text{J}}(t)$ with the inputs of a recurrently connected continuous-variable "rate" network. These networks have been studied intensely [Rajan et al., 2010; Sompolinsky et al., 1988] and have been trained to perform a variety of tasks [Jaeger and Haas, 2004; Laje and Buonomano, 2013; Sussillo, 2014; Sussillo and Abbott, 2009]. To satisfy the second condition, we use the desired spiking network output, f_{out}, as an *input* to the rate network. This allows us to obtain the auxiliary target functions without having to train the continuous variable network. Although this is not guaranteed to work, as we will discuss, it generally works in practice for

most f_{out} of interest to us.

The continuous-variable model we use is a randomly connected network of $\tilde{N}$ firing-rate units (throughout we use tildes to denote quantities associated with the continuous-variable network). Like the spiking networks, these units receive the input f_{in} and, as mentioned above, they also receive f_{out} as an input. The continuous-variable model is described by an $\tilde{N}$-component vector $\boldsymbol{x}(t)$ that satisfies the equation

$$\tau_x \frac{d\boldsymbol{x}(t)}{dt} = -\boldsymbol{x}(t) + \tilde{\boldsymbol{J}} H(\boldsymbol{x}(t)) + \tilde{\boldsymbol{u}}_{\text{out}} f_{\text{out}} + \tilde{\boldsymbol{u}}_{\text{in}} f_{\text{in}} + \tilde{I}, \tag{3.5}$$

where $\tau_x = 10$ ms, H is a nonlinear function (we use $H(\cdot) = \tanh(\cdot)$), and $\tilde{J}$, $\tilde{\boldsymbol{u}}_{\text{in}}$, and $\tilde{\boldsymbol{u}}_{\text{out}}$ are matrices of dimension $\tilde{N} \times \tilde{N}$, $\tilde{N} \times N_{\text{out}}$ and $\tilde{N} \times N_{\text{in}}$, respectively. The elements of these matrices are chosen independently from a Gaussian distribution of zero mean and variance $\tilde{g}^2/\tilde{N}$ for $\tilde{\boldsymbol{J}}$, and a uniform distribution between -1 and 1 for $\tilde{\boldsymbol{u}}_{\text{in}}$ and $\tilde{\boldsymbol{u}}_{\text{out}}$. We set $\tilde{g} = 1.4$. $\tilde{I}$ is a bias term, defined analogously to the bias term into the spiking network. $\tilde{N} = 1000$ for the examples we show.

To be sure that signals from this driven network are appropriate for training the spiking model, the continuous-variable network, driven by the target output, should be capable of producing a good approximation of f_{out}. To check this, we can test whether an $N_{\text{out}} \times \tilde{N}$ matrix can be found (by least squares) that satisfies $\tilde{\boldsymbol{w}} H(\boldsymbol{x}(t)) \approx f_{\text{out}}$ to a sufficient degree of accuracy. Provided $\tilde{\boldsymbol{J}}$ and $\tilde{\boldsymbol{u}}_{\text{out}}$ are appropriately scaled, this can be done for a wide range of tasks [Sussillo, 2014].

The auxiliary target functions $\boldsymbol{f}_{\text{J}}(t)$ that we seek are generated from the inputs to the neurons in the continuous-variable network. There is often, however, a mismatch between the dimensions of $\boldsymbol{f}_{\text{J}}(t)$, which is N, and of the inputs to the continuous-variable model, which is $\tilde{N}$. Additionally, by examining the $\tilde{N} \times \tilde{N}$ covariance matrix of the continuous-variable network input, it becomes

apparent that its inputs are highly correlated and can therefore be described by a smaller set of functions of dimension $\tilde{n}$ by performing Principal Components Analysis (PCA) on these signals.

To deal with these issues, we introduce an $N \times \tilde{N}$ matrix $\boldsymbol{u}_{\mathrm{J}}$, with elements drawn independently from a uniform distribution between -1 and 1, and a $\tilde{n} \times \tilde{N}$ matrix $\tilde{u}_{PC}$ composed of the first $\tilde{n}$ eigenvectors of the covariance matrix of the input into each continuous variable unit as its rows. With these definitions we can write:

$$\boldsymbol{f}_{\mathrm{J}}(t) = \boldsymbol{u}_{\mathrm{J}} \tilde{u}_{PC}^{T} \tilde{u}_{PC} \left(\tilde{\boldsymbol{J}} H(\boldsymbol{x}(t)) + \tilde{\boldsymbol{u}}_{\mathrm{out}} f_{\mathrm{out}} \right). \tag{3.6}$$

From this expression, we can note that $\boldsymbol{f}_{\mathrm{J}}(t)$ is an N-dimensional vector of temporal functions, where each function is a random combination of the first $\tilde{n}$ PCs of the input into every unit of the firing-rate network. Each temporal function in $\boldsymbol{f}_{\mathrm{J}}(t)$ is used as a training signal for the input into each spiking neuron.

We leave out the input term proportion to f_{in} in this expression because the spiking network receives the input f_{in} directly. This set of target functions satisfies both of the requirements listed at the beginning of this section and, as we show in the following examples, allows functional spiking networks to be constructed by finding connections $\boldsymbol{J}_l$ that satisfy Equation 3.4. We do this initially by a recursive least-squares algorithm [Haykin, 2002], but later we discuss solving this problem by batch least-squares instead.

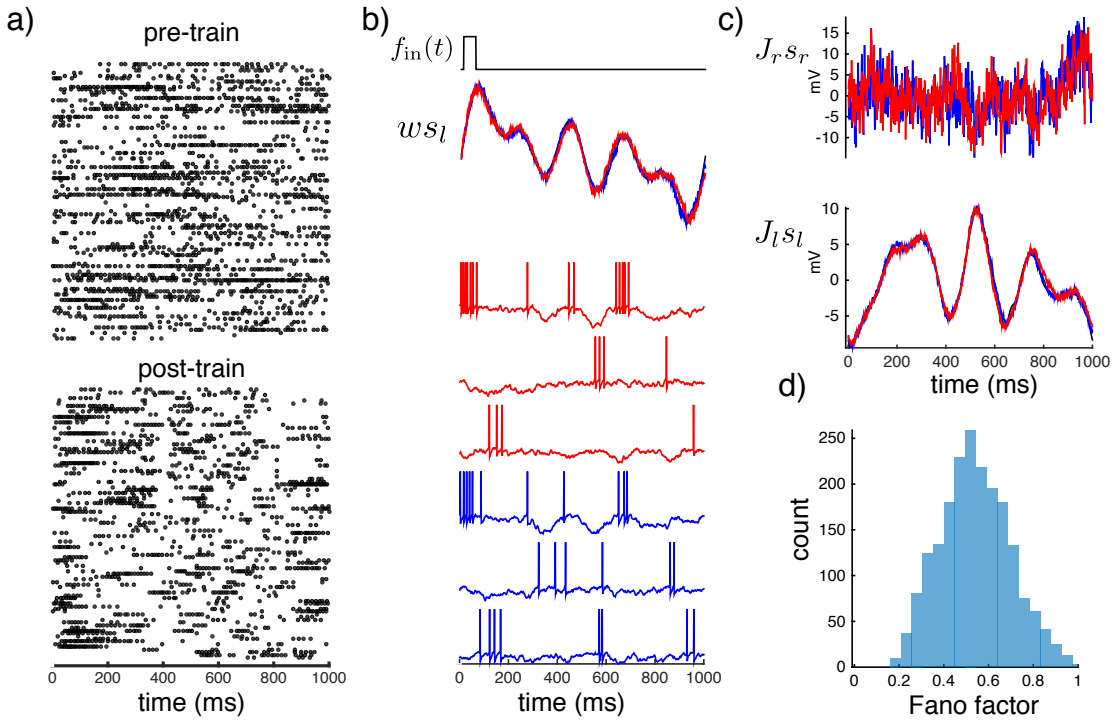

Figure 3.2: Results for a network trained on the oscillation task.

(a) Raster plot of 100 neurons before (top) and after (bottom) training. **(b)** Input f_{in}, target output f_{out} (black) and network output $\boldsymbol{w}\boldsymbol{s}_l$ for two trials (red and blue) (top). Example voltage traces for three neurons for two trials (red and blue). **(c)** Random recurrent input $\boldsymbol{J}_r\boldsymbol{s}_r$ (top) and learned recurrent input $\boldsymbol{J}_l\boldsymbol{s}_l$ into one neuron for two trials (red and blue). **(d)** Fano factor distribution computed across the population. Parameter for this example: $N = 2000$ $n = 400$.

Examples of trained networks

The procedure described above can be used to construct networks that perform a variety of tasks. We present three examples that range from tasks inspired by problems of relevance to neuroscience to modeling experimental data.

Our first example is an oscillation task that requires the network to generate a self-sustained, temporally complex output (Figure 3.2).

f_{out} for this task is a periodic function created by summing sine functions with frequencies of 1, 2, 3, and 5 Hz and with an amplitude of 1.5. f_{in} is a 50 ms brief pulse of amplitude 0.5 at the beginning of each period to compensate for phase drift than can accumulate when learning periodic tasks. Complex, oscillatory dynamics are a feature of neural circuits involved in repetitive motor acts such locomotion [Marder, 2000].

Initially the activity of the network is determined by the random synaptic input provided by $\boldsymbol{J}_r$, and the neurons exhibit irregular spiking (Figure 3.2a). Following the training procedure, the learned postsynaptic currents $\boldsymbol{J}_l \boldsymbol{s}_l$ closely match their respective auxiliary target functions (Figure 3.2c) and the network output similarly matches the target f_{out} (Figure 3.2b). Residual chaotic spiking due to $\boldsymbol{J}_r$ (Figure 3.2c) and the fact that we are approximating a continuous function by a sum of discontinuous functions cause unavoidable deviations. Nevertheless, a network of 2,000 LIF neurons firing at an average rate of 15 Hz with an average Fano factor of 0.43 (computed using 100 ms bins) performs this task with normalized post-training error of 5% (this error is the variance of the difference between $\boldsymbol{w}\boldsymbol{s}_l$ and f_{out} divided by the variance of f_{out}). This can be achieved when only modifying 20% of the recurrent synapses ($n = 400$).

Because the output for this first task can be produced by a linear dynamical system, previous methods could also have been used to construct a functioning spiking network [Boerlin et al., 2013; Eliasmith, 2005]. However, this is no longer true for the following examples. In addition, it is worth noting that the network we have constructed generates its output as an isolated periodic attractor of a nonlinear dynamical system. The other procedures, in particular that of Boerlin et al. [2013], create networks that reproduce the linear dynamics that generates f_{out}. This results in a system that can produce not only $\boldsymbol{w}\boldsymbol{s}_l \approx f_{\text{out}}$, but also $\boldsymbol{w}\boldsymbol{s}_l \approx \alpha f_{\text{out}}$ over a continuous range of α values. This often results in a slow drift in the amplitude of $\boldsymbol{w}\boldsymbol{s}_l$ over time. The point here is that our procedure solves a different problem than previous procedures, despite the fact

that it generates the same output. The previous procedures were designed to duplicate the linear dynamics that produce f_{out}, whereas our procedure duplicates f_{out} uniquely.

The second task we present is a temporal XOR task that requires the network to categorize the input it receives on a given trial and report this decision through its output. Each trial for this task consists of a sequence of two pulses appearing as the network input f_{in} (Figure 3.3). Pulses have amplitudes of 0.5 and 0.35, respectively, and their duration can be either short (100 ms) or long (300 ms). The two pulses are separated by 300 ms, and after an additional 300 ms delay the network must respond with either a positive or a negative pulse (with a shape given by 1/2 cycle of a 1 Hz sine function of amplitude 1.5). The rule for choosing a positive or negative output is an exclusive OR function of the input sequence (short-short $\rightarrow -$, short-long $\rightarrow +$, long-short $\rightarrow +$, long-long $\rightarrow -$). The time between trials and the input sequence on each trial are chosen randomly.

A network of 2,000 LIF neurons with an average firing rate of 15 Hz can perform this task correctly on 95% of trials by only changing 30% of the synapses ($n = 600$). As in the oscillation task, individual neuron spiking activity varies from trial-to-trial due to the effect of $\boldsymbol{J}_r$. The Fano factor computed across all neurons, all analysis times, and all task conditions is 0.61. This task requires integration of each input pulse, storage of a memory of the first pulse at least until the time of the second pulse, memory of the decision during the delay period before the output is produced, and classification according to the XOR rule.

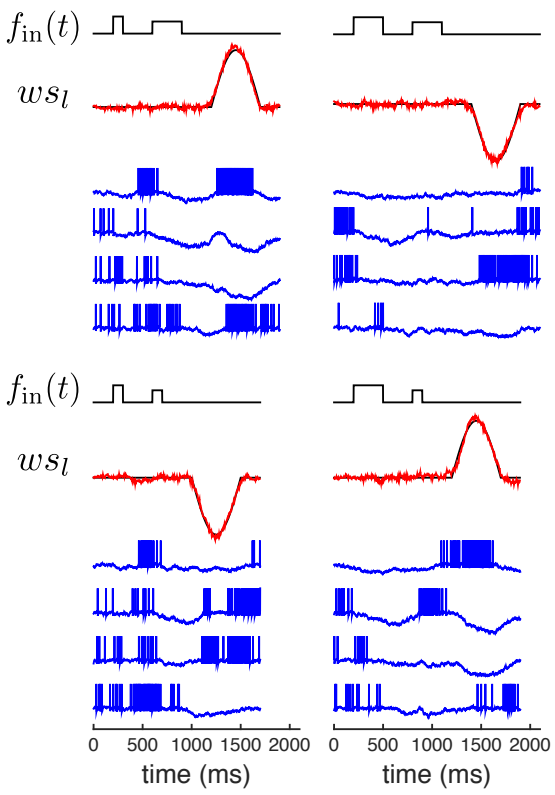

Figure 3.3: Temporal XOR task.

The input f_{in} (black) consists of two pulses that are either short or long in duration. The output ws_l (red) should report an XOR function of the combination of pulses. Membrane potentials of 4 example neurons (blue) are shown for the 4 different task conditions. Parameters for this example: $N = 2000$, $n = 600$.

Generating EMG activity during reaching

We now turn to an example based on data from an experimental study, with the spiking network generating outputs that match electromyograms (EMGs) recorded in 2 arm muscles of a non-human primate performing a reaching task [Churchland et al., 2016]. In this task, a trial begins with the appearance of a target cue at one of eight possible reach directions (task conditions). After a short delay, during which the arm position must be held fixed, a "go" cue appears, instructing a reach to the target. The time between trials and the sequence of reach directions are

varied randomly.

To convey information about target location to the model, we use two network inputs denoted by a two-component vector f_{in} and with amplitudes $\sqrt{2}\cos(\theta)$ and $\sqrt{2}\sin(\theta)$ where the angle θ specifies the reach direction (Figure 3.4a, left). The input is applied for 500 ms and, when it terminates, the network is required to generate two outputs (thus f_{out} is also two-dimensional) that match trial-averaged and smoothed EMG recordings from the anterior and posterior deltoid muscles during reaches to the specified target (Figure 3.4a, right & e). EMGs were filtered with a 4 ms Gaussian window.

A network of 2,000 neurons with an average firing rate of 15 Hz performs this task with a normalized post-training error of 7% (Figure 3.4e), consistent with another modeling study that used a firing-rate network [Sussillo et al., 2015]. This level of performance was achieved when only modifying 40% of the connections ($n = 800$). The activity of the trained network exhibits several features consistent with recordings from neurons in motor cortex. Individual neurons show a large amount of spiking irregularity that is variable across trials and conditions (Figure 3.4b). The Fano factor computed across all neurons and all task conditions drops during task execution (Figure 3.4d), consistent with observations across a wide range of cortical areas [Churchland et al., 2010b]. The Fano factor computed across all time, conditions and neurons was 0.63. This network shows that EMGs can be generated by a network with a realistic degree of spiking variability.

Another interesting feature of the network activity is the variety of different neural responses. Individual neurons display tuning during the input period, the output period, or across multiple epochs with different tunings (Figure 3.4f). As in recordings from motor cortex, consistently tuned neurons represent a minority of the network; most neurons change their tuning during

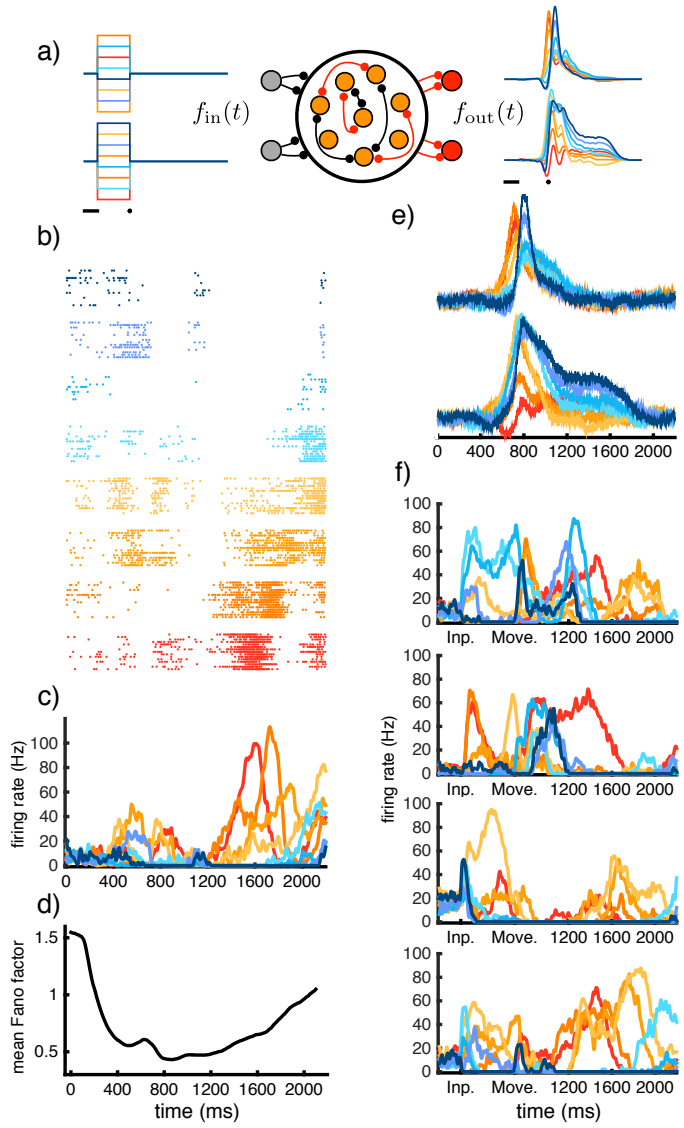

Figure 3.4: Producing EMG during reaching.

(a) Task design. A two-dimensional input (left) is applied to the network for 500 ms to specify a reach direction after which the network must produce output matching the corresponding EMG activity patterns recorded from two arm muscles (right). Each color represents the activity for a specific direction. The time bar represents 200 ms, and the dot denotes movement onset. **(b)** Raster plot of neuron 774 showing the activity of a single neuron across 15 trials (each row) for all conditions (different colors). The Fano factor for this neuron is 1.3. **(c)** Firing rate of neuron 774. Each color represents the trial-averaged firing rate for a single condition. Firing rate was computed by averaging spikes across trials and filtering the average with a 8 ms Gaussian. **(d)** The Fano factor as a function of time computed across all neurons and conditions. **(e)** $\boldsymbol{w}\boldsymbol{s}_l$ for both outputs and all conditions (different colors) on a single trial. **(f)** Firing rates for four network neurons (1,6,11,1629) displaying a variety of response profiles. 'Inp.' indicates the time of input and 'Move.' indicates the time of movement. $N = 2000$, $n = 800$.

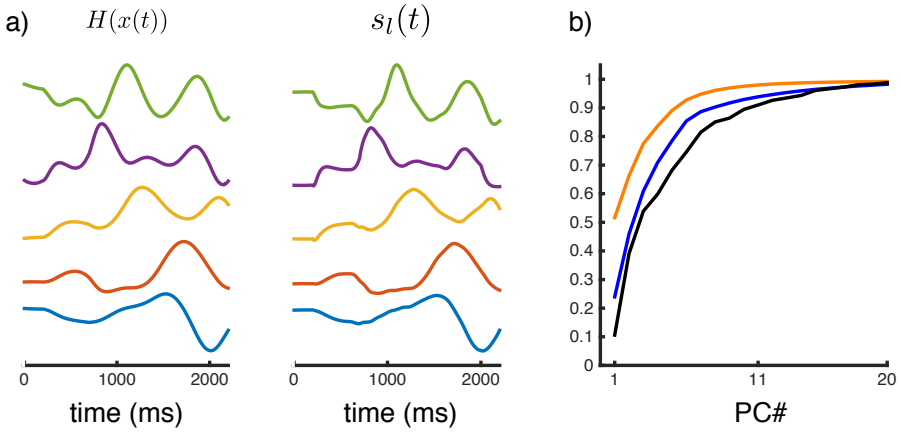

Figure 3.5: Population level analyses of EMG task activity.

(a) Canonical temporal PCs 1-5 for the firing-rate model (left) and the spiking model (right). **(b)** Fraction of the total variance captured in successive temporal PCs of $\boldsymbol{s}$ and $\boldsymbol{f}$ of the spiking network (orange), $H(\boldsymbol{x})$ of the rate network (blue) and fraction of the EMG variance accounted for when regressing against increasing numbers of temporal PCs of the spiking network (black).

the task.

To examine the population dynamics of this model and to determine how these dynamics related to the population dynamics of the continuous variable model used to train it, we performed PCA on the output of both networks (the synaptic currents $\boldsymbol{s}$ and $\boldsymbol{f}$ of the spiking network and $H(\boldsymbol{x})$ from the continuous variable network). We constructed a $T \times 2NC$ data matrix for the spiking network and a $T \times \tilde{N}C$ data matrix for the continuous-variable network, where T is the number of times sampled during a trial (2200), N and $\tilde{N}$ are the number of neurons in each network (2000 and 1000 respectively), and C is the number of reach conditions (8). We computed the eigenvectors of the $T \times T$ covariance matrix obtained from these “data", generating temporal PCs. Each temporal PC represents a function of time that is strongly represented across each population and across all reach conditions, albeit, in different ways across each population. We then performed Canonical Correlation Analysis on the first 10 temporal PCs of both models, to find the temporal signals that were common to both models. We refer to these temporal functions

as canonical temporal PCs (Figure 3.5a). Two important features emerge from this analysis.

PCA applied to $\boldsymbol{s}$ and $\boldsymbol{f}$ of the spiking network and to $H(\boldsymbol{x})$ of the continuous-variable network yield very similar results, at least for the dominant PCs. For example, the leading 10 temporal PCs account for approximately 90% of the total variance for both networks, and the median of the principle angles between the subspaces defined by these two sets of 10 vectors is 5 degrees. Furthermore, this small number of functions are sufficient to construct the network output. This can be verified by reconstructing the network output using increasing numbers of temporal PCs and calculating the fraction of the output variance captured (Figure 3.5b, black). This indicates that the temporal signals that dominate the dynamics and output of these two different types of networks are extremely similar and are the signals necessary for generating the network output.

Learning constrained connectivities

The examples we have presented up to now involve a fully connected $\boldsymbol{J}^{\mathrm{f}}$ and $\boldsymbol{J}^{\mathrm{s}}$ matrix with no sign constraints. In other words, no elements were constrained to be zero, and the training procedure could make the sign of any element either positive or negative. Biological networks tend to be sparse (many elements of $\boldsymbol{J}^{\mathrm{f}}$ and $\boldsymbol{J}^{\mathrm{s}}$ are fixed at zero) and obey Dale's law, corresponding to excitatory and inhibitory cell types. This implies that the columns of $\boldsymbol{J}^{\mathrm{f}}$ and $\boldsymbol{J}^{\mathrm{s}}$ should be labelled either excitatory or inhibitory and constrained to have the appropriate sign ($+$ for excitatory and $-$ for inhibitory). Here we outline a procedure for training a network to solve a task while abiding by these constraints.

In the previous examples, we used a recursive least squares (RLS) procedure to compute $\boldsymbol{J}_l$ because of stability issues that we now explain. Satisfying Equation 3.4 accurately assures that $\boldsymbol{s}_l$

can be generated self-consistently, but it does not guarantee that the resulting network state will be stable, and if it is unstable the least-squares solution is useless. We find that the use of RLS resolves this problem. As explained previously [Sussillo and Abbott, 2009], RLS provides stability because the fluctuations produced by the network when it is doing the task are sampled during the recursive procedure, allowing adjustments to be made that quench instabilities. However, when sparseness and especially sign constraints are imposed, use of RLS becomes impractical. Instead we must resort to a batch least-squares (BLS) algorithm.

The BLS algorithm computes $\boldsymbol{J}_l$ on the basis of samples of $\boldsymbol{s}_l$ that must be provided. To assure a stable solution, these samples should not only characterize the activity of a network doing the task, they should include the typical fluctuations that arise during network operation. To obtain such samples, we perform the network training in two steps. First, we train a fully connected and sign-unconstrained spiking network using the RLS procedure, just as in the previous examples. We sample $\boldsymbol{s}_l$ from this network during the training process, and use these samples as new auxiliary target functions, and solve the BLS problem while enforcing the desired constraints.

Applying this two-step procedure, we can construct networks that perform the oscillation task of Figure 3.2 with 50% sparseness, either without (Figure 3.6a-c) or with (Figure 3.6g-i) a Dale's law constraint. The normalized post-training error for both networks on this task is 5%, although to obtain this level we had to allow an average firing rate of 22 Hz. In addition, $n = N$ in this case (all synapses were trained) and network learning was performed on all slow synapses ($n = 1000$), because for the number of neurons we used (1000), these networks are somewhat fragile to chaotic fluctuations. This fragility could be reduced by using more neurons, but even with BLS the computations, especially for the sign-constrained case, are quite lengthy.

The two-step training procedure is needed to sample network fluctuations properly. Could we

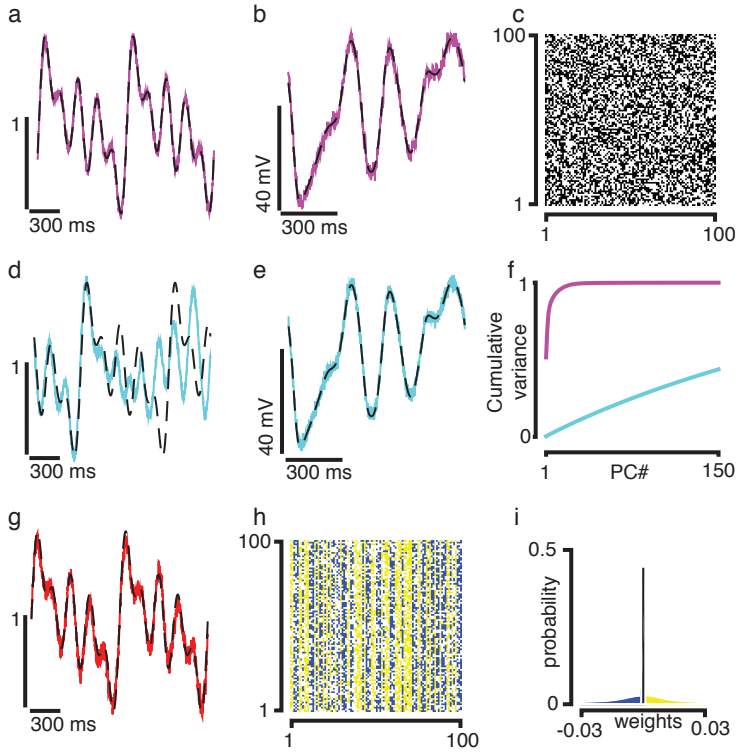

Figure 3.6: Performing the oscillation task with a constrained J

(a) f_{out} (dashed black) and $\boldsymbol{w}\boldsymbol{s}_l$ (magenta) from a network with 50% sparse connectivity. **(b)** $\boldsymbol{f}_{\text{J}}(t)$ (dashed black) and $\boldsymbol{J}_l\boldsymbol{s}_l$ (magenta) for one neuron during training. Residual fluctuations can be seen, critical for stability after learning. **(c)** Entries of $\boldsymbol{J}^{\text{s}}$ for 100 neurons. **(d)** Same as (a) except residuals from RLS solutions were shuffled before BLS was performed. **(e)** Same as (b), except showing shuffled residuals. **(f)** Cumulative variance of successive PCs of the spatial covariance matrix of the RLS residuals (magenta) and the shuffled residuals (cyan). **(g)** f_{out} (dashed black) and $\boldsymbol{w}\boldsymbol{s}_l$ (red) from a network with 50% sparseness satisfying Dale's Law (with 50% excitatory and 50% inhibitory neurons). **(h)** Entries of $\boldsymbol{J}^{\text{s}}$ for (g) for 100 neurons. **(i)** Histogram of the entries of $\boldsymbol{J}^{\text{s}}$ for (g). Network parameters: $N = 1000$, $\mu_{\text{f}} = g_{\text{f}} = g_{\text{s}} = 0$ mV, $g = 15$ mV, $n = N$.

have simply added white noise to samples of $\boldsymbol{s}_l$ that did not contain the actual fluctuations produced by an operating network [Eliasmith, 2005; Jaeger and Haas, 2004]? PCA on the covariance matrix of the network fluctuations obtained during RLS training shows that most of their variance is captured by a small number of PCs (Figure 3.6f, magenta), indicating significant spatial correlations. The temporal autocorrelation function for residual errors also showed significant correlation (not shown). To understand the role of these correlations, we created a dataset of

s_l with the actual network fluctuations replaced by shuffled fluctuations. Although the shuffled synaptic input (Figure 3.6e) is very similar to the un-shuffled input (Figure 3.6b), use of the shuffled data set resulted in poor performance of the trained network (Figure 3.6d). This is because the shuffled data fail to capture the correlations present in the actual network fluctuations (Figure 3.6f, cyan). These results affirm the conclusion that the RLS algorithm is effective for sampling network instabilities, and that the fluctuations obtained in this way can be used effectively to obtain constrained BLS values for $\boldsymbol{J}^{\mathrm{f}}$ and $\boldsymbol{J}^{\mathrm{s}}$.

Discussion

We have developed a framework for constructing recurrent spiking neural networks that perform the types of tasks solved by biological neural circuits and that can be made compatible with biophysical constraints on connectivity. In this approach, the first step in producing a spiking system that implements a task is to identify a continuous-variable dynamical system that can, at least in principle, perform the task. Previous approaches to building models that perform tasks also resorted to identifying continuous analog systems. A key distinction, however, is that by exploiting the rich dynamics of externally driven, randomly connected, continuous-variable models, we can apply our approach to cases where a dynamic description of the task is not readily apparent. In general, any continuous-variable network that can implement a task should generate useful auxiliary targets for training a spiking model. An intriguing future direction would be to use continuous-variable networks trained with back-propagation [Martens and Sutskever, 2011; Sussillo, 2014; Sussillo et al., 2015] for this purpose. Another recent proposal for training spiking networks also makes use of continuous-variable network dynamics, but in this interesting approach, a spiking network is constructed to duplicate the dynamics of a continuous-variable

model, and then it is trained essentially as if it were the continuous model [Thalmeier et al., 2015].

Our work involves a more complex map from the continuous variables of a “rate" model to the action potentials of a spiking model. The simplest map of this type assigns a population of spiking neurons to each unit of the continuous-variable model such that their collective activity represents a continuous “firing-rate". If we had followed this path, our spiking networks would have likely involved hundreds of thousands to millions of model neurons. In our approach, only a few times as many spiking neurons as continuous-variable units are needed. Performing PCA on the activity of a continuous-variable network reveals that relatively small number of PCs capture a large fraction of the variance [Rajan et al., 2010; Sussillo and Abbott, 2009]. Thus, the unit activity in these networks is redundant, and matching every unit with a different population of spiking neurons is wasteful because this amounts to representing the same PCs over and over again. Our approach avoids this problem by distributing continuous signals from the entire “rate” network to overlapping pools of spiking neurons.

Our work involves a novel interpretation of continuous variable models and the outputs of their units. These models are typically considered to be approximations of spiking models [Ermentrout, 1994; Gerstner, 1995; Ostojic and Brunel, 2011; Shriki et al., 2003]. We do not interpret the “firing rates" of units in continuous-variable networks as measures of the spiking rates of any neuron or collection of neurons in our spiking networks (in fact, these “firing rates" can be negative, which is why we avoid the term firing-rate network). Instead, the continuous-variable networks are generators of the principle components upon which the dynamics of both networks are based. The key information being passed from the continuous-variable network to the spiking network is carried by the leading PCs. The continuous-variable network is used in our procedure as a way of computing the PCs relevant to a task. If these can be obtained in another

way, the spiking network could be trained directly from the PCs. One way of doing this is to extract the PCs directly from data, as has been done in other studies [Fisher et al., 2013; Rajan et al., 2016].

Finally, our approach strongly supports the use of continuous-variable models to analyze and understand neural circuits. However, it is important to appreciate that the connection between spiking and continuous-variable networks is subtle. In our procedure, the connectivity and nonlinearity in the continuous network bear no relation to the corresponding features of the spiking model, and the continuous network is not unique. Furthermore, the signals that allow a task to be performed are only apparent at the population level. Finally, because our spiking networks are not constructed by a rational design process, it may not be immediately apparent how they work. However, the underlying continuous-variable model, and especially its leading PCs, capture the essence of how the spiking network operates, and tools exist for understanding this operation in detail [Sussillo and Barak, 2013]. These models and methods should do the same for experimental data.

Acknowledgments

We are grateful to Antonio Lara for providing the EMG data. Research supported by NIH grant MH093338 and by the Simons Collaboration for the Global Brain, the Gatsby Charitable Foundation, the Swartz Foundation, the Mathers Foundation and the Kavli Institute for Brain Science at Columbia University. B.D. was supported by a National Science Foundation Graduate Research Fellowship.

Chapter 4

Full-FORCE Learning in Continuous-Variable Networks [1]

Abstract

Trained recurrent neural networks (RNNs) are powerful tools for modeling dynamic neural computation [Sussillo, 2014]. We present a least-squares based method for training the full connectivity matrix of RNNs that allows small networks of continuous-variable "firing-rate" units to reliably perform tasks that evolve over behaviorally relevant timescales (on the order of seconds), making them relevant to the study of neural dynamics related to behavior. Since our method does not require the computation of gradients, it converges faster than gradient-based methods while performing tasks with similar numbers of units at similar performance levels. Additionally, since our method adjusts the full recurrent connectivity, it can perform tasks with fewer neurons when compared to traditional least-squares approaches to training RNNs.

---

[1] Portions of this chapter were presented as *Full-rank regularized learning in recurrently connected firing rate networks*, COSYNE (2016), co-authored by Brian DePasquale[†], Christopher J. Cueva[†], Raoul-Martin Memmesheimer[†,‡], L.F. Abbott[†] and G. Sean Escola[†,§]

[†] Department of Neuroscience, Columbia University College of Physicians and Surgeons, New York, NY USA. [‡] Department of Neuroinformatics, Donders Institute for Brain Cognition and Behavior, Radboud University, the Netherlands. [§] Department of Psychiatry, Columbia University College of Physicians and Surgeons, New York, NY USA.

Introduction

Normally, the problem of training recurrent neural networks is solved by back-propagating gradients in time, and methods of this type have been quite successful [Martens and Sutskever, 2011; Pascanu et al., 2013]. We seek to develop an alternative training method that does not require computing gradients since these methods, although effective, are computationally costly and converge slowly. Alternative approaches exist (typically referred to as "reservoir computing" or FORCE learning methods [Jaeger and Haas, 2004; Sussillo and Abbott, 2009]). While these methods are computationally efficient and converge quickly (since they only rely on solving a least-squares problem), they do not take advantage of the full connectivity of the network because they only make a low-rank modification to it. As a result, they have been shown to require orders of magnitude more neurons to perform tasks at the performance level of networks trained with gradient-based methods [Triefenbach et al., 2010].

Our method offers a compromise between speed and network size. It is an extension of the same basic principles present in FORCE learning, and thus it converges quickly and training is computationally cheap. Additionally, because of key modifications to the assumptions of FORCE learning, it can learn complex tasks with fewer neurons than traditional methods, and it approaches the performance level of gradient-based methods.

Traditional FORCE learning solves for a recurrent connectivity matrix that is a low-rank perturbation to an initially random recurrent connectivity. Our method solves a similar learning problem while learning the full, recurrent connectivity without a rank restriction. It is for this reason that we call our method Full-FORCE learning. We compare the results of Full-FORCE learning to both gradient-based methods and traditional FORCE learning, highlighting its in-

creased performance when compared to FORCE learning and comparable performance when compared to gradient learning.

Network model and learning

The focus of this work is a gradient-free method—called Full-FORCE learning—for constructing a recurrently connected network that can generate a specified output. The network model we study is a recurrently connected network of N continuous-variable model neurons. These neurons receive an input $f_{\text{in}}(t)$ and, following learning, should produce a desired output function $f_{\text{out}}(t)$. The specifics of $f_{\text{in}}(t)$ and $f_{\text{out}}(t)$ define the nature of the task we would like the network to perform.

The activity state of our model is described by a N-component vector $\boldsymbol{x}$ that satisfies the equation

$$\tau \dot{\boldsymbol{x}} = -\boldsymbol{x} + \boldsymbol{J} H(\boldsymbol{x}) + \boldsymbol{u}_{\text{in}} f_{\text{in}} \tag{4.1}$$

where τ sets the time-scale of the network dynamics, H is a nonlinear function (we use $H(\cdot) = \tanh(\cdot)$), $\boldsymbol{J}$ is a $N \times N$ matrix that will be learned, and $\boldsymbol{u}_{\text{in}}$ is a matrix of dimension $N \times N_{\text{in}}$. The elements of $\boldsymbol{u}_{\text{in}}$ are chosen independently from a uniform distribution between -1 and 1. The network output is a linear projection of the network activity via a $N_{\text{out}} \times N$ matrix $\boldsymbol{w}$ that is also learned. For the examples we show $\tau = 10$ ms and $N_{\text{in}} = N_{\text{out}} = 1$.

The goal of network training is to modify the entries of $\boldsymbol{J}$ and $\boldsymbol{w}$ so that the network output matches the desired target function:

$$\boldsymbol{w}H(\boldsymbol{x}) \approx f_{\text{out}} \tag{4.2}$$

Given an appropriate set of functions $H(\boldsymbol{x})$ this task is trivial, but identifying a matrix $\boldsymbol{J}$ that generates these functions is far trickier and is the primary challenge when constructing recurrent network models.

FORCE learning

Since our method is an extension of the FORCE learning method, it is worthwhile to explain how FORCE learning operates. There are three key components to the success of FORCE learning and reservoir methods: 1) a proper initialization of $\boldsymbol{J}$; 2) the inclusion of a feedback loop with weights $\boldsymbol{u}_{\text{out}}$; and 3) the use of the recursive least-squares (RLS) algorithm to solve Equation 4.2. It is the interaction of the feedback signal with the dynamics generated from the initial choice of $\boldsymbol{J}$ that makes this approach viable for learning. The use of RLS to solve Equation 4.2 leads to stable dynamics after learning [Sussillo and Abbott, 2009]. A FORCE learning network in this standard "feedback" form is depicted in (Figure 4.1(a)).

FORCE learning initializes $\boldsymbol{J}$ as a Gaussian random matrix of zero mean and variance g^2/N. The proper choice of g is important for successful learning. We refer to the initial value of $\boldsymbol{J}$ as $\boldsymbol{J}^{\text{D}}$. The feedback loop, $\boldsymbol{u}_{\text{out}}$, is a matrix of dimension $N \times N_{\text{out}}$ whose elements are drawn independently from a uniform distribution between -1 and 1. Provided that $\boldsymbol{J}^{\text{D}}$ and $\boldsymbol{u}_{\text{out}}$ are scaled appropriately and given a sufficient number of neurons, FORCE learning can be used to solve a variety of tasks.

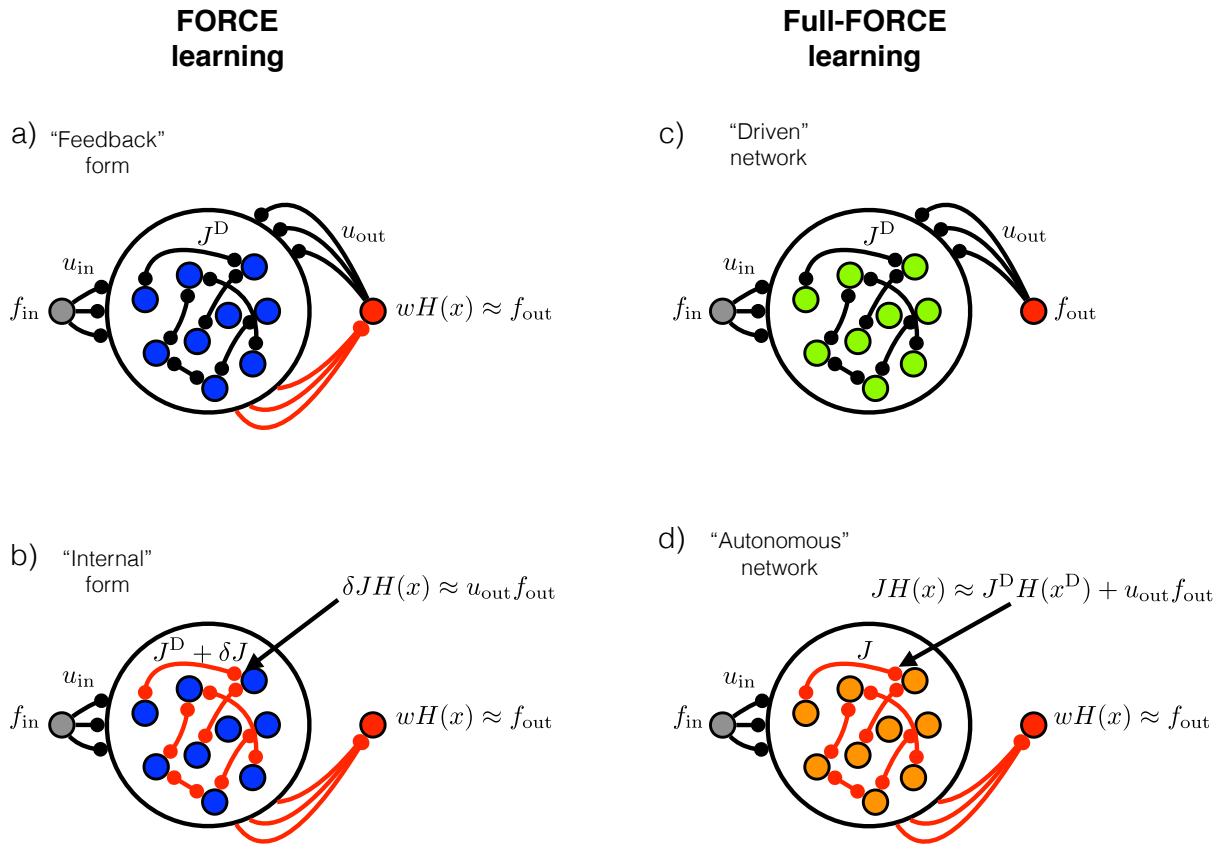

Figure 4.1: Network structure and the learning problem.

A recurrently connected network of N continuous-variable units receives an external input $f_{\mathrm{in}}(t)$. (**a**) FORCE learning with standard "feedback loop" presentation. Training will modify the output connections $\boldsymbol{w}$ so that the output of the network $\boldsymbol{w}H(\boldsymbol{x})$ matches a desired target function $f_{\mathrm{out}}(t)$. This output is then fed back into the network via the feedback loop $\boldsymbol{u}_{\mathrm{out}}$. (**b**) The recurrent connections $\delta \boldsymbol{J}$ are trained so the feedback input into each neuron matches a random projection of the output target function. The total recurrent connectivity $\boldsymbol{J}$ is the sum of its initial value $\boldsymbol{J}^{\mathrm{D}}$ and $\delta \boldsymbol{J}$. (**c**) The driven network used in Full-FORCE learning. The desired output function $f_{\mathrm{out}}(t)$ is applied as input to the initially randomly connected network. (**d**) The recurrent connections $\boldsymbol{J}$ are trained so the input into each neuron matches the input into each neuron of the driven network.

Solving Equation 4.2 and feeding this solution back into the network, yields the following augmented dynamical equations:

$$\tau \dot{\boldsymbol{x}} = -\boldsymbol{x} + \boldsymbol{J}^{\mathrm{D}} H(\boldsymbol{x}) + \boldsymbol{u}_{\mathrm{out}} \boldsymbol{w} H(\boldsymbol{x}) + \boldsymbol{u}_{\mathrm{in}} f_{\mathrm{in}} \tag{4.3}$$

Performing a bit of algebra on Equation 4.3, we can see that following learning, the full recurrent connectivity has been modified from its initial value by a low-rank (in this case, rank-1) matrix,

$$\boldsymbol{J} = \boldsymbol{J}^{\mathrm{D}} + \delta \boldsymbol{J}. \tag{4.4}$$

where $\delta \boldsymbol{J} = \boldsymbol{u}_{\mathrm{out}} \boldsymbol{w}$. In general, for a K-dimensional output, the modification will be of rank K. Viewed in this way, FORCE learning actually solves a second, equivalent least squares problem that determines how the feedback loop will change the recurrent connectivity:

$$\delta \boldsymbol{J} H(\boldsymbol{x}) \approx \boldsymbol{u}_{\mathrm{out}} f_{\mathrm{out}} \tag{4.5}$$

This alternate "internal" form of FORCE learning is illustrated in (Figure 4.1(b)). To generate the network output, Equation 4.2 can be solved with batch least-squares (BLS) after Equation 4.5 has been solved (or with RLS while it is being solved) since they are identical problems.

Full-FORCE learning

Full-FORCE learning approaches the problem in a similar way as FORCE learning, but with a modification to the learned form of $\boldsymbol{J}$. If we examine Equation 4.3 carefully, we notice that

if Equation 4.2 is successfully solved, then the recurrent and feedback input into each network unit (excluding the input due to f_{in}) is approximately $\boldsymbol{J}^{\text{D}}H(\boldsymbol{x}) + \boldsymbol{u}_{\text{out}}f_{\text{out}}$. Stated another way: if FORCE learning is successful, the dynamics of the recurrent units will be identical to the dynamics if the network was simply driven by the desired target function.

Since the goal of learning is to modify the recurrent connectivity such that the network dynamics after learning are as close as possible to what they would be if the system was driven with the target itself, Full-FORCE learning obtains the recurrent dynamics of a network driven by the target function (Figure 4.1(c)) and trains a second, recurrent network to generate those trajectories autonomously (Figure 4.1(d)).

The target function f_{out} is applied as input to a network initially connected with connectivity matrix $\boldsymbol{J}^{\text{D}}$. The recurrent connections of the autonomous network $\boldsymbol{J}$ are trained so the input into each neuron matches the input into each neuron of the driven network, which takes the form of the following least-squares problem:

$$\boldsymbol{J}H(\boldsymbol{x}) \approx \boldsymbol{J}^{\text{D}}H(\boldsymbol{x}^{\text{D}}) + \boldsymbol{u}_{\text{out}}f_{\text{out}} \tag{4.6}$$

where $\boldsymbol{x}^{\text{D}}$ is the network dynamics for a network externally driven by the output function f_{out}. For Full-FORCE learning, $\boldsymbol{J} = 0$ before learning.

In choosing to solve Equation 4.6 for the recurrent connectivity instead of solving Equation 4.5 we are removing the restriction of FORCE learning that $\boldsymbol{J}$ take the form of a rank-1 perturbation to $\boldsymbol{J}^{\text{D}}$; instead the full recurrent connectivity is learned. Once Equation 4.6 has been solved using RLS, then Equation 4.2 can be solved as in standard FORCE learning to generate the network

output.

Input driven periodic task

We now show that the modified least-squares problem of Full-FORCE leads to improved performance over traditional FORCE learning. We illustrate this with a simple periodic task where f_{out} takes the form of a frequency modulated oscillator of period 2 seconds. In this task, $f_{\text{out}} = \sin(\omega(t)t)$ where ω increases linearly from $1/2\pi$ to $3/2\pi$ for the first half of the oscillator's period, and then linearly decreases in absolute value from $-3/2\pi$ to $-1/2\pi$ during the second half. f_{in} is a 50 ms pulse of amplitude 1.0 at the beginning of each period of the oscillation; this was done to address phase drift that tends to accumulate in these models when learning periodic tasks. For this task we set $g = 1.5$ and all other parameters were set as stated above. Results from these simulations are shown in Figure 4.2.

A Full-FORCE network of 300 units was able to perform this task perfectly whereas a network trained with traditional FORCE learning could not. We therefore systematically examined how learning was effected by the number of units in each model. Results from these simulations are shown in Figure 4.3(a). Full-FORCE learning could solve the task reliably using 150 units where FORCE learning required more than twice that to perform the task reliably. These simulations suggested that learning is achieved more gracefully as a function of N in Full-FORCE networks than in FORCE networks. Using FORCE learning, networks appeared to fail at learning up until a certain value of N, after which they appeared to reliably learn.

We also examined the robustness of Full-FORCE networks and FORCE networks to external noise. Gaussian white-noise of variance η^2 was applied to each network unit during learning and

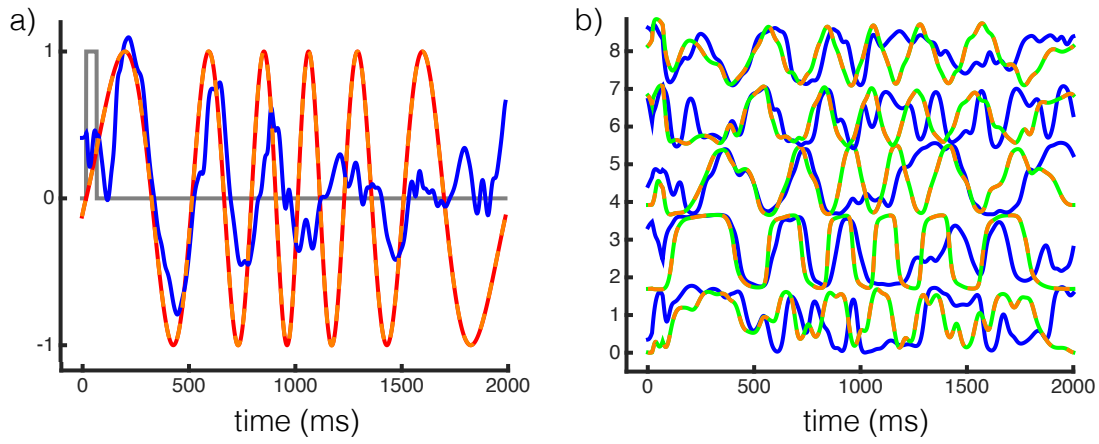

Figure 4.2: Input, output and firing-rates.

(**a**) $f_{\text{in}}(t)$ (grey), $f_{\text{out}}(t)$ (red) and $\boldsymbol{w}H(\boldsymbol{x})$ for both networks (FORCE is in blue, Full-FORCE is orange) after training, for a network of 300 neurons. The desired target function is shown in red and is completely overlapping with output of the Full-FORCE network. (**b**) $H(\boldsymbol{x})$ for both networks (FORCE in blue, Full-FORCE in orange). The green trace shows the activity of the driven network, $H(\boldsymbol{x}^{\text{D}})$.

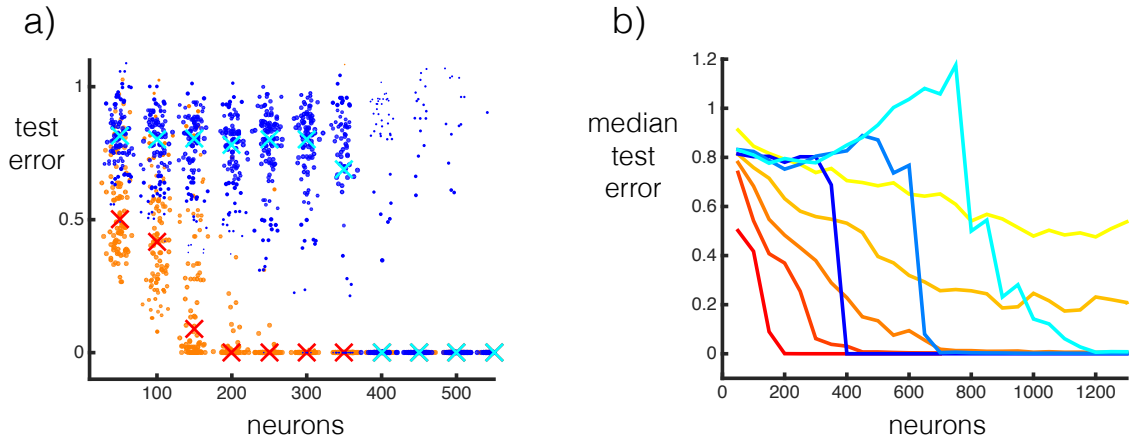

Figure 4.3: Performance of FORCE and Full-FORCE networks.

(**a**) Test error for FORCE (blue) and Full-FORCE (orange) for networks of increasing size. Each dot represents the test error for one random initialization of $\boldsymbol{J}^{\text{D}}$, the 'x' is the median value across all simulation for a given value of N and the size of each dot is proportional to the distance of that point from the median. (**b**) Median test error for both training procedures averaged across 100 random initializations of $\boldsymbol{J}^{\text{D}}$ for various noise levels. The red color class (ranging from red to yellow) show results for Full-FORCE networks and the blue color class (ranging from blue to cyan) show results for FORCE networks. Noise levels for Full-FORCE networks are $\eta = 0$, 0.0125, 0.025, 0.05 and 0.1. Noise levels for FORCE networks are $\eta = 0$, 0.00625 and 0.0125.

during testing. We systematically varied the noise and examined the performance of the two methods as a function of the noise level and number of units. Performance for both networks for various levels of noise is shown in Figure 4.3(b). Here again, the Full-FORCE network outperformed the FORCE network. FORCE learning required 3 times as many neurons to perform this task when compared to Full-FORCE learning for the weakest noise level. Again, the Full-FORCE network appeared to learn the task more gracefully as a function of N, whereas FORCE networks abruptly jump from failing to solving the task at a specific value of N. This robustness to noise is qualitatively similar to the results observed by other research groups using related methods to train the recurrent connectivity [Laje and Buonomano, 2013].

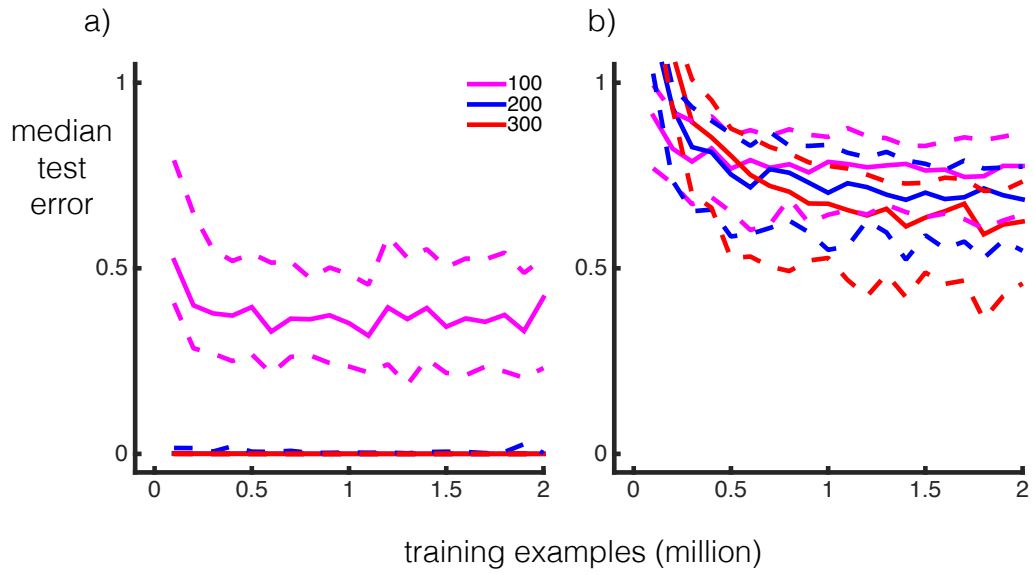

Figure 4.4: Test error as a function of training time.

(**a**) Median and quartile test error for a Full-FORCE network across 100 random initializations of $\boldsymbol{J}^{\mathrm{D}}$ for various training intervals and network sizes when trained on the oscillation task of Figure 4.2. (**b**) Same as (**a**) for a FORCE network.

Perhaps FORCE networks would eventually perform the task at a similar performance level if they were trained for longer? To check this we examined the relationship between number of

training examples and test error, which is plotted for both networks in Figure 4.4. In both cases, training beyond a certain number of training examples did not appear to significantly improve test performance. Stated another way—for either FORCE or Full-FORCE learning—a network of a given size is either sufficient or insufficient to solve the task. These results are interesting when compared to gradient-based approaches that tend to systematically improve with training time. These results suggest that for performing least-squares learning in recurrent networks, the form of targets used for learning are more important than the length of training. We will address these ideas further below.

Singular values of J

To understand the source of Full-FORCE learning's improved performance compared to FORCE learning, we examined the singular value distribution of $\boldsymbol{J}$ after learning (Figure 4.5).

As expected, the singular value distribution for a FORCE network contained one large singular value while the remaining singular values were distributed according to $\boldsymbol{J}^{\mathrm{D}}$. The singular value distribution of a Full-FORCE network shows an augmentation of approximately 20 of the largest singular values and a suppression of the remaining modes. Thus, learning the full connectivity matrix with Full-FORCE learning removes superfluous and potentially harmful modes of the initial connectivity. This likely explains its increased noise-robustness and ability to learn with fewer neurons.

To understand how this shrinking of unused modes of $\boldsymbol{J}^{\mathrm{D}}$ arises, it is instructive to write the closed form solution to the least squares problem (even though we in fact solve this problem recursively). To solve the standard FORCE learning problem of Equation 4.5 we must find $\delta\boldsymbol{J}$

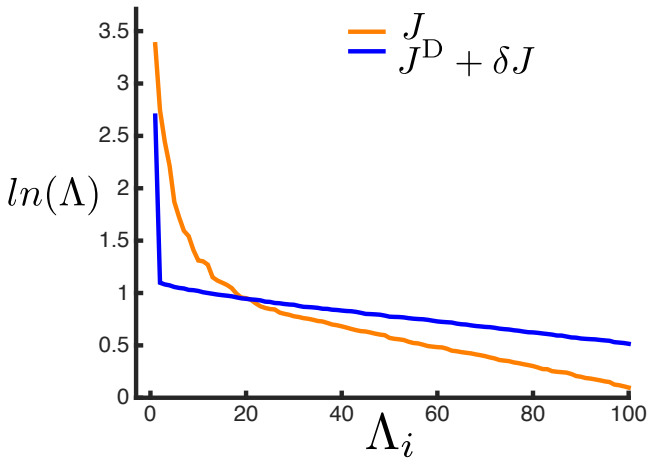

Figure 4.5: Singular values of learned connectivity.

Singular values of the learned connectivity for FORCE method (blue) and Full-FORCE method (orange). The Full-FORCE method augments additional singular values and suppress unwanted singular values, while the FORCE method only significantly modifies one singular value.

that minimizes the following error function:

$$E = \frac{1}{2}\left\langle ||\delta \boldsymbol{J} H(\boldsymbol{x}) - \boldsymbol{u}_{\text{out}} f_{\text{out}}||^2 \right\rangle + \frac{1}{2}\lambda ||\delta \boldsymbol{J}||^2 \tag{4.7}$$

We include a regularization term weighted by λ that appears in the RLS algorithm as an initialization on the inverse covariance matrix and $\langle \cdot \rangle$ is an average over time. The closed form solution to this least-squares problem takes the following form:

$$\delta \boldsymbol{J} = \left(\langle HH^T \rangle + \lambda I \right)^{-1} \langle H(\boldsymbol{u}_{\text{out}} f_{\text{out}})^T \rangle \tag{4.8}$$

H is shorthand for $H(\boldsymbol{x})$. From Equation 4.8, we can see that $\delta \boldsymbol{J}$ when learned with FORCE is a rank-1 matrix.

Full-FORCE learning solves Equation 4.6 by finding $\boldsymbol{J}$ that minimizes a different error function:

$$E = \frac{1}{2}\left\langle ||\boldsymbol{J}H(\boldsymbol{x}) - \boldsymbol{J}^{\mathrm{D}}H(\boldsymbol{x}^{\mathrm{D}}) - \boldsymbol{u}_{\mathrm{out}}f_{\mathrm{out}}||^2 \right\rangle + \frac{1}{2}\lambda||\boldsymbol{J}||^2 \tag{4.9}$$

The closed form solution to this least-squares problem takes the following form:

$$\boldsymbol{J} = \left(\langle HH^T\rangle + \lambda I\right)^{-1}\langle H(\boldsymbol{u}_{\mathrm{out}}f_{\mathrm{out}})^T\rangle + \left(\langle HH^T\rangle + \lambda I\right)^{-1}\langle H(J^{\mathrm{D}}H^{\mathrm{D}})^T\rangle \tag{4.10}$$

where the first term on the right-hand side is $\delta\boldsymbol{J}$ from standard FORCE learning. We call the second term on the right-hand-side $\delta\boldsymbol{J}^{\mathrm{FR}}$, since this matrix will, in general, be full-rank. $\delta\boldsymbol{J}^{\mathrm{FR}}$ likely accounts for the singular value results observed for Full-FORCE networks: large modes of $H(H^{\mathrm{D}})^T$, corresponding to correlated network dynamics, amplify useful modes of $\boldsymbol{J}^{\mathrm{D}}$ and suppresses superfluous modes.

Finally, we note that the paradoxical *decrease* in performance for increasing N when noise was applied to FORCE networks (see Figure 4.3(b), around $N = 800$ for $\eta = 0.0125$) can be explained in light of these results. The only tool FORCE has at its disposal for solving the task is to increase the magnitude of the one singular value it has control over. For N too small, it seems, FORCE can never increase the size of that singular value enough to solve the task, and the test error reflects a random output. As N increases, approaching a value where FORCE *can* solve the task, the value of its one controllably singular value has grown large but not large enough to solve the task. For these values of N, that large singular value is likely to contribute detrimentally to testing error by generating large fluctuations in, an effectively, random output.

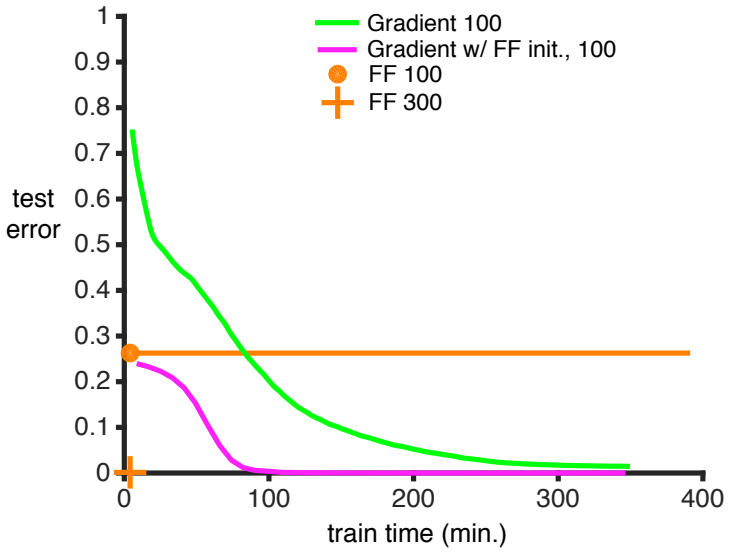

Figure 4.6: Training time for Full-FORCE and gradient-based networks.

Test error as a function of training time for Full-FORCE networks and Hessian-Free networks of different sizes. A Full-FORCE network of 300 neurons can perform this task in 6 minutes (orange cross), where a Hessian-Free network of 100 units (green line) can take over 3 hours to achieve this level of performance. A Full-FORCE network of 100 units (orange dot) cannot achieve the same performance as a Hessian-Free network of the same size, even with extensive training time (orange line). However, if a Hessian-Free network is initialized with the solution from a Full-FORCE network (magenta line) training time is dramatically reduced.

Comparing Full-FORCE networks to gradient-based networks

Since the Full-FORCE method was successful when using relatively few units, we sought to compare our results with gradient-based methods which normally do not require large networks to solve tasks. We trained recurrent networks using Hessian-Free optimization [Martens and Sutskever, 2011; Pascanu et al., 2013]. This is a second-order gradient-method that has been effectively used to solve tasks that suffer from the standard pathological learning problems associated gradient-methods.

Using these methods, networks of 100 units could be built to solve the task, a small but sig-

nificant improvement over the approximately 150-200 neurons that were required with Full-FORCE learning. However, as stated earlier, the main benefit of using least-squares approaches over gradient-based methods can be seen in Figure 4.6. Here we plot test error after training as a function of training time. While gradient-based methods can solve this task with 100 neurons, requiring over three hours of training time, Full-FORCE learning could solve this task reliably with 300 neurons in only 6 minutes. Although smaller networks of 100 units could not be successfully trained on this task using Full-FORCE, these solutions could be used as initializations for Hessian-Free learning, leading to a dramatic decrease in training time. Going forward, it will be important to compare and relate least-squares and gradient-based methods since each method has a specific speed/size trade-off.

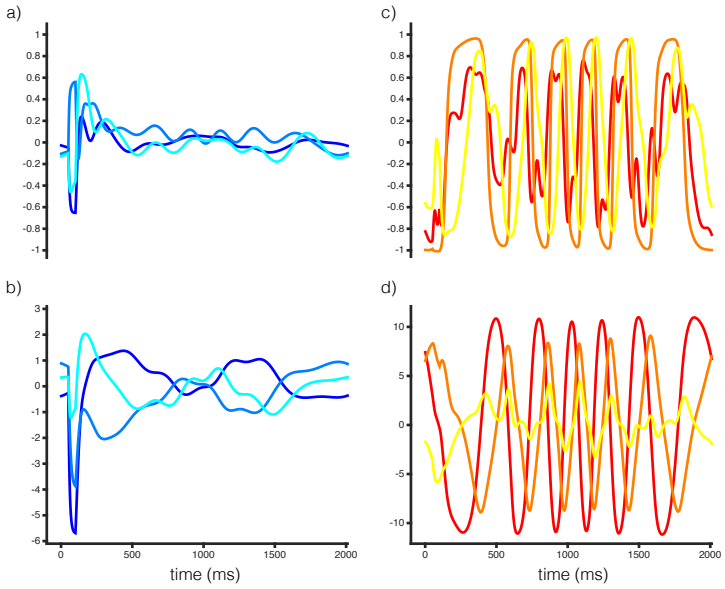

Figure 4.7: Dynamics of a Full-FORCE trained network and a Hessian-Free trained network.
(**a**) $H(\boldsymbol{x})$ for 3 neurons from a Hessian-Free network of 100 units. (**b**) $H(\boldsymbol{x})$ for 3 neurons from a Full-FORCE network of 300 units. (**c**) $H(\boldsymbol{x})$ for the Hessian-Free network projected onto the 1^{st} 3 principal components of its activity. (**d**) $H(\boldsymbol{x})$ for the Full-FORCE network projected onto the 1^{st} 3 principal components of its activity.

To better understand the types of dynamic solutions that each method constructed, we compared $H(\boldsymbol{x})$ for each network, as shown in Figure 4.7(a & c). We additionally projected these trajectories onto their principal components (PCs) to see what collective dynamics they each exhibit, as shown in Figure 4.7(b & d). The most striking difference in the activity patterns of the two networks is the overall magnitude of the fluctuations produced, as can be seen in both the PCs and the individual units. The large fluctuations in the Full-FORCE network are a result of the high g value used, a necessary evil for performing learning in these networks. We did not systematically study the success of Full-FORCE learning as a function of g. If these networks can still learn without requiring g too large, presumably the number of neurons required for learning would decrease. Additionally, the question of choosing $\boldsymbol{J}^{\mathrm{D}}$ differently, perhaps by including simple statistical features, is relevant here; if choosing $\boldsymbol{J}^{\mathrm{D}}$ in a different way leads to less complex dynamics, then Full-FORCE networks might be able to learn tasks with fewer units.

It is interesting to note that 10 PCs accounted for over 98% of the variance in both networks. The mean subspace angle between these two spaces was $\pi/3.43$ indicating that they shared some structure but were not strongly aligned.

Finally, we sought to understand the source of Full-FORCE's failure to learn for small N. Learning could have failed for two reasons: 1) because of the learning algorithm; or, 2) because the supervising signals were insufficient. To address this question we used the inputs into each unit of the trained Hessian-Free network as the supervising signal of Equation 4.6. Results from these simulations are shown in Figure 4.8. This procedure was successful, indicating that the learning algorithm itself is able to construct very small networks, comparable to the networks built with gradient methods. The key question then for Full-FORCE learning is to identify useful, parsimonious and easy-acquired training signals. As discussed above, the solution found with gradient-based methods had significantly less erratic dynamics; understanding how to select $\boldsymbol{J}^{\mathrm{D}}$

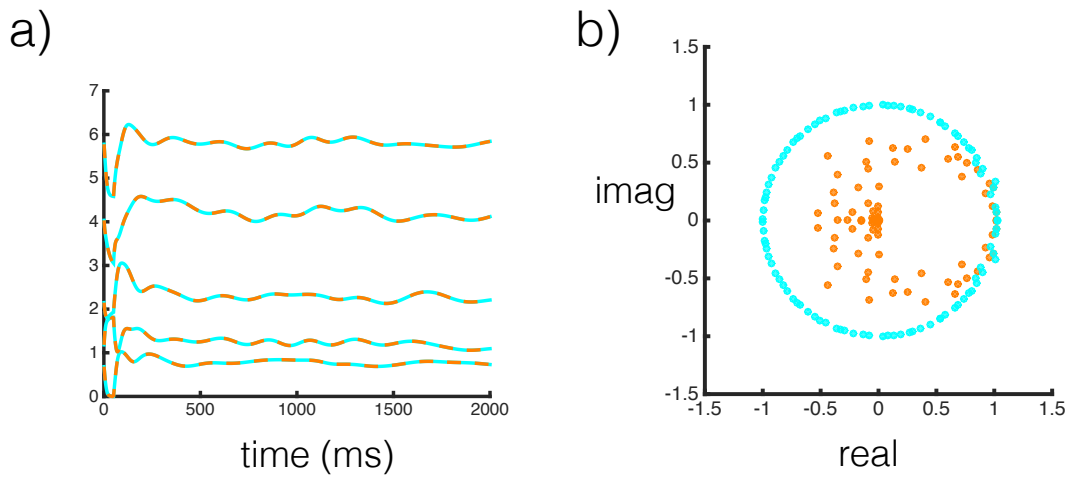

Figure 4.8: Using Full-FORCE to learn a Hessian-Free network.

(**a**) $H(\boldsymbol{x})$ for a Full-FORCE network (orange) of 100 neurons trained using the trajectories of a Hessian-Free trained network (cyan). (**b**) Eigenvalue spectrum of $\boldsymbol{J}$ for both networks after training. The Hessian-Free network was initialized with an orthogonal matrix, which was mostly left undisturbed by learning.

in Full-FORCE learning to make the dynamics more amenable to learning is a key question going forward.

Discussion

We have presented a simple, least-squares method for training recurrently connected neural networks. Our method, called Full-FORCE learning (due to its relatedness to FORCE) solves the standard least-squares problem of FORCE learning while also learning the full recurrent connectivity. Networks using this form of learning can perform better than traditional FORCE learning and almost as well as gradient-based methods. Since the learning problem is solved using least-squares, these methods are attractive for their speed and computational simplicity. We have highlighted our approach's balance between performance and training time. Future research un-

derstanding the role that fast, least-squares solutions could play in improving gradient methods by offering useful initializations is of extreme interest.

The question of regularization when using least squares methods to construct recurrent neural networks is interesting and has been pursued by other groups as well [Jaeger, 2014; Reinhart and Steil, 2011]. Future work will have to address what other forms of regularization can be included to make learning even more robust. Additionally, Full-FORCE continues to rely on RLS, as FORCE learning did, to solve for a stable recurrent system. It is important to understand exactly what noise model is being generated during RLS learning that leads to stability, and if its effect can be incorporated as a regularization term in a standard batch least-squares (BLS) framework. Moving towards a BLS framework would make learning these models even faster and permit the use of standard optimization packages.

These results support the notion that extensive training time when using least-squares methods for solving recurrent systems does not significantly impact test success, indicating that the main concern when applying these methods is the identification of excellent, self-consistent and parsimonious supervising signals for the input into each recurrent neuron. Future work will have to address what other forms of $\boldsymbol{J}^{\mathrm{D}}$ can be useful for learning; an orthogonal initialization seems fruitful based on its success with Hessian-Free methods. The appeal of using simple, random matrices for $\boldsymbol{J}^{\mathrm{D}}$ is appealing when considering biological computation but small deviations from random, based on coarse statistical properties or simple learning rules, might retain this simplicity while increasing performance.

We also highlighted the relatively simple trajectories that gradient-based methods generate and how solutions of this form seem to require fewer units. Understanding the relationship between the learned connectivity of these solutions and the dynamics they generate would be of great

use for finding generic connectivity matrices to use as initializations for least-squares learning of recurrent networks.

Acknowledgments

We thank Ben Dongsung Huh for coming up with the name Full-FORCE that integrates the original FORCE acronym and David Sussillo, Omri Barak and Vishwa Goudar for helpful suggestions regarding this work.

Chapter 5

Conclusion

Returning to the question of spikes

The results of this thesis have three important implications with regard to the future of network modeling and what it can do for neuroscience. First, if one wishes to build a spiking network, our results show that the full power of machine-learning can be brought to bear on the problem by first solving the continuous model problem with tools built for continuous variables, and then "transferring" this solution to a spiking model. Looking forward to the future, it will be interesting to see what insights can be gained when solutions found with more powerful gradient-based methods are transferred to spiking networks (Figure 5.1). Based on our findings, there is no fundamental reason to believe that a task that can be performed by a network trained with gradient-based methods (which is to say a great many tasks) cannot be performed by a spiking network.

Second, these results imply that continuous-variable models should be given the full respect they deserve within the neuroscience community. Any lingering doubt about the relevance of these models to real brains should be collectively cast away. As the full complexity of the brain descends upon us, progress depends on carefully chosen abstractions that make this problem tractable. It is the job of theoretical neuroscientists to show the rest of the community why

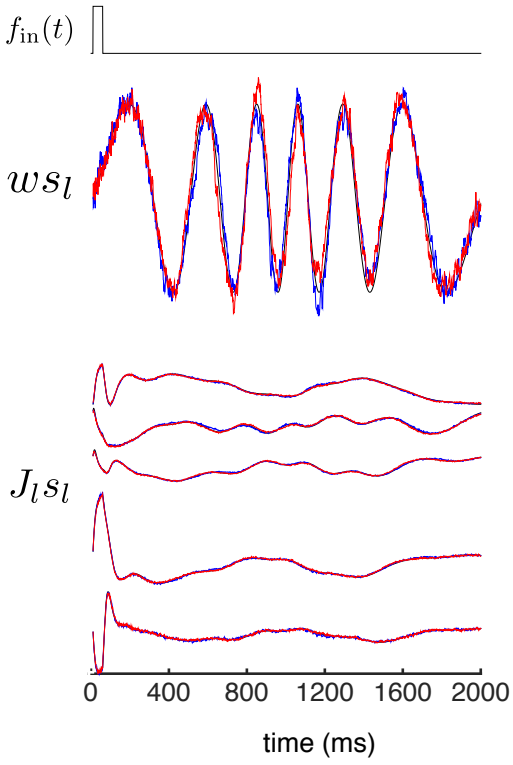

Figure 5.1: Training a spiking network with targets from a back-propagation trained network. (**top**) A network with identical parameters to the tasks of Chapter Two is trained to solve the oscillation task of Chapter Four. Two example trials are shown (red & blue, respectively). (**bottom**) $f_{\text{J}}(t)$ are derived from the Hessian-Free trained network of Chapter Four. Performance was almost identical to the networks trained in Chapter Two.

continuous-variable networks are helpful and why they are relevant. More generally, encouraging a movement towards various levels of abstraction in our models (which has never been a problem for other fields of science, and a curious sticking point for traditional neuroscientists), should be a high priority for the theoretical neuroscience community.

Third, these results question the effort to extend gradient-based learning to non-differentiable systems. This again comes down to a situation where, although it might be possible to eventually do this, we should probably first ask if it makes sense to do it. A great deal of reverence is paid to back-propagation—mostly because it works so well—but it is not a panacea for building

networks. The success of back-propagation, I believe, is mostly because the right tool (computing differential errors) was developed for the right problem (networks that are differentiable). While a hammer *can* be used to screw in a screw, it probably shouldn't. Developing different learning methods that solve tasks given the constraints of the particular system to which it is being applied is an important step in expanding the toolbox of theoretical neuroscience and machine learning.

I began this thesis by stating that at this juncture in our understanding of neural circuits, a major question is how to build network models from spiking neurons. Ironically, one of the messages arising from my work is that a model need not include spikes simply because it should or could, but rather only when it must. My work has justified the procedure of abstracting away spiking neurons in favor of simpler continuous variables, and this should continue to yield new insights into the relationship between neural activity and behavior. Nevertheless, with the tools for training spiking networks identified, it will be interesting to see what new questions can be answered about how these networks solve tasks, if these solutions are fundamentally different from the way continuous-variable models solve them and what new questions arise about the role of spikes in neural coding.

Revising the interpretation of rate models

We have offered compelling evidence supporting a reinterpretation of firing-rate models. In fact, if we attempted to draw a connection between spiking networks and firing rate networks in this work using a traditional definition, our approach would have failed without resorting to building unrealistically large spiking networks.

We believe that allowing the interpretation of rate-models to become more abstract is, in general,

a good thing. Doing so will free these models from some of the more cumbersome constraints a traditional view imposes. For example, even though traditional rate models use nonlinear functions that give rise to a negative output, doing so has always seemed questionable, since technically, a firing-rate should not be negative. If we instead allow the nonlinear function of a rate model to take whatever functional form is required to give rise to the continuous dynamics a neural circuit exhibits, we can exploit the full expressive power of these models in an unencumbered way without compromising anything. Since the output of these models can be firmly connected to the collective, dynamical patterns exhibited by ensembles of spiking neurons, the use of these models to understand network dynamics in biological neural systems can be done without ambiguity.

This view does, however, make relating some aspects of spiking models, rate models and neural data less straightforward. For example, the connectivity of a rate model only shares a passing relationship to the connectivity of the spiking model it approximates, and the specific connectivity of either model when trained to perform a task is not likely to relate in any useful way to the connectivity of a biological neural circuit performing the same task. While connectivity is undoubtedly an important aspect of how neural circuits operate, the non-uniqueness of a learned solution should encourage the neuroscience community to focus on the functional aspects of how neural circuits compute instead of the anatomical ones.

There has been an explosion in the complexity of tasks being performed during experiments. In order for neural network modeling to be of any aid in understanding how neural circuits perform these tasks, these models too will have to become more complex and therefore more abstract. The most important piece of the puzzle for using abstract neural networks as tools for understanding neural circuits is to understand how they themselves work [Sussillo and Barak, 2013].

Encouraging abstract thinking in data analysis

The modeling work presented in this thesis was developed in part to bolster support for a neural coding hypothesis, and in that way, we were quite successful. Our work places modern methods for decoding behavior using more abstract definitions of population dynamics—which are inexorably linked to assumptions about how neurons encode information—on firmer theoretical ground. Our spiking model was able to perform tasks without constraints on individual neural tuning and without requiring that groups of similarly tuned neurons exist, consistent with modern datasets that exhibit equally heterogeneous responses across neurons and in time. These results support the view that biological neural circuits are likewise not bound by these constraints and that analysis methods that unburden themselves from these constraints are sound approaches to understanding the relationship of neural activity to behavior [Churchland et al., 2007, 2012; Machens, 2010]. In light of our results, these abstract analysis methods should not be viewed with skepticism, but rather as tools of the appropriate complexity and abstraction given the complexity of the data they seek to describe. I hope that by illustrating how these encoding assumptions and decoding methods can be instantiated in an artificial neural circuit, all members of the neuroscience community will come around to their utility and soundness.

When considering how neural activity relates to behavior, there is no reason to first abstract away spikes to firing-rates and then abstract away firing-rates to lower-dimensional population dynamics; moving directly from spiking activity to population dynamics is a sound and parsimonious thing to do, and, as we have shown, constructing spiking networks that move directly between the concept of low-dimensional population dynamics and spiking activity is achievable.

We believe that our modeling efforts will support the continued development of abstract anal-

ysis methods. A problem faced by researchers developing tools for analyzing neural data is that they often do not have "ground-truth" models against which to test their methods. This problem is especially true for biologically realistic spiking networks, since, until now, methods for constructing such models were not available. We believe that the models developed here can be used in the future in this way, and therefore, continue to bolster support for these developing analysis tools.

The role of randomness

The concept of using random connectivity has come up several times in this thesis. It is interesting to point out a growing focus on understanding which aspects of neural circuit function can be achieved through random connectivity alone [Aljadeff et al., 2014; Caron et al., 2013; Ganguli and Sompolinsky, 2012; Rigotti et al., 2013; Stern et al., 2014]. In this nascent line of research, it has been found that much of the heavy lifting for which synaptic learning has been ascribed can be achieved with random connectivity or connectivity that only varies from random based on coarse statistical aspects. While some amount of learning in these networks helps [Babadi and Sompolinsky, 2014] seeking out network models with fairly random connectivity, augmented by simple learning rules, appears to be a rich research path moving forward.

Concerning the technical aspects of constructing network models, random connectivity can be harnessed as a tool for constructing network models suitable for learning, and that exhibit the types of complex activity patterns observed in neural data. Traditional network modeling approaches that "design" solutions to tasks by exploiting known dynamical phenomena (for example, by using line attractors) often fail to capture these more esoteric features of neural data [Barak et al., 2013]. Using networks that exhibit some degree of unstructured connectivity

is a fruitful path forward in our effort to construct networks that exhibit the level of complexity that biological networks exhibit.

Including additional biological realism in spiking networks

While this work represents a significant advance in terms of bringing a dose of biological realism to the field of recurrent neural network training, there is a lot more work to be done. The models we developed were stripped of most of their biologically realistic properties so the problem of training spiking neurons could be addressed and the power of synaptic modifications in these models could be understood. Moving forward, it will be interesting to see what spiking model networks can do when additional biological complexity like short-term plasticity [Buonomano, 2000; Maass et al., 2002; Mongillo et al., 2008] or complex synaptic dynamics [Benna and Fusi, 2015] is included in a model that can be trained with the tools of supervised learning. It seems logical that these additional processes will endow these networks with even more computational power.

A major shortcoming of this work is that the learning rule is not biologically plausible. I view this work as a useful stepping stone towards performing more biologically realistic learning in spiking networks. To me, the first steps forward in this effort should be to understand how the concept of a mutually dependent set of supervising functions can be connected to other learning mechanisms such as spike-timing dependent plasticity [Song et al., 2000].

Finally, training these models directly to neural data could be a fruitful path forward in understanding neural circuit dynamics [Fisher et al., 2013; Rajan et al., 2016]. To date, few neural network modeling studies have attempted to do this, and even fewer have attempted it with spik-

ing models, mostly because the data and the tools for learning did not exist. Developments in this line of research will surely lead to exciting new results.

Bibliography

L. F. Abbott. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain Research Bulletin*, 50(5/6):303–304, 1999.

L. F. Abbott and T. Kepler. Model neurons: From hodgkin-huxley to hopfield. In L. Garrido, editor, *Statistical mechanics of neural networks*, pages 5–18. Berlin: Springer-Verlag, 1990.

L. F. Abbott, B. DePasquale, and R.-M. Memmesheimer. Building functional networks of spiking model neurons. *Nat. Neurosci.*, 19:350–355, 2016.

J. Aljadeff, M. Stern, and T. O. Sharpee. Transition to chaos in random networks with cell-type-specific connectivity. *arXiv*, 2014. arXiv:1407.2297.

D. Amit and N. Brunel. Model of global spontaneous activity and local structured activity during delay periods in the cerebral cortex. *Cerebral Cortex*, 7:237–252, 1997.

D. J. Amit and M. Tsodyks. Quantitative study of attractor neural network retrieving at low spikes rates: I. substrate–spikes, rates and neuronal gain. *Network*, 2:259–273, 1991a.

D. J. Amit and M. Tsodyks. Quantitative study of attractor neural networks retrieving at low spike rates II: Low rate retrieval in symmetric networks". network. *Network*, 2:275–294, 1991b.

A. Arieli, A. Sterkin, A. Grinvald, and A. Aertsen. Dynamics of ongoing activity: Explanation of the large variability in evoked cortical responses. *Science*, 273:1868–1871, 1996.

B. B. Averbeck, P. E. Latham, and A. Pouget. Neural correlation, population coding and computation. *Nature Reviews Neurosci.*, 7:358–366, 2006.

B. Babadi and H. Sompolinsky. Spareness and expansion in sensory representations. *Neuron*, 83: 1213–1226, 2014.

O. Barak, D. Sussillo, R. Romo, M. Tsodyks, and L. F. Abbott. From fixed points to chaos: Three models of delayed discrimination. *Prog. in Neurobio.*, 103:214–222, 2013.

Y. Bengio. How auto-encoders could provide credit assignment in deep networks via target propagation. arXiv:1407.7906, 2014.

M. K. Benna and S. Fusi. Computational principles of biological memory. *arXiv*, 2015. arXiv:1507.07580v1.

M. Boerlin and S. Denève. Spike-based population coding and working memory. *PLoS Comput. Biol.*, 7:e1001080, 2011.

M. Boerlin, C. K. Machens, and S. Denève. Predictive coding of dynamical variables in balanced spiking networks. *PLoS Comput. Biol.*, 9:e1003258, 2013.

S. M. Bohte, J. N. Kok, and H. L. Poutré. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48:17–37, 2002.

R. Bourdoukan and S. Denève. Enforcing balance allows local supervised learning in spiking recurrent networks. *Advances in Neural Information Processing Systems*, 28:982–990, 2015.

R. Bourdoukan, D. G. Barrett, C. K. Machens, and S. Denève. Learning optimal spike-based representations. *Advances in Neural Information Processing Systems*, 25:2294–2302, 2012.

T. Branco and M. Häusser. Synaptic integration gradients in single cortical pyramidal cell dendrites. *Neuron*, 69(5):885–892, Mar 2011.

J. Brea, W. Senn, and J.-P. Pfister. Matching recall and storage in sequence learning with spiking neural networks. *J Neurosci*, 33(23):9565–9575, Jun 2013.

R. Brette. Philosophy of the spike: Rate-based vs. spike-based theories of the brain. *Front. in Sys. Neurosci.*, 9(151), 2015.

K. H. Britten, S. M. N., W. T. Newsome, and A. J. Movsohn. The analysis of visual motion: a comparison of neuronal and psychophysical performance. *Journal of Neurosci.*, 12(12):4745–4765, 1992.

N. Brunel. Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. *J. Comput. Neurosci.*, 8:183–208, 2000.

N. Brunel and P. E. Latham. Firing rate of the noisy quadratic integrate-and-fire neuron. *Neural Computation*, 15:2281–2306, 2003.

N. Brunel and M. C. W. van Rossum. Lapicque's 1907 paper: from frogs to integrate-and-fire. *Bio. Cybern.*, 97:337–339, 2007.

D. V. Buonomano. Decoding temporal information: A model based on short-term synaptic plasticity. *The Journal of Neuroscience*, 20(3):129–1141, 2000.

D. V. Buonomano and M. M. Merzenich. Temporal information transformed into a spatial code by a neural network with realistic properties. *Science*, 267:1028–1030, 1995.

Y. Burak and I. R. Fiete. Accurate path integration in continuous attractor network models of grid cells. *PLoS Comput Biol*, 5(2):e1000291, Feb 2009.

S. Caron, V. Ruta, L. F. Abbott, and R. Axel. Random convergence of afferent olfactory inputs in the drosophila mushroom body. *Nature*, 497:113–117, 2013.

M. Churchland, A. H. Lara, and J. P. Cunningham. The motor cortex and supplementary motor area exhibit different population-level dynamics. *COSYNE annual meeting abstract*, 2016.

M. M. Churchland and K. V. Shenoy. Temporal complexity and heterogeneity of single-neuron activity in premotor and motor cortex. *J. Neurophysiol.*, 97:4235–4257, 2007.

M. M. Churchland, B. M. Yu, M. Sahani, and K. V. Shenoy. Techniques for extracting single-trial activity patterns from large-scale neural recordings. *Curr. Opin. in Neurobio.*, 17:609–618, 2007.

M. M. Churchland, J. P. Cunningham, M. T. Kaufman, S. I. Ryu, and K. V. Shenoy. Cortical preparatory activity: representation of movement or first cog in a dynamical machine. *Neuron*, 68:387–400, 2010a.

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, S. G. Lisberger, N. J. Priebe, I. M. Finn, D. Ferster, S. I. Ryu, G. Santhanam, M. Sahani, and K. V. Shenoy. Stimulus onset quenches neural variability: a widespread cortical phenomenon. *Nat. Neurosci.*, 13(3): 369–378, 2010b.

M. M. Churchland, J. P. Cunningham, M. T. Kaufman, J. D. Foster, P. Nuyujukian, S. I. Ryu, and K. V. Shenoy. Neural population dynamics during reaching. *Nature*, 487:51–56, 2012.

P. Dayan and L. F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. Cambridge, MA: MIT Press, 2001.

S. Denève and C. Machens. Efficient codes and balanced networks. *Nat. Neurosci.*, 19:375–382, 2016.

B. DePasquale, M. Churchland, and L. F. Abbott. Using firing-rate dynamics to train recurrent networks of spiking model neurons. *arXiv*, 2016. arXiv:1601.07620.

M. Diesmann, M.-O. Gewaltig, and A. Aertsen. Stable propagation of synchronous spiking in cortical neural networks. *Nature*, 402:529–533, 1999.

S. Druckmann and D. B. Chklovskii. Neuronal circuits underlying persistent representations despite time varying activity. *Current Biology*, 22:2095–2103, 2012.

S. Druckmann, L. Feng, B. Lee, C. Yook, T. Zhao, J. C. Magee, and J. Kim. Structured synaptic connectivity between hippocampal regions. *Neuron*, 81(3):629–640, Feb 2014.

C. Eliasmith. A unified approach to building and controlling spiking attractor networks. *Neural Comput*, 17(6):1276–1314, Jun 2005.

C. Eliasmith and C. Anderson. *Neural Engineering: Computation, Representation and Dynamics in Neurobiological Systems*. MIT Press, Cambridge MA, 2003.

C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, C. Tang, and D. Rasmussen. A large-scale model of the functioning brain. *Science*, 338(6111):1202–1205, Nov 2012.

B. Ermentrout. Reduction of conductance-based models with slow synapses to neural nets. *Neural Comput.*, 6:679–695, 1994.

D. Fisher, I. Olasagasti, D. W. Tank, E. R. F. Aksay, and M. Goldman. A modeling framework for deriving the structural and functional architecture of a short-term memory microcircuit. *Neuron*, 79:987–1000, 2013.

R. V. Florian. The chronotron: A neuron that learns to fire temporally precise spike patterns. *PLoS One*, 7:e40233, 2012.

J. Friedrich and W. Senn. Spike-based decision learning of nash equilibria in two-player games. *PLoS Comput Biol*, 8(9):e1002691, 2012.

S. Ganguli and H. Sompolinsky. Compressed sensing, sparsity and dimensionality in neuronal information processing and data analysis. *Annu. Rev. Neurosci.*, 35:485–508, 2012.

A. P. Georgopoulos, M. Taira, and A. Lukashin. Cognitive neurophysiology of the motor cortex. *Science*, 260:47–52, 1993.

W. Gerstner. Time structure of the activity in neural networks models. *Phys. Rev. E*, 51:738–758, 1995.

R. L. Goris, A. J. Movshon, and E. P. Simoncelli. Partitioning neuronal variability. *Nat. Neurosci.*, 17(6):858–865, 2014.

R. Gütig and H. Sompolinsky. The tempotron: A neuron that learns spike timing-based decisions. *Nat. Neurosci.*, 9:420–428, 2006.

D. Hansel and H. Sompolinsky. Modeling feature selectivity in local cortical circuits. In C. Koch and I. Segev, editors, *Methods in Neuronal Modeling, 2nd ed*, pages 499–566. MIT press, Cambridge, MA., Cambridge, MA., 1998.

O. Harish and D. Hansel. Asynchronous rate chaos in spiking neuronal circuits. *PLoS Comp. Bio.*, 11(7), 2015.

S. Haykin. *Adaptive Filter Theory*. Upper Salddle River, NJ: Prentice Hall, 2002.

G. Hennequin, T. Vogels, and W. Gerstner. Optimal control of transient dynamics in balanced networks supports generation of complex movements. *Neuron*, 82:1394–1406, 2014.

G. M. Hoerzer, R. Legenstein, and W. Maass. Emergence of complex computational structures from chaotic neural networks through reward-modulated hebbian learning. *Cereb Cortex*, 24 (3):677–690, Mar 2014.

J. J. Hopfield. Neural networks and physical systems with emergent collective computation abilities. *Proc. Natl. Acad. Sci.*, 79:2554–2558, 1982.

T. Hosoya, S. A. Baccus, and M. Meister. Dynamic predictive coding by the retina. *Nature*, 436 (7047):71–77, Jul 2005.

D. Hubel and T. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *Journ. of Physiol.*, 148:574–591, 1959.

D. Huh. private communication, 2015.

H. Jaeger. Controlling recurrent neural networks by conceptors. *arXiv*, 2014. arXiv:1403.3369.

H. Jaeger and H. Haas. Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communcation. *Science*, 304:78–80, 2004.

S. Jahnke, M. Timme, and R.-M. Memmesheimer. Guiding synchrony through random networks. *Phys. Rev. X*, 2:041016, 2012.

J. Kadmon and H. Sompolinsky. Transition to chaos in random neuronal networks. *Phys. Review X*, 5, 2015.

A. Kennedy, G. Wayne, P. Kaifosh, K. Alviña, L. F. Abbott, and N. B. Sawtell. A temporal basis for predicting the sensory consequences of motor commands in an electric fish. *Nat Neurosci*, 17(3):416–422, Mar 2014.

T. Kleindienst, J. Winnubst, C. Roth-Alpermann, T. Bonhoeffer, and C. Lohmann. Activity-dependent clustering of functional synaptic inputs on developing hippocampal dendrites. *Neuron*, 72(6):1012–1024, Dec 2011.

B. W. Knight. Dynamics of encoding in a population of neurons. *J. Gen. Physiol.*, 59:734–766, 1972.

G. La Camera, A. Rauch, H.-R. Lüsteer, W. Senn, and S. Fusi. Minimal models of adapted neuronal response to *In Vivo*-like input currents. *Neural Computation*, 16:2101–2124, 2004.

R. Laje and D. V. Buonomano. Robust timing and motor patterns by taming chaos in recurrent neural networks. *Nat. Neurosci.*, 16:925–933, 2013.

Y. LeCun. Learning processes in an asymmetric threshold network. In E. Bienenstock, F. Fogelman, and G. Weisbuch, editors, *Disordered Systems and Biological Organization*, pages 233–240. Springer, Berlin, 1986.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

D.-H. Lee, S. Zhang, A. Fischer, A. Biard, and Y. Bengio. Difference target propagation. *ICLR 2015*, 2015.

S. Lim and M. S. Goldman. Balanced cortical microcircuitry for maintaining information in working memory. *Nat Neurosci*, 16(9):1306–1314, Sep 2013.

A. Litwin-Kumar and B. Doiron. Slow dynamics and high variability in balanced cortical networks with clustered connections. *Nat. Neurosci.*, 15:1498–1505, 2012.

J. K. Liu and D. V. Buonomano. Embedding multiple trajectories in simulated recurrent neural networks in a self-organizing manner. *J Neurosci*, 29(42):13172–13181, Oct 2009.

M. London and M. Häusser. Dendritic computation. *Annu. Rev. Neurosci.*, 28:503–532, 2005.

M. Lukosevicius, H. Jaeger, and B. Schrauwen. Reservoir computing trends. *Künstl. Intell.*, 26: 365–371, 2012.

W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Comput.*, 14:2531–2560, 2002.

W. Maass, P. Joshi, and E. D. Sontag. Computational aspects of feedback in neural circuits. *PLoS Comput. Biol.*, 3:e165, 2007.

C. Machens, R. Romo, and C. Brody. Flexible control of mutual inhibition: a neural model of two-interval discrimination. *Science*, 207:1121–1124, 2005.

C. K. Machens. Demixing population activity in higher cortical areas. *Frontiers in Computational Neuroscience*, 4(126), 2010.

G. Major, M. E. Larkum, and J. Schiller. Active properties of neocortical pyramidal neuron dendrites. *Annu. Rev. Neurosci.*, 36:1–24, Jul 2013.

V. Mante, D. Sussillo, K. V. Shenoy, and W. T. Newsome. Context-dependent computation by recurrent dynamics in prefrontal cortex. *Nature*, 503(7474):78–84, Nov 2013.

E. Marder. Motor pattern generation. *Curr. Opin. Neurobiol.*, 10:691–698, 2000.

E. Marder and A. L. Taylor. Multiple models to capture the variability in biological neurons and networks. *Nat. Neurosci.*, 14(2):133–137, 2011.

J. Martens and I. Sutskever. Learning recurrent neural networks with hessian-free optimziation. *Proc. 28th Int. Con. on Machine Learning, (Bellevue, WA)*, 2011.

D. A. McCormick, B. W. Connors, J. W. Lighthall, and D. A. Prince. Comparative electrophysiology of pyramidal and sparsely spiny stellate neurons in the neocortex. *J. Neurophysio.*, 54 (4):782–806, 1985.

R.-M. Memmesheimer, R. Rubin, B. Ölveczky, and H. Sompolinsky. Learning precisely timed spikes. *Neuron*, 82:011053, 2014.

G. Mongillo, O. Barak, and M. Tsodyks. Synaptic theory of working memory. *Science*, 319(23): 1543–1546, 2008.

S. Ostojic. Two types of asynchronous activity in networks of excitatory and inhibitory spiking neurons. *Nat. Neuro*, 17(4):594–600, 2014.

S. Ostojic and N. Brunel. From spiking neuron models to linear-nonlinear models. *PLoS Comput. Biol.*, 7:e1001056, 2011.

R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. *Proc. of the 30^{th} Inte. Con. on Mach. Learn.*, 28, 2013.

J.-P. Pfister, T. Toyoizumi, D. Barber, and W. Gerstner. Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning. *Neural Comput.*, 18:1318–1348, 2006.

F. Ponulak and A. Kasiński. Supervised learning in spiking neural networks with ReSuMe: sequence learning, classification, and spike shifting. *Neural Comput.*, 22:467–510, 2010.

W. Potjans, A. Morrison, and M. Diesmann. A spiking neural network model of an actor-critic learning agent. *Neural Comput*, 21(2):301–339, Feb 2009.

K. Rajan, L. F. Abbott, and H. Somponlinsky. Stimulus-dependent suppression of chaos in recurrent neural networks. *Physical Review E*, 82:011903, 2010.

K. Rajan, C. Harvey, and D. Tank. Recurrent network models of sequence generation and memory. *Neuron*, 90:1–15, 2016.

A. Rauch, G. La Camera, H.-R. Lüsteer, W. Senn, and S. Fusi. Neocortical pyramidal cells respond as integrate-and-fire neurons to in vivo-like input currents. *J. Neurophysio.*, 90:1598–1612, 2003.

F. R. Reinhart and J. J. Steil. A constrained regularization approach for input-driven recurrent neural networks. *Differ. Equ. Dyn. Syst.*, 1 & 2(19):27–46, 2011.

A. Renart, P. Song, and X.-J. Wang. Robust spatial working memory through homeostatic synaptic scaling in heterogeneous cortical networks. *Neuron*, 38(3):473–485, May 2003.

A. Renart, N. Brunel, and X.-J. Wang. *Mean-field theory of irregularly spiking neuronal populations and working memory in recurrent cortical networks*. Computational neuroscience: A comprehensive approach, 2004.

A. Renart, J. de la Rocha, P. Bartho, L. Hollender, N. Parga, A. Reyes, and K. D. Harris. The asynchronous state in cortical circuits. *Science*, 327:587–590, 2010.

J. Reutimann, V. Yakovlev, S. Fusi, and W. Senn. Climbing neuronal activity as an event-based cortical representation of time. *J Neurosci*, 24(13):3295–3303, Mar 2004.

M. Rigotti, O. Barak, M. R. Warden, X.-J. Wang, N. D. Daw, E. K. Miller, and S. Fusi. The importance of mixed selectivity in complex cognitive tasks. *Nature*, 497:585–590, 2013.

A. Rivkind and O. Barak. Local dynamics in trained recurrent neural networks. *arXiv*, 2015. arXiv:1511.05222.

R. Romo and E. Salinas. Touch and go: Decision-making mechanisms in somatosensation. *Ann. Review of Neuroscience*, 24:107–137, 2001.

D. Rumelhart and J. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1, Foundations*. Cambridge, MA: MIT Press, 1986.

E. Salinas and L. F. Abbott. Vector reconstruction from firing rates. *Journ. of Comp. Neurosci.*, 1:89–107, 1994.

M. A. Schwemmer, A. L. Fairhall, S. Denève, and E. T. Shea-Brown. Constructing precisely computing networks with biophysical spiking neurons. *J Neurosci*, 35(28):10112–10134, Jul 2015.

H. S. Seung, D. Lee, B. Reis, and D. W. Tank. Stability of the memory of eye position in a recurrent network of conductance-based model neurons. *Neuron*, 26:259–271, 2000.

M. N. Shadlen and W. T. Newsome. Noise, neural codes and cortical organization. *Curr. Opin. in Neurobio.*, 4:569–579, 1994.

M. N. Shadlen and W. T. Newsome. The variable discharge of cortical neurons: implications for connectivity, computation, and information coding. *The Journal of Neurosci.*, 18(10):3870–3896, 1998.

O. Shriki, D. Hansel, and H. Sompolinsky. Rate models for conductance-based cortical neuronal networks. *Neural Comput.*, 15:1809–1841, 2003.

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

H. Sompolinsky, A. Crisanti, and H. Sommers. Chaos in random neural networks. *Phys. Rev. Lett.*, 61:259–262, 1988.

P. Song and X.-J. Wang. Angular path integration by moving “hill of activity": a spiking neuron model without recurrent excitation of the head-direction system. *J Neurosci*, 25(4):1002–1014, Jan 2005.

S. Song, K. Miller, and L. F. Abbott. Competitive hebbian learning through spike-timing dependent synaptic plasticity. *Nature Neurosci.*, 3:919–926, 2000.

I. Sporea and A. Grüning. Supervised learning in multilayer spiking neural networks. *Neural Comput.*, 25:473–509, 2013.

M. Stern, H. Sompolinsky, and L. F. Abbott. Dynamics of random neural networks with bistable units. *Phys. Rev. E*, (062710), 2014.

D. Sussillo. Neural circuits as computational dynamical systems. *Opin. Neuro.- biol.*, 25:156–163, 2014.

D. Sussillo and L. F. Abbott. Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63:544–557, 2009.

D. Sussillo and L. F. Abbott. Transferring learning from external to internal weights in echo-state networks with sparse connectivity. *PLoS One*, 7:e37372, 2012.

D. Sussillo and O. Barak. Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural Comput.*, 25:626–649, 2013.

D. Sussillo, M. M. Churchland, M. T. Kaufman, and K. V. Shenoy. A neural network that finds a naturalistic solution for the production of muscle activity. *Nat. Neurosci.*, 18:1025–1033, 2015.

I. Sutskever. *Training recurrent neural networks*. PhD thesis, University of Toronto, 2013.

D. Thalmeier, M. Uhlmann, H. J. Kappen, and R.-M. Memmesheimer. Learning universal computations with spikes. *arXiv*, arXiv:1505.07866, 2015.

P. Tino and A. J. S. Mills. Learning beyond finite memory in recurrent networks of spiking neurons. *Neural Comput.*, 18:591–613, 2006.

F. Triefenbach, A. Jalalvand, B. Schrauwen, , and J. Martens. Phoneme recognition with large hierarchical reservoirs. *Advances in neural information processing systems*, (23):2307–2315, 2010.

M. Tsodyks, T. Kenet, A. Grinvald, and A. Arieli. Linking spontaneous activity of single cortical neurons and the underlying functional architecture. *Science*, 286:1943–1946, 1999.

C. van Vreeswijk and H. Sompolinsky. Chaos in neuronal networks with balanced excitatory and inhibitory activity. *Science*, 274:1724–1726, 1996.

C. vanVreeswijk and H. Sompolinsky. Chaotic balanced state in a model of cortical circuits. *Neural Computation*, 10:1321–1371, 1998.

E. Vasilaki, N. Frémaux, R. Urbanczik, W. Senn, and W. Gerstner. Spike-based reinforcement learning in continuous state and action space: when policy gradient methods fail. *PLoS Comput Biol*, 5(12):e1000586, Dec 2009.

T. Vogels and L. F. Abbott. Signal propagation and logic gating in networks of integrate-and-fire neurons. *J. Neurosci.*, 25:10786–10795, 2005.

T. P. Vogels, K. Rajan, and L. F. Abbott. Neural network dynamics. *Annu. Rev. Neurosci.*, 28: 357–376, 2005.

T. P. Vogels, H. Sprekeler, F. Zenke, C. Clopath, and W. Gerstner. Inhibitory plasticity balances excitation and inhibition in sensory pathways and memory networks. *Science*, 334:1569–1573, 2011.

X.-J. Wang. Probabilistic decision making by slow reverberation in cortical circuits. *Neuron*, 36: 955–968, 2002.

P. Werbos. Backpropagation through time: what it does and how to do it. *Proc IEEE*, 78:1550–1560, 1990.

H. R. Wilson and J. C. Cowan. Excitatory and inhibitory interactions in localized populations of model neurons. *Biophysical Journal*, 12:1–24, 1972.