Modeling and Computational Framework for the Specification and Simulation of Large-scale Spiking Neural Networks

David James Herzfeld
Marquette University

MODELING AND COMPUTATIONAL FRAMEWORK FOR THE SPECIFICATION AND SIMULATION OF LARGE-SCALE SPIKING NEURAL NETWORKS

by

David J. Herzfeld, B.S.

A Thesis submitted to the Faculty of the Graduate School,
Marquette University,
in Partial Fulfillment of the Requirements for
the Degree of Master of Science

Milwaukee, Wisconsin

August 2011

ABSTRACT
MODELING AND COMPUTATIONAL FRAMEWORK
FOR THE SPECIFICATION AND SIMULATION OF
LARGE-SCALE SPIKING NEURAL NETWORKS

David J. Herzfeld, B.S.

Marquette University, 2011

Recurrently connected neural networks, in which synaptic connections between neurons can form directed cycles, have been used extensively in the literature to describe various neurophysiological phenomena, such as coordinate transformations during sensorimotor integration. Due to the directed cycles that can exist in recurrent networks, there is no well-known way to *a priori* specify synaptic weights to elicit neuron spiking responses to stimuli based on neurophysiology. Using a common mean field assumption, that synaptic inputs are uncorrelated for sufficiently large populations of neurons, we show that the connection topology and a neuron's response characteristics can be decoupled. This assumption allows specification of neuron steady-state responses independent of the connection topology.

Specification of neuron responses necessitates the creation of a novel computational framework which allows modeling of large populations of connected spiking neurons. We describe the implementation of a spike-based computational framework, designed to take advantage of high performance computing architectures when available. We show that performance of the computational framework is improved using multiple message passing processes for large populations of neurons, resulting in a worst-case linear relationship between the number of neurons and the time required to complete a simulation.

Using the computational framework and the ability to specify neuron response characteristics independent of synaptic weights, we systematically investigate the effects of Hebbian-style learning on the hemodynamic response. Changes in the magnitude of the hemodynamic responses of neural populations are assessed using a forward model that relates population synaptic currents to the blood oxygen level dependant (BOLD) response via local field potentials. We show that the magnitude of the hemodynamic response is not a accurate indicator of underlying spiking activity for all network topologies. Instead, we note that large changes in the aggregate response of the population may occur with a decrease in the overall magnitude of the BOLD signal. We hypothesize that the BOLD magnitude changed due to fluctuations in the balance of excitatory and inhibitory inputs in neural subpopulations. These results have important implications for mean-field models, suggesting that the underlying excitatory/inhibitory neural dynamics within a population may need to be incorporated to accurately predict BOLD signals.

ACKNOWLEDGEMENTS

David J. Herzfeld, B.S.

This research was furthered, in part, by the National Science Foundation awards OCI-0923037, CBET-0521602, and CTS-0521602. Additional assistance, especially for the preliminary research, was provided by the Richard W. Jobling Fellowship as well as the Anthony J. and Rose E. Bagoszzi Medical Research Fellowship.

I would like to thank the following people for their contributions to the completion of this thesis. Dr. Scott Beardsley, my thesis advisor, for his invaluable guidance throughout the construction and writing of this thesis and my entire undergraduate and graduate careers. Dr. Craig Struble furthered my understanding of high performance/throughput computing. His perspectives regarding software creation and evaluation have shaped my understanding of computer science throughout my undergraduate and graduate education. I also wish to thank Dr. Robert Scheidt for serving on my committee and his continued constructive feedback throughout the process. Dr. Lars Olson and Dr. Kristina Ropella provided valuable advice during my years at Marquette University. Without their support and friendship, this thesis would not have been possible.

Byron Galbraith provided technical assistance during the development of the original high performance simulator code. Byron also provided friendly competition during code development, the result of which dramatically increased the scale and speed of the underlying simulator environment.

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION & SPECIFIC AIMS

1.1 Specific Aims

A fundamental objective of computational neuroscience is to elucidate the neural processes involved in perception, cognition, and motor control. These processes involve neurons which produce and relay stereotypical action potentials (Adrian, 1926; Adrian and Zotterman, 1926) or continuous graded signals (Roberts and Bush, 1981) to other neurons.

The mechanisms underlying construction, transmission, and decoding of neural activity can be characterized at multiple spatial and temporal scales. Techniques for modeling neural activity typically focus on a particular spatiotemporal scale due to the computational cost associated with spanning a large spatial or temporal range. For example, spiking neural models typically use differential equations to examine neuronal activity within groups of hundreds of neurons over millisecond timescales (Noble and Stein, 1966; Hodgkin and Huxley, 1952; Burkitt, 2006). Rate-based models, which estimate neuronal firing rates, usually simulate physiological responses within populations of thousands of neurons over tens to hundreds of milliseconds (Hertz et al., 1991; Dayan and Abbott, 2001; Eliasmith and Anderson, 2002). Mean-field models simulate the aggregate rate activity of cortical circuits over millimeters of the cortex, encompassing thousands of neurons, over tens of milliseconds to seconds (Bojak et al., 2010; Touboul and Ermentrout, 2011). Due in part to computational restrictions and model assumptions, the emphasis on a particular spatiotemporal scale limits the ability of current models to make explicit experimental predictions relating functional changes

in neural processing and structure to large-scale population responses measured with fMRI, EEG, and MEG.

There are few neural models which explicitly bridge the experimental responses of individual spiking neurons with physiological measures of neural activity across large populations. The use of existing models, which typically focus on a particular spatiotemporal scale, are generally not well-suited for determining the effect on population dynamics caused by changes at the neuronal level. For instance, the effects of long-term synaptic plasticity within a neural population on the neurovascular response cannot be readily determined from existing mean-field models. We propose a modeling framework, capable of describing arbitrary connected neural networks, and a computational framework, capable of simulating these networks, which bridges the gap between spike-based neuron models and mean-field models to investigate sensorimotor processing and the effects of learning across spatiotemporal scales.

Here, we seek to develop a modeling and computation framework for the specification, construction, and simulation of recurrent spiking network models using neuron steady-state responses to known sensory stimuli typically reported in neurophysiology literature. The specific aims are:

Aim 1: Develop a modeling framework capable of describing independent (unconnected) and recurrent network topologies. This framework will allow specification of well-characterized neuron steady-state responses based on relevant biological and physiological literature. The modeling framework will be implemented as a series of programming scripts which specify a neural network in terms of neuron steady-state responses and synaptic weights.

Aim 2: Create a computational framework that can construct and simulate arbitrarily large networks of recurrently connection neurons specified in Aim 1. The computational framework will simulate neural responses

to user defined inputs and provide neuron parameter outputs (voltage, current, spike trains, etc.) at millisecond resolution. The computational framework will be be implemented in C and designed to take advantage of high-performance computing architectures when available.

Aim 3: Using the modeling and computational frameworks developed in Aims 1 and 2, implement a spiking model to characterize the effect of synaptic plasticity on the blood oxygen level dependent (BOLD) response. Using this model, the effects of synaptic plasticity on the magnitude of the neurovascular response will be examined, considering each neural population as a spatial point source.

1.2 Limitations of Existing Frameworks

We propose a modeling framework which explicitly bridges the experimental responses of individual spiking neurons with physiological measures of neural activity across large populations. We identify three key issues with existing model architectures that are addressed by the proposed framework. First, the focus on a particular spatiotemporal scale limits the ability of models to make predictions about population dynamics from functional changes in neuronal processing or connection structure. Second, the maximum and background rate responses of neurons in existing models are intimately coupled with the synaptic connection weights. Decoupling the connection topology and the steady-state neuron responses allows investigation of population effects from changes to either the connection structure or neuron responses independently. Finally, existing simulator environments are not explicitly designed to simulate large populations of spiking neurons with highly-structured connections. In the remainder of this section, we explore each of these issues in further detail.

1.2.1 Temporal Scale

Spiking models typically simulate the responses of small groups of connected neurons at sub-millisecond to millisecond resolution. In contrast, many rate-based models estimate neuron firing rates over tens to hundreds of milliseconds. The reduction in temporal resolution when moving from spike to rate-based models is attractive due to decreased computational demands. The interpretation of continuous signals associated with neuron firing rates as opposed to the highly discretized signals associated with spiking responses has facilitated their use in modeling the visual system (Nowlan and Sejnowski, 1995; Wang, 1995; Beardsley and Vaina, 1998; Beck and Neumann, 2010; Thielscher and Neumann, 2008), sensorimotor integration (Deneve et al., 1999; 2001), and the vestibular system, particularly with regard to the encoding of head position (Xie et al., 2002; Stringer et al., 2002a;b), among others. Such models implicitly assume that the timing of individual spikes does not significantly impact the representation of information in the brain. Recent studies have shown that the occurrence (or absence) of individual spikes in a neuronal network can impact dynamic and steady state responses (Izhikevich and Edelman, 2008). Rate-based models also incur quantization error which may affect population dynamics and decoding accuracy at the system output as a result of firing rate estimation (Herzfeld and Beardsley, 2010).

Mean-field models provide estimates of neuronal dynamics for large populations (i.e. those that can be measured using conventional imaging techniques) (Bojak et al., 2010; Coombes, 2010). The evolution of rate-based to mean-field models is largely due to the computational demands associated with evaluating the differential equations associated with each neuron. Many mean-field models make the assumption that suitably large populations of neurons can be

described by the evolution of a probability density function, evaluated with respect to membrane voltage and time (Nykamp and Tranchina, 2001; Deco et al., 2008). This simplification can significantly reduce the computational complexity of the simulation from N coupled differential equations (where N is the number of simulated neurons) to a single equation per population. However, this simplification limits the ability of the model to describe the effects of changes in neuronal processing on the population response. For instance, the effects of changing connection weights between neurons within a population are unclear in existing mean-field models. In addition, most mean-field models make steady state assumptions which may not be valid when connections exist within a population, particularly when synaptic inputs to individual neurons are correlated or when the network exhibits synchrony (Deco et al., 2008).

1.2.2 Synaptic Weight Structure

In spiking and rate-based modeling approaches, connection topologies are intimately coupled with steady-state neuron response properties drawn from neurophysiology literature and published single-unit response characteristics (Albright, 1984; Albright and Desimone, 1987; Tanaka et al., 1989; Amirikian and Georgopoulos, 2000). These neuron responses (steady-states), such as a neuron's maximum response or background rate, are particularly difficult to guarantee if the connection profile features directed cycles (recurrent connections). There is no well-known way to *a priori* specify connection weights to achieve desired neuron response properties; instead, weighting profiles are either manually tuned (Deneve et al., 2001) to ensure network stability or learned (Wang, 1995; Beardsley and Vaina, 1998; Beardsley et al., 2003) until neuron steady-states align with available measures from neurophysiology studies. Connection profiles which use manually tuned or learned weights are difficult to generalize to novel stimuli, since the network dynamics may

violate the initial neuron steady-state assumptions. For instance, the weighting structure of a neural network can be learned to evoke assigned maximum responses for a particular preferred stimulus. However, if the weighting structure is not sufficiently constrained, application of a non-preferred stimulus may result in network instability due to the topology of the learned connection profile. This instability may result in firing rates that exceed the prescribed maximum response, violating the assignment of preferred stimulus to which the neuron is most responsive. To overcome the inherent history associated with recurrently connected network topologies, recent studies have imposed constraints on the total input to neural populations as well as the input to individual neurons via nonlinear activating functions (Deneve et al., 1999; 2001), in an effort to ensure stability.

We propose a modeling framework to simulate spiking neurons which specifies model parameters in terms of neuron steady-states, drawn from the neurophysiology literature. Here, the technique used to achieve neuron steady-states is largely independent of the chosen connection topology.

1.2.3 Spike-based Neural Simulator

Implementation of this modeling framework necessitates the creation of a novel simulator (computational framework) which allows construction of large populations of connected spiking neurons. Existing simulation environments are focused on specific spatiotemporal scales. For instance, Neuron (Carnevale and Hines, 2006) can simulate multicompartment models and small populations of connected spiking neurons, but large-scale network models are intractable. The Genesis simulator has similar size limitations for models of spiking neurons (Bower and Beeman, 1998). The Nest simulation environment provides many of the features necessary to construct the required spiking network, but does not explicitly allow for structured connection topologies (Diesmann et al., 2002). None of these simulation

environments allow for specification of neuron background/maximum responses in populations of connected neurons which feature directed cycles.

1.3 Thesis Layout

The remainder of the thesis is organized into six chapters. Chapter 2 contains background material describing individual and connected models of neurons as well as information related to parallel (high performance) computing techniques. Chapters 3 and 4 outline the modeling and computational frameworks developed to construct and simulate large populations of connected neurons. Chapter 5 characterizes the effects of network structure and synaptic plasticity on the neurovascular response in two well-defined models of neuronal processing in a single area as well as multi-area visual motion processing. Concluding remarks are made in Chapter 6, providing commentary on the achievement of the specific aims outlined in Section 1.1. In addition, Chapter 6 describes areas of potential improvement to both the modeling and computational frameworks. Finally, appendices containing the details of several referenced models are provided for the reader.

2 BACKGROUND

We propose a modeling framework which explicitly bridges the experimental responses of individual spiking neurons with physiological measures of neural activity across large populations. The modeling framework relies heavily on the concepts of computational neural modeling. Implementation of the modeling framework necessitates the creation of a novel simulator (computational framework) which allows construction and simulation of large populations of connected spiking neurons. Using this joint modeling and computational framework, we develop a series of models which relate neuron spiking activity to the hemodynamic BOLD response.

In this chapter, we focus on the background material necessary for the creation of the modeling and computational frameworks. In Sections 2.1 and 2.2, we identify current research in the area of independent and connected neural models. In Section 2.3, we describe current high performance computing paradigms, required for the creation of the computational framework which is described in Chapter 4. Finally, we outline the current state of knowledge regarding the structure and temporal modeling of the neurovascular response for neural populations without spatial extent.

2.1 Independent Neural Models

The encoding, decoding, and transmission processes associated with the neural code are, at the lowest level, performed by individual neurons. Neurophysiology studies have sought to characterize the performance of both individual neurons and groups

of neurons (neural networks) for over a century. Adrian (1926) demonstrated that many neurons relay and produce stereotypical action potentials (spikes). More recent studies have also provided examples of graded continuous signals in the nervous system (Roberts and Bush, 1981).

Elucidation of the neural code can occur across multiple spatiotemporal scales. In general, however, the neural spike remains the smallest unit of information transmission in the nervous system. Models of individual neurons typically exist on a continuum spanning temporal scales. At one extreme, the timing between individual neural spikes has been shown to convey information about the environment. Studies have suggested that the precise timing of individual spikes may carry a significant amount of information in the brain (Stein et al., 1993; Abeles et al., 1994). Modeling at this temporal scale requires fine temporal resolution, typically sub-millisecond to millisecond. At the opposite extreme, the average number of spikes per unit time has been shown to contain information about the environment in the form of a rate code (Mountcastle et al., 1957). These "rate-based" neuron models can operate at a coarser temporal resolution since spike averaging can occur over a wide range of temporal scales. Such models implicitly assume that the timing of individual spikes does not significantly impact the representation of information in the brain.

Temporal and rate-based encoding exist on a temporal continuum. At the finest level of temporal resolution exist single or multi-compartment spike-based models (Hodgkin and Huxley, 1952; Nabi and Moehlis, 2011). These models typically operate at sub-millisecond to millisecond resolution. Rate-based models estimate neuron firing rates over tens to hundreds of milliseconds, and can be constructed by counting the number of generated spikes from a spiking neuron per time interval. Alternatively, many neuron models are explicitly constructed to avoid the need to generate individual spike events (Eliasmith and Anderson, 2002). While

a clear limitation of these inherently rate-based neuron models is that they cannot provide exact spiking timing information, the reduction in the temporal resolution can dramatically reduce the computational demands associated with neuron modeling. As an added benefit, the interpretation of the continuous signals associated with neuron firing rates as opposed to the discretized action potentials of spiking neuron models has facilitated their use in numerous computational models, including models of the visual system (Nowlan and Sejnowski, 1995; Wang, 1995; Beardsley and Vaina, 1998; Beck and Neumann, 2010; Thielscher and Neumann, 2008), sensorimotor integration (Deneve et al., 1999; 2001), and the vestibular system, particularly with regard to the encoding of head position (Xie et al., 2002; Stringer et al., 2002a;b), among others.

In the remainder of this section, we identify the major computational models associated with spiking neurons. We begin by discussing the representation of individual action potentials for a single neuron as well as the construction of spike trains via stimulus encoding.

2.1.1 Stimulus Encoding

Neurons can receive stimuli (input) from other connected neurons or from the environment, as in the case of sensory neurons. The process of converting stimuli into the all-or-none action potentials used to transmit information between neurons is referred to as encoding. Due to similarities between the temporal profiles of action potentials, these events are typically assumed to be stereotyped. Due to the relatively short duration of an action potential, approximately 1 millisecond, as compared to the time between neuron spikes (the interspike interval, typically >10

ms), the action potential can be modeled as a Dirac delta function,

$$\delta(t) = \begin{cases} 1, & t = t_n \\ \\ 0, & \text{otherwise} \end{cases} , \tag{2.1}$$

where $\int_{-\infty}^{\infty} \delta(t - t_n)\, dt = 1$. Therefore, a temporal series of action potentials, referred to as a spike train, $a(t)$, can be defined as

$$a(t) = \sum_n \delta(t - t_n) \ , \tag{2.2}$$

where t_n corresponds to the time of the n-th spike due to the presentation of a stimulus. The neuron's instantaneous firing rate, $\tilde{r}(t_n)$, can be defined as the reciprocal of the time between the arrival of two successive action potentials:

$$\tilde{r}(t_n) = \frac{1}{\Delta t} = \frac{1}{t_{n+1} - t_n} \tag{2.3}$$

The number of spikes, n, in a given time interval, T, can be determined by integration of the spike train, $a(t)$, as

$$n = \int_0^T a(t)dt \ . \tag{2.4}$$

Since the time interval between spikes can be highly variable, even to identical stimuli, the mean firing rate, r, is often used as a measure of neural activity,

$$r = \lim_{T \to \infty} \frac{\int_0^T a(t)dt}{T} \ , \tag{2.5}$$

where r is expressed in units of spikes/second or Hertz (Hz). In practice, the mean firing rate is usually obtained by averaging neural responses over a large number of stimulus presentations. Throughout this document we use the terms mean firing

rate and rate response interchangeably.

Many neurophysiology studies systematically present different stimuli and observe their effect on an individual neuron's mean firing rate. We use the term "*neuron response profile*" to describe the relationship between presented stimuli and the mean firing rate observed in neurophysiology studies. The neuron response profile encompasses input from one or more of the following sources: (a) sensory input directly (as in sensory neurons), (b) projections from neurons at earlier stages of cortical processing, (c) projections from neurons at the same level of cortical processing, or (d) top-down projections from later stages of cortical processing. Depending on the cortical area, particular characteristics of the sensory stimulus may be related to the neuron's rate response (e.g., direction, velocity, and location) via a "*stimulus response profile*."

A response profile which maps motion direction to mean firing rate for an idealized neuron is shown in Figure 2.1. In characterizing the neuron response profile, the preferred stimulus is the external stimulus which elicits the highest mean firing rate or maximum response. For instance, the neuron in Figure 2.1 fires at its maximum response when presented with a motion direction stimulus of π radians. The background response is the mean rate at which the neuron responds when no stimulus is presented. In Figure 2.1, the anti-preferred stimulus is the external stimulus which results in the lowest rate response or minimum response of the neuron. We use the maximum and background responses of a neuron to identify a set of steady-state responses to sensory stimuli; we refer to this set of steady-states responses generically as neuronal states.

The stimulus response profile is a modeling technique which may be used to account for the neural and/or sensory inputs that are external to a simulated neural population (or if neurons are assumed to be independent). The stimulus response profile describes the response of a neuron to either a sensory input directly (as in

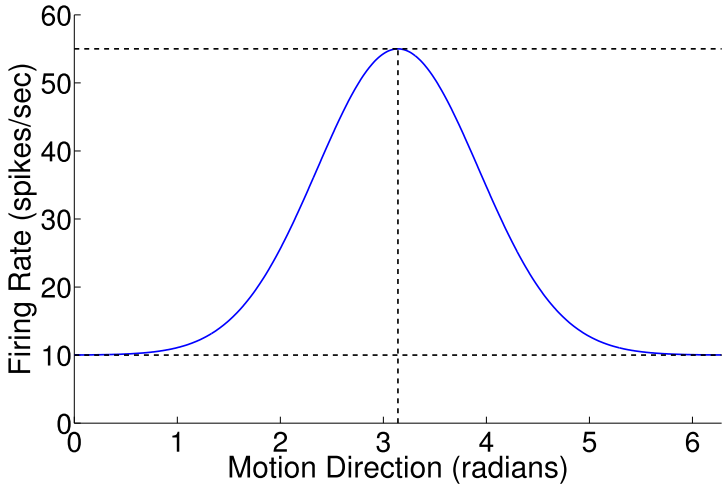


Figure 2.1: Idealized *neuron response profile*, mapping direction of motion from a sensory stimulus to the mean firing rate in units of spikes/second. The maximum response to the stimulus is 55 spikes/second (upper horizontal line), when a sustained motion direction of π radians is supplied to the neuron (vertical line). The neuron's mean firing rate at its minimum response (lower horizontal line) is 10 spikes/second when presented with its anti-preferred stimulus, approximately zero radians. The neuron responds at its background rate when no sensory stimulus is provided.

the case of sensory neurons) or to bottom-up projections from neurons in earlier stages of cortical processing not explicitly represented by the current model. The stimulus response profile does not incorporate the contribution of neurons from the same or higher levels of cortical processing.

Activating Functions

Rather than map stimuli directly to neuron firing rates, the neuron response profile may map the stimulus into an intermediate quantity (such as membrane current), which can then be used to determine the neuron firing rate. This intermediate result can be useful if the neuron responds to multiple sensory stimuli simultaneously, whose individual contributions to the overall spiking response may be nonlinear, exist in different units, or frames of reference.

In the case where the stimulus response profile is used to convert presented stimuli to an intermediate quantity, an additional relationship which maps this

intermediate quantity to neuron firing rates is required. We refer to this intermediate mapping to $r(t)$ as an activating function. Many studies assume the activation function to be sigmoidal due to the experimentally observed saturation in the firing rate of neurons for large input currents. Piecewise linear functions are also frequently used to impose saturation in the firing rate. For instance, a piecewise linear activating function which maps current into firing rate can be defined as

$$r(t) = \begin{cases} 0, & J(t)\eta < \gamma \\ R^{max}, & J(t)\eta > R^{max} \\ J(t)\eta - \gamma, & \text{otherwise} \end{cases}, \tag{2.6}$$

where $J(t)$ is the input current, η provides a scaling from the units of current to firing rate (e.g. from nA to Hz), R^{max} is the saturated response (maximum response) of the neuron, and γ is a spiking threshold measured in Hz. Since the input current of a neuron must be integrated to produce the membrane potential, a low-pass filtered version of $r(t)$ is sometimes used.

Inhomogeneous (Poisson) Spiking

The generation of spikes in the cortex is highly irregular, even for constant input to a neuron, which is partly why mean firing rates are often used for cortical modeling. If one assumes that the mean firing rate is the primary conveyor of information in the mammalian brain, the stochastic nature of spike times could reflect random processes occurring at the level of the individual neuron or across cortical networks. Further assuming that spike-time irregularities are due only to random cortical processes and not to neuron processes, then the timing of a given spike is independent of any preceding spike (i.e. there is no history dependence since the network has not history dependence). In this simplified case, the timing of an

individual spike can be described by a renewal process (Koyama and Kass, 2008). Experimental evidence has shown that neuron interspike intervals in response to a constant input are exponentially distributed, implying that instantaneous firing rates follow a Poisson process. In the case where stimuli are time-varying, neuron rate responses follow an inhomogeneous Poisson process, given by

$$\hat{n} = \int_0^T r(t)dt \ , \tag{2.7}$$

where $\hat{n}$ represents the mean spike count over a given time interval from $[0, T]$.

We note that the spiking in biological neurons is, to a degree, history dependent. For instance, following spiking, a neuron enters a refractory period in which the probability of spiking is extremely low. This is contrary to the strict interpretation of spike generation as a renewal process (particularly because the Poisson model has a high likelihood of spikes with short interspike intervals). These issues are further discussed in Chapter 3.

2.1.2 Spiking Neuron Models

There exists a variety of models designed to replicate the spiking characteristics of neurons. The primary differences between the various models usually lie in the level of biological detail incorporated into the mathematical description. In general, the more biological detail that a neuron model incorporates, the higher the computational cost. Here, we provide a general overview of several frequently used spiking neuron models, with a particular emphasis on the relationships between biological detail and computational cost.

Hodgkin-Huxley Model

The Hodgkin-Huxley (H-H) model neuron has been used extensively in neural models since its inception in 1952 (Hodgkin and Huxley, 1952). The H-H model uses a set of coupled differential equations to represent the spiking behavior observed in the squid giant axon. Typical implementations of the H-H neuron use two ion channels, corresponding to the movement of potassium and sodium ions across the neuron's cell membrane. An additional leakage current is normally modeled which defines the aggregate channel dynamics for all ions that are not explicitly described. In more recent implementations, the original H-H model has been extended to incorporate additional ion channel dynamics (Meunier and Segev, 2002).

Unlike some other neuron models, action potential generation is incorporated directly into the neuron's governing equations. If an external input causes the neuron's membrane voltage to rise, the conductance of the sodium channels also increases. If the positive feedback resulting from the inflow of positive sodium ions is large enough, an action potential is generated. Following spike generation, the neuron enters a brief refractory period. We note that this refractory period is relative; the neuron can spike during this period of hyperpolarization provided the external voltage/current input is sufficiently large.

Numerous studies have examined the ability of the H-H model neuron to emulate the spiking of *in vivo* recordings, generally noting that the H-H model provides a good representation of the spiking of actual neurons (see Meunier and Segev (2002) for a review). However, due to the number of coupled differential equations, the use of the H-H neuron in the simulation of large populations has been limited. Additional neuron models incorporate ion channel dynamics, such as the Fitzhugh-Nagumo model, have been proposed which reduce the number of coupled differential equations in an effort to increase computational tractability (Fitzhugh, 1961).

Leaky Integrate-and-fire Model

The leaky integrate-and-fire (LIF) model neuron provides a computationally simple spiking model and has been widely used in neural simulations. Typical LIF neurons contain both subthreshold and suprathreshold operating regimes. When the membrane voltage is under a predefined threshold, external input currents are integrated to produce a change in membrane voltage. An additional leakage current exists which allows the membrane voltage to decay back to baseline for insufficient input currents. The subthreshold dynamics of the LIF membrane potential can be modeled as an RC circuit, as shown Figure 2.2. Each of the passive components found in the equivalent RC circuit correlates to physiological neuron structures. The cell membrane (lipid bilayer) acts as a capacitor, separating charges. The resistance incorporated into the model is analogous to the embedded proteins in the lipid membrane, which facilitate the passive movement of charge across the membrane. Given an input current, $J^{in}(t)$, the time-varying voltage, $V(t)$, can be found by Kirchoff's current law, resulting in a first order differential equation of the form,

$$\frac{dV(t)}{dt} = \frac{1}{C}\left(J(t) - \frac{V(t)}{R}\right) \ , \tag{2.8}$$

where C and R represent the neuron's membrane capacitance and resistance, respectively.

When the membrane voltage, $V(t)$, reaches a predefined threshold, V^{th}, a spike is generated. This stereotypical action potential is modeled as a Dirac delta function. As described previously, the characterization of the action potential as a Delta function is valid provided the width of action potentials from *in vivo* recordings is suitably small compared to the interspike interval. Following spike generation, the membrane potential is reset to zero for a time, τ^{ref}, characterized by the absolute refractory period, during which an action potential cannot be

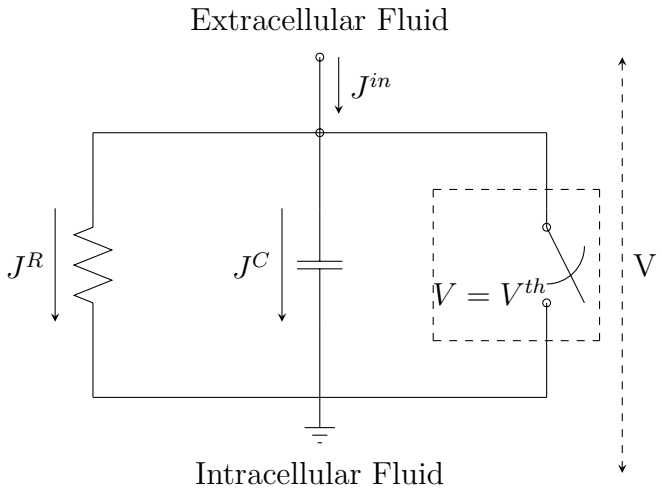


Figure 2.2: The RC equivalent circuit for a leaky integrate-and-fire (LIF) neuron when the voltage, V, is subthreshold. When the voltage is equal to the threshold voltage, $V = V^{th}$, the switch is closed, the action potential generator (dashed grey box) produces an an action potential. The switch remains closed, causing the voltage to reset ($V = 0$) until it is opened after some absolute refractory period, τ^{ref}.

generated regardless of the input current. Therefore, the maximum allowable firing rate for the LIF model is given by the reciprocal of the absolute refractory period.

Since the neuron dynamics are governed solely by a single differential equation linking the input current and membrane voltage, the LIF model is less computationally intensive than neuron models which incorporate ion channel dynamics, such as the Hodgkin-Huxley model. In addition, the LIF model has been shown to be the limiting case of more complex models (Partridge, 1966).

Canonical Neuron Models

Canonical neuron models seek to reduce an entire family of neurons into a parameterized model. The benefit of this approach is two fold. First, canonical models usually have a lower computational cost than their conductance-based counterparts. Second, by varying parameter values, entire families of neurons can be examined. Izhikevich (2003) demonstrate that the dynamics of conductance models,

such as the Hodgkin-Huxley model neuron, can be reduced to canonical models. Two well-known canonical models include the θ-neuron (Gutkin and Ermentrout, 1998) and a series of models proposed by Izhikevich (see Izhikevich (2003) for a review).

The θ-neuron models neuronal states using a spike trajectory and θ, a phase variable. The model can be written as

$$\frac{d\theta}{dt} = (1 - \cos\theta) + (1 + \cos\theta)(\beta + \sigma) \quad \theta \in [0, 2\pi] \ , \tag{2.9}$$

where the bias (due to noise) is represented by β and the input to the neuron model is σ. A spike occurs in the small region where $\theta \approx \pi$. Following spike generation, the neuron enters a refractory period since $(1 + \cos\theta) \approx 0$ when the value of theta is close to the spiking regime, $\theta \approx \pi$. For instance, with a constant input, the θ-neuron models fires periodically as θ wraps around the [0, 2π] polar space.

Izhikevich (2003) proposed a canonical model of type I neurons whose spiking dynamics are governed by two coupled differential equations:

$$\begin{aligned} \frac{dv}{dt} &= 0.04v^2 + 5v + 140 - u + I \\ \frac{du}{dt} &= a(bv - u) \ , \end{aligned} \tag{2.10}$$

where v and u are dimensionless time-varying signals and a, and b are dimensionless parameters. When $v \geq 30$, the neuron spikes and the value of v is set to the value of an additional dimensionless parameter, c. At this time, the value of u is set to be the sum of u and a fourth dimensionless parameter, d. Through parameter exploration, Izhikevich has identified the values of the dimensionless parameters (a-d) for "regular spiking" neurons as well as several other subclasses of type I neurons.

While canonical models provide both reduced computational cost, as compared to conductance based models, as well as the ability to explore entire

classes of neurons, typically the model parameters do not have equivalent physiological analogs. For instance, the θ-neuron model does not model the membrane voltage of biological neurons (although spike timing is modeled). Similarly, the models proposed by Izhikevich feature a number of dimensionless quantities whose values do not correspond directly to the physiological properties of neurons. The ability to relate model parameters to their physiological analogs is particularly important when investigating how changes in model parameters affect spiking dynamics. In addition, we note that the computational complexity of these canonical neuron models is approximately the same as that of the leaky integrate-and-fire neuron. The θ-neuron uses a single differential equation, identical to the LIF neuron, while the model proposed by Izhikevich has two coupled equations.

2.2 Connected Neural Models

The human brain contains more than 10^{11} neurons and more than 10^{14} synapses. Insights into brain function generally exists under the unifying theory that processing of information is performed by populations of connected neurons (Deco et al., 2008). Typical neurons in the mammalian CNS receive thousands of incident synaptic inputs (Dayan and Abbott, 2001). Therefore, investigation of brain function via models must incorporate both neuron and synaptic units. In this section, we first provide an overview of the physiology of synapses in the central nervous system. We then provide terminology used to describe connected populations of neurons.

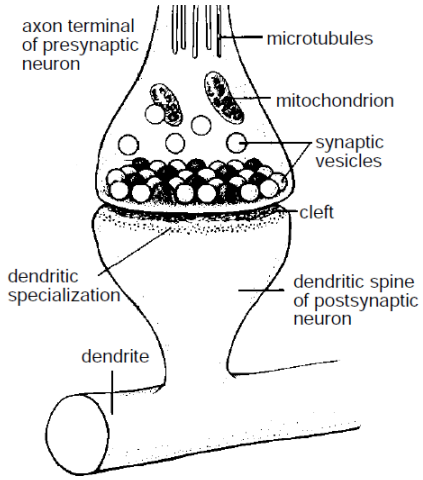


Figure 2.3: Schematic diagram of a synapse, from Dayan and Abbott (2001). The presynaptic neuron axon (top) contains synaptic vesicles which encapsulate neurotransmitter. When an action potential arrives at the axon terminal, neurotransmitter is released across the synaptic cleft where it later binds to receptor proteins on the postsynaptic cell. The neurotransmitter can cause excitatory or inhibitory effects on the membrane voltage of the postsynaptic cell.

2.2.1 Synapse Physiology

The synapse contains a minimum of three structures: the presynaptic neuron axon, the synaptic cleft, and the postsynaptic neuron dendrite (Figure 2.3). The transient voltage change due to a presynaptic neuron spike opens voltage-gated ion channels, allowing calcium ions to move from the extracellular space into the axon. The influx of Ca^{2+} ions leads to the release of neurotransmitter into the synaptic cleft.

Following diffusion across the synaptic cleft, the neurotransmitter binds to receptors on the dendrite of the postsynaptic neuron, causing ion channels to open. Based on ion flow into/out of the cell, the synapse can have either an excitatory (depolarizing) or inhibitory (hyperpolarizing) effect on the membrane voltage of the postsynaptic neuron.

2.2.2 Post-synaptic Current

The release of neurotransmitter from the presynaptic neuron across the synaptic cleft may cause either depolarization or hyperpolarization of the postsynaptic neuron's membrane voltage. The effect of ion flow through the ligand-sensitive gates can be modeled by describing the relationship between an excitatory postsynaptic potential (EPSP) or inhibitory postsynaptic potential (IPSP) and the neuron's membrane current. We define the postsynaptic current (PSC) as the change in ionic current through the cell membrane as a result of a single presynaptic spike.

The form of the PSC is well-characterized by an exponential decay process. Weber et al. (2003) described the postsynaptic current waveforms for ESPS's in rat Purkinje cells. They found that PSC waveforms decay from their peak value back to baseline within ∼15 milliseconds. Similar results were obtained by Wu et al. (2004) in a preparation of rat neurons. These results allow us to define a general mathematical form of the PSC, $h(t)$, as

$$h(t) = t^n \exp\left(-\frac{t}{\tau^{PSC}}\right) \ , \tag{2.11}$$

where τ^{PSC} is the time constant of the postsynaptic current and n is an integer value describing the order of the filter. Based on available evidence from *in vitro* preparations, the value of the n is typically small (0 or 1) and τ^{PSC} usually falls within the range of [5, 10] ms (Weber et al., 2003; Wu et al., 2004).

2.2.3 Neural Population

A neural population is a useful abstraction for computational models that contain hundreds to thousands of neural units. We define a population of neurons as a group of neurons which share a similar function. Based on neurophysiological evidence, the processing of neural signals in cortex exhibit anatomical and

functional structure, both among neurons within a local region of cortex and between separated regions. In sensory systems, the complexity of information tends to increase as signals progress from primary sensory areas to later stages of processing in the parietal and frontal cortices. In this case, a neural population typically reflects a group of neurons that exist at a particular stage of processing. For instance, in Chapter 5, we outline a computational model for a portion of the visual processing system. In this model, we define two explicit populations of neurons in the middle temporal and dorsal medial superior temporal areas, whose properties are well characterized by neurophysiology literature. In cases where there is insufficient physiological data to functionally define a neural population, we leave it up to the modeler to define these relationships explicitly.

2.2.4 Connection Classes

We define several broad types of connection classes which will be used throughout this document. Due to the synaptic physiology, we note that a synapse is, to a very good approximation, unidirectional. Therefore, all classes are directed, containing a source and destination neuron. A group of connections can be classified into a single connection class based on the populations containing their source and destination neurons.

Feed-forward connections exists between two distinct populations, representing the flow of information through a directed network (generally, a bottom-up connection from an earlier to later stage of processing). Feedback connections represent the opposite scenario, top-down connections which join neural populations at later stages of cortical processing to those at earlier stages of processing. Finally, recurrent connections exist within a population (i.e. the source and destination neurons are within the same population); this document also refers to these synapses as lateral connections. Additional connections may exist between

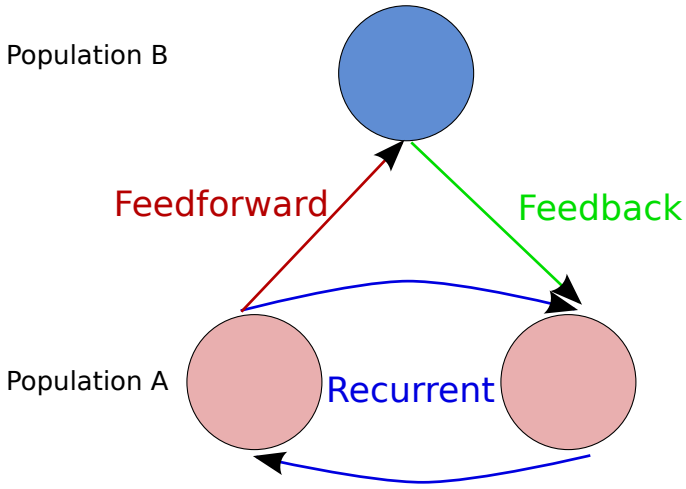


Figure 2.4: Connection classes which exist between and within neural populations. Population A (bottom) contains two neurons, represented by red circles. Population B (top) contains a single neural unit. Connections whose source neuron reside in population A and destination in population B are denoted as feed-forward (red line). Feedback connections, from population B neurons to A, are shown in green. Recurrent (lateral) connections exist between the two neurons in population A (blue lines). The distinction between feed-forward and feedback connections in this diagram is arbitrary. It is assumed that population B exists at a later stage of cortical processing.

two populations at the same stage of cortical processing. In these cases, the modeler must define a directed path, allowing classification of these connections as either feed-forward or feed-back. Figure 2.4 defines these relationships graphically.

2.2.5 Mean-field Models

Mean-field models provide estimates of neuronal dynamics for large populations, including those that can be measured using conventional imaging techniques (Bojak et al., 2010; Coombes, 2010). These models are a commonly used analytical approach for studying the dynamics of complex networks (Barrat et al., 2008). The evolution from rate-based to mean-field models is largely due to the computational demands associated with evaluating the differential equations associated with each neuron. Many mean-field models make the assumption that suitably large populations of similar neurons can be described by the evolution of a probability

density function, evaluated with respect to membrane voltage and time (Nykamp and Tranchina, 2001; Deco et al., 2008). This simplification can significantly reduce the computational complexity of the simulation from N coupled differential equations, where N is the number of simulated neurons to a single equation per population.

Derivations of mean-field models are typically performed using several assumptions. First, when examining the possible changes to the state of a particular neuron within a population, it is usually assumed that the state of its neighbors are independent of one another. This assumption may not be true if synaptic connections exist between neurons of the same population (i.e. lateral connections exist). In addition, modelers typically assume that all neurons with similar statistics can be described by a single "equivalent" neuron. Under these assumptions, the mean field activity for a population of neurons can be derived by analysis of the mean neuron response properties of the neurons in the population. In a general scenario, the mean firing rate, r, of the i-th cortical cell can be derived as

$$r_i = F(\mathbf{S}) + \sum_j w_{ij} r_j \ , \tag{2.12}$$

where F is a function which scales the multiparameter stimulus, $\mathbf{S}$, to units of spikes/second. The synaptic weight between the i-th and j-th neuron is given by w_{ij}. We implicitly assume that the derivation of r_j does not depend on the firing rate of the i-th neuron. We define $\bar{r}$ as the averaged firing rate of all of the cortical cells in a population with N neurons (this is not to be confused with the previous definition of mean firing rate, r):

$$\bar{r} = \frac{1}{N} \sum_i^N r_i \ . \tag{2.13}$$

The mean field approximation can then be obtained by replacing r_j in the sum in

Equation 2.12 by its average value,

$$r_i = F(\mathbf{S}) + \bar{r} \sum_j w_{ij} \ . \tag{2.14}$$

Assuming that the response of the neurons in the population to stimuli is located between threshold and saturation, the rate response across the population can be approximated as linear to the stimulus presentation (Cooper and Scofield, 1988),

$$r_i = m_i \mathbf{S} + \bar{r} \sum_j w_{ij} \ , \tag{2.15}$$

where m_i is the slope of the linearly approximated neuron response function. It follows that

$$\bar{r} = \bar{m}\mathbf{S} + \bar{r} w_0 = (1 - w_0)^{-1} \bar{m} \mathbf{S} \ , \tag{2.16}$$

where

$$\bar{m} = \frac{1}{N} \sum_i m_i \tag{2.17}$$

and

$$w_0 = \frac{1}{N} \sum_{ij} w_{ij} \ , \tag{2.18}$$

such that

$$r_i = \left(m_i + \left[\sum_j w_{ij} \right] (1 - w_0)^{-1} \bar{m} \right) \mathbf{S} \ . \tag{2.19}$$

It is possible to make an additional assumption that the network is largely inhibitory, mainly $w_0 < 0$, which ensures that the network is stable.

2.3 High Performance Computing

In recent years, the low price of commercial, off-the-shelf computers has allowed researchers to assemble distributed clusters with considerable power. The ability to

harness the computational abilities of computer hardware requires careful consideration of the algorithms involved as well as the expertise to implement these algorithms in robust software tools. This section focuses on the background material necessary for an understanding of the computational framework outlined in Chapter 4. The simulator described in that chapter is designed to take advantage of high performance computing (HPC) abilities when available. There are numerous computational paradigms for exploiting HPC abilities, many of which are tied to the underlying hardware architecture. We focus here on the message passing paradigm as other methods for parallelization are not used in the computational model presented in this document.

2.3.1 Message Passing

The message passing computational paradigm typically assumes a set of discrete processes (units of execution) which exist on the same or differing physical machines. Each of these processes have their own local memory, but also posses the ability to send and receive messages. The message passing paradigm typically assumes that sending and receiving messages requires operations to be performed by both the source and destination processes. A specific instance of the message passing paradigm, the Message Passing Interface (MPI) Standard, was completed in 1994 (Hempel, 1994).

2.3.2 Terminology

We define several terms that are encountered in the remainder of this document when referring to the message passing paradigm. We use the term “process” to refer to a unit of execution, with its own local memory (memory that is distinct from all other MPI processes). MPI applications must contain at least one process. These processes can be located on the same physical machine or on different machines

connected via some communication hardware (ethernet, infiniband, myranet, etc.). Collections of MPI processes form groups with definite size. Each member of a group is prescribed an integer rank from 0 to $n-1$, where n is the size of the group. MPI contexts are created at run time and are used for matching of messages. Groups of processes and the underlying context are combined into the concept of a communicator. Send/receive operations are usually performed between processes that are members of the same communicator (although some mechanisms exist to transmit messages between communicators). Interprocess communication can occur between two processes or by all processes in the communicator (termed collective communication).

2.4 Neurovascular Response

A common technique for the noninvasive mapping of brain activity during task execution uses functional magnetic resonance imaging (fMRI) to measure the blood oxygen level dependant (BOLD) response throughout the brain. BOLD contrast studies measure the hemodynamic response, which depends on blood oxygenation, flow, and volume (Nair, 2005). Correlation to neural activity is indirect, reflecting the energy demands of actively spiking neurons, necessitating the delivery of glucose and O_2 via cerebral blood flow. While studies have demonstrated correlations between neural activity and BOLD response (Heeger et al., 2000; Rees et al., 2000; Logothetis et al., 2001), interpretation of the signal in light of the underlying interconnections among neurons is difficult. This difficultly is partially due to the BOLD's dependence on the structure of the underlying vasculature (Nair, 2005) as well as the connected nature of neural networks. In addition, the temporal differences between neural activity (which change at a millisecond timescale) and the BOLD signal, which evolves over tens of seconds can confound the direct

interpretation of BOLD in terms of neural activity. In the remainder of this section, we provide a brief review of the neural mechanisms which affect the hemodynamic BOLD response.

2.4.1 Neurovascular Coupling

Glucose provides the primary energy source in the human brain. Simultaneous recording of neural activity and fMRI signals has shown that the hemodynamic response in primates is directly correlated to local field potentials (LFP's) (Logothetis et al., 2001), typically associated with the synchronous activity of neurons within 1-3 millimeters of the measuring electrode (Mitzdorf, 1985; Juergens et al., 1999). In additional to recording local fields, Logothetis et al. (2001) also examined the correlation between multi-unit activity, associated with the spiking of neurons within 300-400 micrometers of the electrode, and the BOLD response. They found that local field potentials correlate better to the hemodynamic response than the activity of multi-unit electrode recordings. These results suggest that the synchronous activity across neural populations is more closely related to the hemodynamic response than spiking activity alone.

Other studies have compared the activity of single unit electrode recordings to measured fMRI signals (Heeger et al., 1999; 2000; Rees et al., 2000). These studies provide evidence that the magnitude of fMRI signals is directly proportional to neural population rate responses. Rees et al. (2000) demonstrated that the rate response of neurons in area V5 (MT complex) to the coherence of motion stimuli is proportional to the amplitude of the hemodynamic response, such that an average increase of 9 spikes/second across a neural population resulted in a one percent increase in the BOLD signal. Heeger et al. (2000) also examined the linear relationship between coherence stimuli and the BOLD response, but focused instead on area V1. Similar to the results of Rees et al. (2000), the study found a linear

relationship between the aggregate rate activity and the hemodynamic response, with a proportionality constant of ~ 2.5, indicating that an average increase of 0.4 spikes/second in the V1 population resulted in a 1% increase in the magnitude of the BOLD signal. The disparities between the proportionality constants reported by the two studies may indicate differences in the underlying methodology or could indicate differences in the underlying functional connectivity of these two visual areas.

Logothetis (2008) notes that changes in the BOLD signal may be introduced by neural activity that is not related to aggregate mean firing rate. The excitatory-inhibitory role of recurrently connected cortical circuits has been shown to be involved in a variety of cortical activities (Douglas et al., 1995; Shadlen and Newsome, 1994; Chance et al., 2002). In this context, changes in BOLD activation could reflect changes in the balance of excitatory and inhibitory connections even though the aggregate rate activity across a neural population remains constant.

2.4.2 Nonlinear BOLD Responses

Previous studies have shown nonlinearities in the temporal profile of the BOLD response as a function of stimulus duration (Birn et al., 2001; Boynton et al., 1996; Dale and Buckner, 1997). These studies have shown a stronger response to short stimuli (whose length < 4 s), than would be expected given hemodynamic curves for longer stimuli. There are several hypotheses regarding the source of this nonlinearity. First, spiking at the level of the individual neuron tends to slow following an initial peak due to adaptation (Boynton et al., 1996). Second, longer stimuli may elicit cerebral blood oxygenation which is bound by an upper ceiling.

Liu et al. (2010) combined EEG and fMRI to further discern the timescales associated with nonlinearities introduced by neural adaptation and the blood flow ceiling. Their results demonstrate that steady-state neural responses in area V1 are linear when stimuli are spaced more than 194 milliseconds apart. Nonlinear neuron

responses, presumably due to neuron refractory effects, only affect the BOLD signal when interstimulus intervals were short (< 194 ms). They found that the nonlinearities introduced by neuron refractory effects depended almost exclusively on the interstimulus interval. Liu et al. (2010) also identify nonlinear vascular responses to neural activity when stimuli were spaced less than 4 seconds apart. They attributed this to a vascular refractory effect that is dependent on both the interstimulus interval as well as the absolute level of the BOLD response to a single stimulus.

2.4.3 Temporal Resolution & Functional Connections

The temporal resolution of fMRI analyses is limited by the signal-to-noise ratio associated with acquisition as well as the temporal characteristics of the hemodynamic BOLD response (Kim et al., 1997). On a 1.5T scanner, Bandettini et al. (1993) demonstrated that separable hemodynamic response to a finger tapping task could be made when the interstimulus interval was 8.0 seconds. When the interstimulus interval was 4.0 seconds, control versus tapping responses could not be separated. The resolution can be improved by employing signal averaging techniques, when the noise is Gaussian. Kim et al. (1997) extended these studies using a 4T scanner and found that temporal resolution increased to 3.0 seconds within a single area, confirming that the signal-to-noise ratio increases for high field scanners.

The temporal resolution within an individual area, which allows separation of two different stimuli, is significantly different than the temporal resolution between areas. When BOLD signals evolve in areas with differing time courses, the changes may not reflect differences in the neural events since, the hemodynamic response is influenced heavily by the vascular architecture. If the hemodynamic response time for all areas were equivalent, the order of neural activity could be

determined from fMRI acquisition unequivocally.

Determining the relative order of neural activity given a stimulus is useful, in that it can provide an estimate of the topology that connects activated cortical areas (i.e. functional connectivity). In addition, it provides information regarding the order in which neural areas are incorporated into and subsequently removed from the act of neural processing (Nair, 2005). That is, it provides an estimate of the temporal activity of large populations of neurons.

2.4.4 Effect of Excitatory/Inhibitory Connections

Both excitatory and inhibitory neural activity have been shown to evoke positive changes in the hemodynamic response (Attwell and Iadecola, 2002; Cauli et al., 2004; Fergus and Lee, 1997). Somewhat contradictorily, inhibitory connections have also been shown to induce decreases in the hemodynamic curve (Shmuel et al., 2006; Stefanovic et al., 2004). Bartels et al. (2008) hypothesizes that these contradictory results are due to different types of inhibition: either directly or via interneurons.

Logothetis (2008) has proposed that increases in the BOLD signal may result from changes in the relative balance of excitatory and inhibitory connections in a network. Balanced changes in inhibitory-excitatory levels, for instance, may result in increases in spontaneous spiking without a net increase in stimulus-related spiking activity. Logothetis (2008) also notes that changes in the BOLD response due to changes in excitatory-inhibitory connections likely depends on the underlying cortical area and its structure.

3 MODELING FRAMEWORK

Current research in the area of computational neural modeling has shown that populations of recurrently connected neurons can provide a maximum likelihood estimate of encoded signals (Deneve et al., 1999). While recurrently connected network topologies provide impressive results, the construction of many published neural models is not based on available neurophysiology. Due to the inherent history associated with directed cycles of neuron connections, stability of the neural systems can pose problems for researchers seeking to develop recurrent models. In addition, the prevalent use of rate-based models in recurrent network simulations may affect the overall network dynamics as temporal changes in spike patterns in the millisecond regime are difficult to represent in rate-codes estimated from tens of milliseconds.

In order to accurately capture the neural dynamics of cortical networks, we describe a spike-based modeling architecture capable of specifying neural networks with directed cycles. At a high level, this architecture consists of a single encoding model, capable of describing neuron responses in connected networks. In section 3.1, we initially assume that neurons are independent (i.e. their spiking activity does not contribute to other simulated neurons). Section 3.2 extends the independent encoding model to incorporate neuronal input due to interneuron connections.

3.1 Independent Neuron Stimulus Encoding

This section describes a biologically plausible neural model capable of encoding a presented stimulus. This model assumed that each neuron independently encodes

the presented stimulus (i.e. neuronal spiking does not affect the input of any other neurons).

3.1.1 Neuron Model

We elected to use leaky integrate-and-fire (LIF) neurons as a convenient encoding model that provides sufficient biological plausibility while maintaining computational tractability for large populations. The LIF neuron has a long history of use in biologically plausible neural models (Arbib, 2002; Koch, 2004). In addition, the LIF model provides similar results in the limit of more computationally demanding conductance-based models (Partridge, 1966). Other spiking models (e.g. Hodgkin-Huxley or Izhikevich) could be used.

Deterministic LIF Firing

The LIF membrane voltage, $V_j(t)$, of the j-th neuron in response to an input current, $J_j(t)$, can be found by integrating

$$\frac{dV_j(t)}{dt} = \frac{1}{C_j}\left(J_j(t) - \frac{V_j(t)}{R_j}\right) \ , \tag{3.1}$$

where C_j and R_j represent the neuron's membrane capacitance and resistance, respectively. We use the RC time constant, $\tau_j^{RC} \equiv RC$, to rewrite Equation 3.1 as

$$\frac{dV_j(t)}{dt} = -\frac{1}{\tau_j^{RC}}\left(V_j(t) - J_j(t)R_j\right) \ . \tag{3.2}$$

Solving Equation 3.2 for the time-varying neuron voltage, $V_j(t)$, yields

$$V_j(t) = J_j(t)R_j\left(1 - \exp(-t/\tau_j^{RC})\right) \ , \tag{3.3}$$

assuming that $J_j(t)$ is approximately constant over the interspike interval. When $V_j(t)$ in Equation 3.3 crosses a voltage threshold, V_j^{th}, a spike is generated at time t_n. As described in Section 2.1.1, the response of each neuron over time can be represented by a train of Dirac delta functions, given by

$$a_j(t) = \sum_n \delta(t - t_{jn}) \ , \tag{3.4}$$

where t_{jn} corresponds to the time of the n-th spike for the j-th neuron. After spiking, the neuron enters an absolute refractory period whose length is τ_j^{ref}. While in this refractory period, voltage integration does not occur.

As described above, a spike occurs after some time, t_j^{th}, by eliciting a neuron voltage greater than or equal to the threshold voltage. Solving Equation 3.3 for the temporal location of this spike,

$$\begin{aligned} V_j^{th} &= J_j(t)R_j\left(1 - \exp\left[-t_j^{th}/\tau_j^{RC}\right]\right) \\ t_j^{th} &= -\tau_j^{RC}\ln\left(1 - \frac{V_j^{th}}{J_j(t)R_j}\right) \ . \end{aligned} \tag{3.5}$$

The instantaneous firing rate, $\tilde{r}_j(t_j^{th})$, can then be found by noting that the interspike interval (ISI) is the reciprocal of the time it takes to reach threshold, t_j^{th}, and the absolute refractory period, τ_j^{ref}, such that

$$\tilde{r}_j(t_j^{th}) = \frac{1}{t_j^{th} + \tau_j^{ref}} \ . \tag{3.6}$$

As noted by Eliasmith and Anderson (2002), the instantaneous firing rate as a function of time can be found by substitution of 3.5 into 3.6:

$$\tilde{r}_j(t) = \frac{1}{\tau_j^{ref} - \tau_j^{RC}\ln\left(1 - \frac{V_j^{th}}{J_j(t)R_j}\right)} \ . \tag{3.7}$$

By Ohm's law, we note that $J_j^{th} = V_j^{th}/R_j$, which allows simplification to

$$\tilde{r}_j(t) = \frac{1}{\tau_j^{ref} - \tau_j^{RC} \ln\left(1 - \frac{J_j^{th}}{J_j(t)}\right)} \ . \tag{3.8}$$

Poisson LIF Firing

We note that the response of each neuron using Equation 3.3 is deterministic, varying only as function of the input current, $J_j(t)$. Available neurophysiology indicates that neuron rate responses elicited by a constant stimulus vary in a Poisson-like fashion (Koyama and Kass, 2008). We modified the deterministic LIF model to produce inhomogeneous Poisson distributed spike trains.

Poisson distributed spike trains of encoded stimuli can be rudimentarily obtained by perturbing the timing of each action potential. When a spike occurs, a measure of the instantaneous rate, $\tilde{r}(t^{th})$, can be determined using the reciprocal of the preceding interspike interval. This estimate of the instantaneous rate is then used as the mean of a Poisson distribution given by

$$g(\tilde{r}, k) = \frac{\tilde{r}^k exp(-\tilde{r})}{k!} \ . \tag{3.9}$$

A new instantaneous rate is randomly drawn from the Poisson distribution and the current spike's interspike interval is recalculated as the inverse of the new instantaneous rate. Figure 3.1 shows a plot of the mean spike rate and variance for a population of 150 neurons using this modified LIF model, where each neuron was presented with its preferred stimulus. The red line denotes a Poisson distributed response in which the mean equals the variance. It is important to note that the linear regression line associated with the instantaneous firing rates would not correspond exactly to a Poisson distributed response. This discrepancy is due to the inability to jitter firing times beyond the next spike (i.e. spike times can only

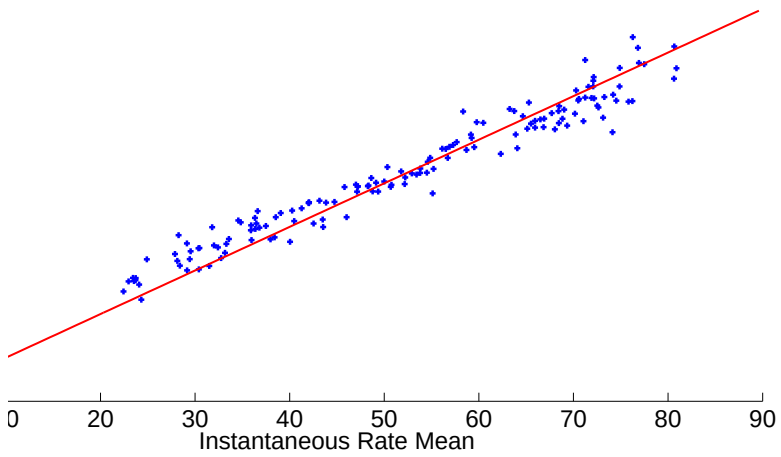


Figure 3.1: Mean and variance of the instantaneous firing rates for a population of 150 neurons with Poisson spike statistics presented with their preferred stimulus. Maximum firing rates were uniformly distributed between [40, 80] Hz. Instantaneous firing rates were calculated using the reciprocal of the interspike interval. The line denotes a Poisson distributed response in which the mean equals the variance. The figure is reproduced from Herzfeld and Beardsley (2010).

become shorter not longer) since the temporal sequence of action potentials must be maintained. While the responses are not exactly identical to a Poisson distribution, they do provide a good analog to Poisson firing using a *post hoc* method.

In order to elicit Poisson distributed spike trains which do not depend on the absolute timing of preceding spikes, which may not be available in some computational frameworks, we derive an additional method for spike generation. Provided the input current is roughly stationary over the integration interval (Δt), we use Equation 3.3 to solve for the temporal location, n, of the next spike, t_j^{th}. Therefore, a single spike exists over the interval of duration, $t_j^{th} + \tau_j^{ref}$. With a sufficiently small timestep, such that $\tilde{r}_j \Delta t \leq 1$, the probability of observing a particular spike during this interval is approximately $\tilde{r}_j \Delta t$. The exact position of a Poisson spike can then determined by evaluating the output of a uniform random generator from the interval [0, 1]. When the generated random number is less than or equal to the product of $\tilde{r}_j \Delta t$, a spike is generated.

Additional noise can be introduced into the encoding model by supplying an

additive noise to the input current. However, it is difficult to guarantee Poisson distributed instantaneous firing rates using this approach since the current is first passed through the nonlinear LIF neuron model before spikes are generated. Therefore, even if Poisson noise were added to the neuron input current, the spiking output of the neuron model would not be Poisson distributed.

Other Neural Models

We used LIF neurons as the basis for the encoding model. However, the modeling framework is general enough to incorporate other spiking neuron models, including multi-compartment and canonical models. A neuron model must posses two properties for successful inclusion in the modeling framework. First, the neuron must be capable of describing the instantaneous firing rate in terms of an input current (refer to Equation 3.3). Second, the neuron model must generate discrete action potentials.

3.1.2 Stimulus Response Profile

The total input current to a neuron, $J_j(t;\mathbf{S})$, whose value is a function of both time and a multi-parameter stimulus, $\mathbf{S}$, can be defined as

$$J_j(t;\mathbf{S}) = J_j(t;\mathbf{S}) + J_j^{spike}(t) + J^{noise}(t) \ , \tag{3.10}$$

where J_j^d is the driving current due to stimulus presentation, J_j^{spike} accounts for the contribution of the spiking activity of connected neurons, and $J_j^{noise}(t)$ reflects stimulus non-specific input currents that contribute to the neuron's background response. In order to define the input currents independent of the chosen neuron model, we normalized J_j^d and J_j^{spike} such that both quantities are less than or equal

to one. We redefine Equation 3.10 to account for these normalized quantities,

$$J_j(t;\mathbf{S}) = \alpha\left[\beta J_j(t;\mathbf{S}) + J_j^{spike}(t)\right] + J^{noise}(t)\ , \tag{3.11}$$

where α relates the normalized J_j^d and J_j^{spike} quantities to units of current, and the value of β serves to define the relative contributions of the driving and spiking currents to the total current. When neurons are unconnected, $J_j^{spike} = 0$ and β is assumed to be equal to one.

A stimulus response profile, $F(t;\mathbf{S})$, relates the multiparameter stimulus, $\mathbf{S}$, to the normalized driving current. This function serves to include neural and/or sensory inputs from neurons which are not explicitly incorporated into the model via an equivalent input current. Therefore, the presentation of a stimulus results in a change of the membrane driving current of a neuron,

$$J^d(t;\mathbf{S}) = F(t;\mathbf{S})\ , \tag{3.12}$$

whose amplitude is defined by the neuron-specific stimulus response profile. The underlying structure of $F(t;\mathbf{S})$ is based on the modeled cortical area.

Since a neuron may respond to multiple dimensions of a stimulus differently, the stimulus response profile, $F(t;\mathbf{S})$ may also define the functional relationships between each dimension of the stimulus and the driving current. For instance, neurons in the primary motor cortex have been shown to response linearly with respect to the speed of an intended movement (Moran and Schwartz, 1999; Paninski et al., 2004). The neuron responses to the direction of movement has also been shown to be well characterized by a von Mises curve (Amirikian and Georgopoulos, 2000). While Equation 3.12 allows for arbitrary transformations between the multidimensional stimulus and the driving current, we note that the relationship is

typically multiplicative. A multiplicative transformation,

$$J^d(t;\mathbf{S}) = \prod_{k=1}^{N} F_k(t;S_k) \ , \tag{3.13}$$

where k is the index in the N-dimensional stimulus space, provides a simple way to encode multiple stimuli (or multiple features of the same stimulus). This type of encoding is seen in primary motor cortex as well as gain fields in the posterior parietal cortex (Dayan and Abbott, 2001).

In order to define synaptic currents independently of the underlying neuron model, we have constrained $J^d \leq 1$. When the neuron is presented with its preferred stimulus, $\mathbf{S^{pref}}$, the neuron should fire at its maximum response, R^{max}. If we ensure that $F(t;\mathbf{S^{pref}}) = 1$, then we can find a scaling factor, α, which relates the input of the neuron when presented with its preferred stimulus to its maximum response. In the case of a leaky integrate-and-fire neuron, as presented in Section 3.1.1, α can be determined explicitly by substitution into Equation 3.3,

$$\alpha_j = \frac{J_j^{th}}{1 - \exp\left(\frac{\tau_j^{ref} R_j^{max} - 1}{\tau_j^{RC} R_j^{max}}\right)} - J_j^{noise} \ . \tag{3.14}$$

When the neuron is not presented with a stimulus, the driving current will be identically zero, $J_j^d(t;\mathbf{S}) = 0$. In the case where neurons are unconnected, $J_j^{spike} = 0$, only $J_j^{noise}(t)$ contributes to the background responses of the neurons.

3.1.3 Validation

We present a case study which seeks to validate the stimulus encoding framework presented in Section 3.1. We constructed a neural network model featuring 1,000 independent neurons, in which maximum responses, R_j^{max}, were assigned randomly from a uniform distribution between 40 and 100 spikes/sec. Neuron background rate

responses were assigned at 10% of the neuron's maximum response. RC integration time constants were assigned from a uniform distribution between 10 and 30 milliseconds. The absolute refractory period of each neuron was assigned uniformly from [2, 5] ms. Neuron spiking responses were Poisson distributed using the methods outlined in Section 3.1.1.

Each of the neurons was assigned a Gaussian stimulus response profile,

$$F_j(S;t) = \exp\left[-\frac{(S - S_j^{pref})^2}{2\sigma_j^2}\right] , \tag{3.15}$$

where S represented a supplied stimulus in the [0, 2π] polar space and σ_j represents the standard deviation of the Gaussian profile. Neuron preferred directions, S_j^{pref}, were uniformly distributed within the [0, 2π] polar space. Standard deviations in the direction tuning profiles were distributed randomly from $\pi/4$ to $\pi/2$ radians. A stimulus located at π radians was supplied to all neurons in the population for the duration of the simulation.

Neuron responses were simulated for a total of 50 seconds with quarter millisecond temporal resolution. Spike trains were recorded from all neurons in the simulation to facilitate analysis of spiking statistics in response to the supplied stimulus. The original source code for this independent model is provided in Appendix 7.1.

Figure 3.2 shows the normalized firing rate for the unconnected Poisson simulation plotted against neuron preferred stimulus directions. The figure shows the total spike count for each of the 1,000 neurons normalized to $R_j^{max} \times T$, where T is the total simulation duration (50 seconds), resulting in a normalized neuron rate responses from [0, 1]. Neurons whose preferred stimulus was near π radians show responses near 1, indicating that these neurons are firing near R_j^{max}. Neuron responses at 0 and 2π radians show a range of normalized firing rates. The upper

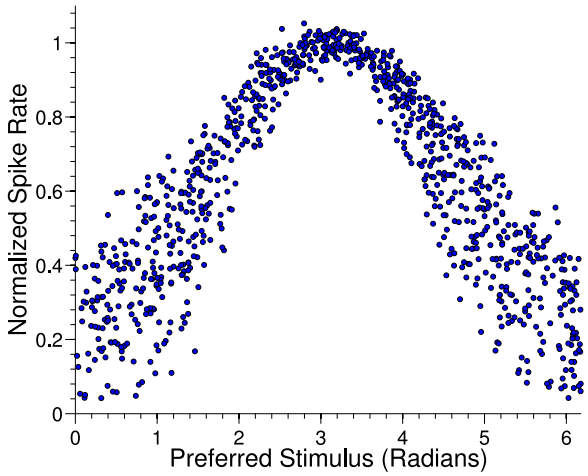


Figure 3.2: Normalized firing rates from independent Poisson neurons. Each neuron was Gaussian tuned to the stimulus defined in a polar space, with maximum responses ranging from [40, 100] Hz. The spike count of each neuron across the simulation has been normalized to its maximum response times the simulation duration (i.e. $R_j^{max} \times T$) to scale responses to the same range.

and lower bounds of this range correspond to Gaussian curves with standard deviations of $\pi/2$ and $\pi/4$, respectively.

An additional simulation was constructed with identical parameters to those described above, except we used the deterministic LIF neuron model described in Section 3.1.1. The simulation results are provided in Figure 3.3. Figure 3.3 is very similar to Figure 3.2, which is expected since stimulus response properties and LIF parameters were matched between the two sets of simulations. Of note, however, is the response of the neurons near their background firing rate. In the case of the deterministic LIF neurons, neurons do not response at rates lower than 0.1 (normalized). This value corresponds to the 10% background rate assigned to each neuron. In the Poisson distributed LIF neurons, neuron rate responses neuron whose preferred directions were near zero radians do fire at rates less than 0.1 (normalized). This discrepancy between the two populations is due to the distribution of interspike intervals in the case of Poisson-modified neurons, which, due to their random nature, allows normalized average firing rates less than 0.1.

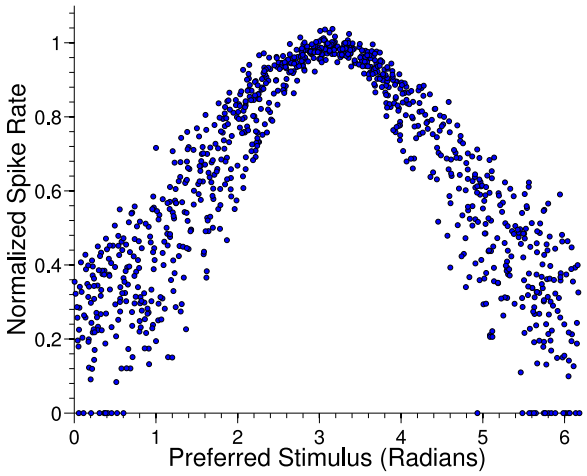
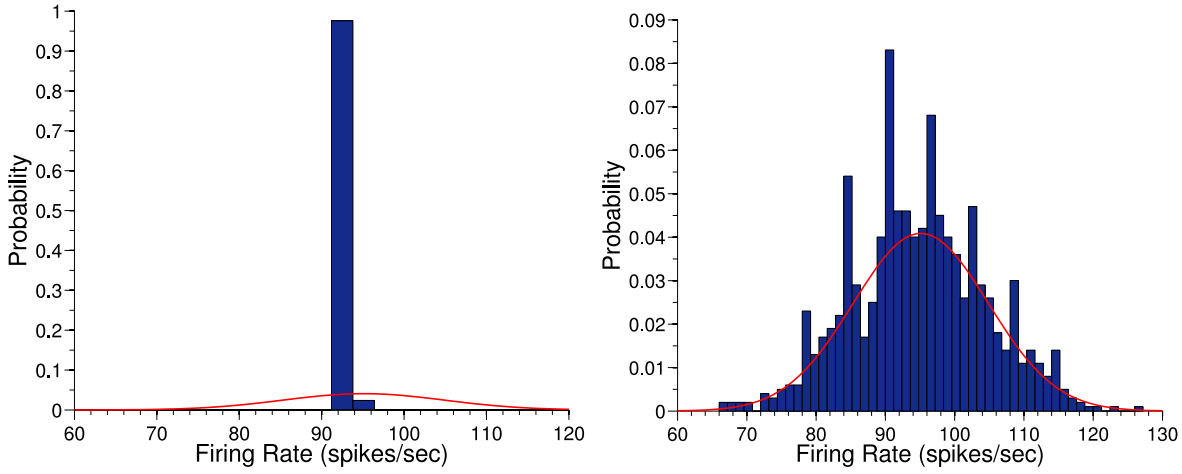


Figure 3.3: Normalized firing rates for a simulation featuring unconnected deterministic leaky integrate-and-fire neurons. Identical to the Figure 3.2, neurons were assigned Gaussian stimulus response profiles with maximum responses ranging from [40, 100] spike/sec. The total spike count of each neuron was normalized to its maximum response times the stimulus duration (i.e. $R_j^{max} \times T$).



(a) (b)

Figure 3.4: Firing rate probability density functions for deterministic and Poisson LIF unconnected neurons. Neuron parameters were identical to the simulations results shown in Figures 3.2 and 3.3 except that the simulation time was increased to 1,000 seconds. Firing rate probability density functions are plotted for a representative neuron with a preferred stimulus near π radians. The neuron's maximum response, R^{max}, was 95.1 spikes/sec. The red line in each plot shows a Poisson distribution centered at 95.1 spikes/sec. a) Probability density function for the deterministic LIF neuron model. b) Probability density function for the Poisson LIF neuron model.

We performed an additional series of simulations with a maximum time of 1,000 seconds. Once again, a stimulus located at π radians was supplied to all neurons for the duration of the simulation. We increased the simulation time in order to obtain an accurate estimate of the probability density function for the firing rates of the Poisson neurons. The results for both the deterministic and Poisson LIF neurons are shown in Figure 3.4. The red line in each plot shows a Poisson distribution centered at 95.1 spikes/sec, corresponding to this representative neuron's maximum response. The distribution of firing rates in the deterministic LIF case is very narrowly distributed about the neuron's maximum response. In contrast, the Poisson neuron model shows a distribution of firing rates that is close to the ideal Poisson distribution (red line).

3.1.4 Conclusions

As previously noted, this encoding model assumes that neurons encode the stimulus independently. In this simplistic case, the neuron response profile is identical to the stimulus response profile since lateral (recurrent) and feedback connections are not included in the network model. While this simplistic case violates a central motivation of biologically constrained neuron network modeling, mainly that networks of *connected* neurons perform computation, the model does provide a basis for more complicated network topologies. A rigorous extension to this model is described in Section 3.2 in which lateral and feedback connections are incorporated into the model architecture.

The independent encoding model, however, can capture numerous aspects of neurophysiology without recurrent or feedback connections. For instance, gain modulation, found in several cortical areas, included the primary motor cortex and the visual cortex can be modeled without including any explicit connections between simulated neurons. As an example, Pouget et al. (1995) describe gain

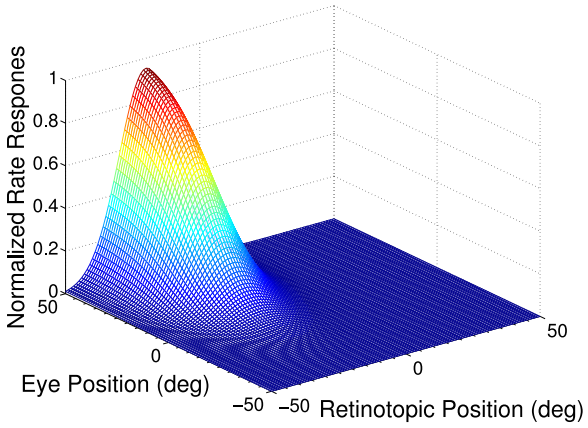


Figure 3.5: Gain modulation of a neuron in the posterior parietal cortex. The neuron's normalized response is shown as a function of both retinotopic position as well as the position of the eye within the orbit. Neuron responses are similar to those described in Pouget et al. (1995).

modulation of neurons in the posterior parietal cortex. These neurons show responses that are Gaussian tuned to the retinotopic coordinates of an object and simultaneously sigmoidally related to the location of the eye within the orbits. Such a neuron response can be accomplished by defining

$$F_j(t;\mathbf{S}) = \exp\left[-\frac{(S_r - S_{jr}^{pref})^2}{2\sigma_j^2}\right]\left[\frac{1}{1+\exp(-\gamma_j[S_e - S_{je}^{pref}])}\right], \tag{3.16}$$

where S_r represents the retinotopic coordinates of the object, S_e is the position of the eye within in the orbit, σ_j describes the standard deviation of the Gaussian response, and γ_j describes the width of the sigmoid. The neuron responds maximally when $\mathbf{S} = \{S_r, S_e\}$. A neuron with this stimulus response profile is shown in Figure 3.5.

3.2 Stimulus Encoding in Recurrent Networks

In this section, we extend the independent encoding model presented in Section 3.1 to allow synaptic input due to directed connections between neurons. Synaptic

connections from the i-th to the j-th neuron are assigned a scalar weight, denoted w_{ij}. With the addition of connections between neurons, the value of J_j^{spike} in Equation 3.10 is no longer constrained to be zero.

Representing the spiking of the i-th neuron at time t_n as a Dirac delta function, the j-th neuron's input due to the spiking can be written as

$$J_j^{spike}(t) = \sum_i \sum_n \delta(t - t_{in}) * w_{ij} h_j(t - T_{ij}^{delay}) \ , \tag{3.17}$$

where $h_j(t)$ is a post-synaptic current filter and T_{ij}^{delay} represents the transmission delay associated with the synapse. This relationship can also be written in terms of the action potential train of the i-th neuron, $a_i(t)$, as

$$J_j^{spike}(t) = \sum_i a_i(t) w_{ij} h_j(t - T_{ij}^{delay}) \ . \tag{3.18}$$

The post-synaptic current filter, $h_j(t)$, is well characterized as a simple exponential (Equation 2.11) due to the low-pass filtering effects associated with transmission of an EPSP/IPSP across the synaptic cleft.

Synaptic weights can be assigned functionally using $w_{ij} = H(\kappa_{\mathbf{ij}})$, which compares a vector of neuron response properties, $\kappa_{\mathbf{ij}}$, such as stimulus tuning or spatial location, specific to the presynaptic and postsynaptic neurons. The function, H, may be structured (e.g. a difference of Gaussians curve) or unstructured via the output of a pseudorandom number generator.

Given the definition of mean firing rate from Equation 2.5, the steady state input at j due the firing of i is directly proportional to the i-th neuron's time-varying firing rate, $r_i(t; \mathbf{S})$. Therefore, the input current to the j-th neuron is given by

$$J_j^{spike} = \lim_{T \to \infty} \frac{1}{T} \int_0^T w_{ij} r_i(t; \mathbf{S}) h_j(t) dt \ . \tag{3.19}$$

Table 3.1: Expected behavior of the neural network model to stimulus conditions. For connected populations of neurons, we make the assumption that the population is sufficiently large to produce the desired network response. The opposite case, where neurons are unconnected (i.e. independent), assumes that the conditions are defined by the independent encoding model, as outlined by Section 3.1.

Stimulus Condition	Connections	Model Response
No stimulus	Independent	All neurons fire at the assigned background rate, R^{back}, where R^{back} is related to J^{noise} through the neuron model.
No stimulus	Connected	All neurons fire at the assigned background rate, R^{back}, where R^{back} is related to J^{noise} through the neuron model.
$\mathbf{S_j^{pref}}$	Independent	All neurons fire at the assigned maximum response rate, R^{max}.
$\mathbf{S_j^{pref}}$	Connected	Neurons perform a maximum likelihood estimate of the stimulus. Maximum population responses are defined by R_j^{max}.

This reduces to $J_j^{spike} = w_{ij} r_i(\mathbf{S})$, provided the area of the post-synaptic current filter is normalized to one,

$$\frac{h_j(t)}{\int_0^T h_j(t)dt} \ , \tag{3.20}$$

and the stimulus is approximately constant over the filter length.

Given conventions in neurophysiology studies as well as other connected neuron models, we define the behavior of a neural network when presented with various stimuli, $\mathbf{S}$. The behavior of the network is defined in terms of neuron steady-state mean rate responses, $\lim_{T\to\infty} r(T;\mathbf{S})$, where, for convenience, we restrict $\mathbf{S}$ to be constant over time T. These steady-state responses are described in Table 3.1. In the case where neurons are assumed to be independent, the neuron behavior is dictated by the independent encoding model, as described in Section 3.1. The remainder of this section serves to define synaptic weights in order to elicit these steady-state rate responses.

3.2.1 Neuron Background Responses

As described in Table 3.1, many models incorporate an estimate of a neuron's background response, corresponding to the nominal activity of the neuron, independent of the presented stimuli. Identical to Section 3.1, it is assumed that this activity is due to synaptic input that is not accounted for directly by other modeled neurons. This stimulus-independent activity is incorporated via a nonzero value of J^{noise} in Equation 3.10.

When the network is not presented with a stimulus, the driving input, J^d, is, by definition zero. However, synaptic connections still exist within the network, providing input via J^{spike}. Therefore, when neurons in the network are firing at their background rates, we must ensure that the spiking input, $J^{spike}(t) \approx 0$ (i.e. the only contribution to the overall membrane current is due to J^{noise}). The mean spiking input from all other neurons firing at their background rates is given by

$$b_j^{spike} = \sum_i w_{ij} r_j(\mathbf{0}) \ , \tag{3.21}$$

where $\mathbf{0}$ represents a lack of a stimulus and $r_j(\mathbf{0})$ corresponds to the j-th neuron's rate response due only to J^{noise}. We use this background input to define a constant offset in the weight profile,

$$w_j^o = \frac{b_j^{spike}}{\bar{r}_j(\mathbf{0})M} \ , \tag{3.22}$$

where M is the number of incident synapses and $\bar{r}_j(\mathbf{0})$ is the mean background firing rate of all connected neurons:

$$\bar{r}_j(\mathbf{0}) = \frac{1}{M} \sum_{i=0}^{M} r_i(\mathbf{0}) \ . \tag{3.23}$$

This offset effectively balances the network's excitation and inhibition at background. The modified weight is given by $w_{ij} - w_j^o$.

3.2.2 Neuron Maximum Responses

Given the constraint that $\beta J^d(t;\mathbf{S}) + J_j^{spike}(t) \leq 1$, we note that $\beta J^d(\mathbf{S^{pref}}) + J^{spike}(\mathbf{S^{pref}}) \approx 1$, when a neuron is presented with its preferred stimulus. The magnitude of the incident synaptic weights can then be scaled by evaluating the response of the i-th neuron at the j-th neuron's preferred stimulus,

$$w_{ij}^{scaled} = w_{ij} \frac{1 - \beta J_j^d(\mathbf{S_j^{pref}})}{\sum_i w_{ij} r_i(\mathbf{S_j^{pref}})} \ , \tag{3.24}$$

where w_{ij}^{scaled} represents the normalized synaptic weight.

3.2.3 Neuron Response Profiles

In practice, it may difficult to evaluate $r_i(\mathbf{S_j^{pref}})$, particularly in a recurrent network where the i-th and j-th neurons may be bidirectionally connected. However, provided the modeled neural network is sufficiently large (thousands of neurons), the contribution of the i-th neuron's response to itself through its interaction with j is small (i.e. $w_{ij}w_{ji} \approx 0$). Computationally, this is equivalent to a common assumption of mean field models: for a neural population that is sufficiently large, incident synaptic activity is not correlated (Deco et al., 2008). This dramatically simplifies the evaluation of $r_i(\mathbf{S})$, for arbitrary stimuli.

The neuron response profile can then be determined explicitly given the neuron stimulus response and the connection topology among neurons, since the inputs are decoupled. This allows the specification of spiking neuron responses in terms of the familiar rate-based neuron response curves typically reported in neurophysiology studies, Therefore, the overall neuron response curve, typically measured in neurophysiology studies, can be found by substitution of Equation 3.19

into Equation 3.10,

$$r_j(\mathbf{S}) = \alpha_j \left(F_j(\mathbf{S}) + \sum_i w_{ij}\alpha_i \left[F_i(\mathbf{S}) + \sum_k w_{ik} r_k(\mathbf{S}) \right] + J_j^{noise} \right), \tag{3.25}$$

or equivalently,

$$r_j(\mathbf{S}) = \alpha_j \left(F_j(\mathbf{S}) + \sum_i w_{ij} r_i(\mathbf{S}) + J_j^{noise} \right), \tag{3.26}$$

The synaptic weights can be computed offline and then be used to obtain response profiles specified *a priori* for each neuron.

3.3 Case Studies

We provide examples from two case studies which illustrate how the modeling framework can be used to derive synaptic weights in order to elicit physiological responses in recurrent spiking neural networks. Again, we used leaky integrate-and-fire neurons as a convenient encoding model, however other spiking models can be used.

3.3.1 Single Layer Model

The first model consisted of a single population of 100,000 neurons featuring recurrent connections. Each neuron was assigned a Gaussian stimulus response profile,

$$F_j(t; S) = \exp\left[\frac{-(S - S_j^{pref})^2}{2\sigma_j^2} \right], \tag{3.27}$$

where S_j^{pref} was the neuron's preferred stimulus and σ_j was the standard deviation of the response profile in the stimulus space. Preferred stimuli were uniformly distributed within the [0, 2π] polar space. Standard deviations were randomized across the population and uniformly distributed between [$\pi/8$, $\pi/4$] radians.

Neurons' maximum response to their preferred stimuli were chosen from a uniform distribution between 40 and 80 spikes/second. The bias current, J^{noise}, was selected to produce a background firing rate that was 10% of the neuron's maximum response.

Recurrent connections among neurons were characterized by a Gaussian profile whose standard deviation was matched to the neuron's stimulus response profile, σ_j. Each neuron featured 10,000 pseudorandomly selected efferent synapses; the model, therefore, featured 10^9 total synapses. The synaptic weights were offset and scaled using the framework presented in Section 3.2. Figure 3.6(a) provides a schematic diagram of the neural network structure. The complete simulation specification code can be found in Appendix 7.2.

Figure 3.6(b-c) shows the results of a three second simulation. During the first second, no stimulus was supplied, allowing all neurons to spike at their background rates. A stimulus located at π radians was then supplied to the population for one second. The neuron shown in Figure 3.6 was assigned a preferred stimulus close to the presented stimulus, $S^{pref} \approx \pi$, and thus has a response near its assigned maximum. When the input stimulus was removed during the final second, neuron responses returned to baseline levels within 20 ms. Responses across the population show a Gaussian profile; neurons which preferred a stimulus of approximately π radians feature spike rates near their maximum response.

3.3.2 Cue Integration Model

In a second series of simulations, we used the modeling framework to characterize the temporal dynamics of cue integration. Using as a basis the rate-based model of cue integration proposed by Pouget and colleagues (Deneve et al., 2001; Avillac et al., 2005), we modeled the transformation of an object coded in eye-centered (retinotopic) coordinates, x_r, into head-centered coordinates, x_h. Provided the

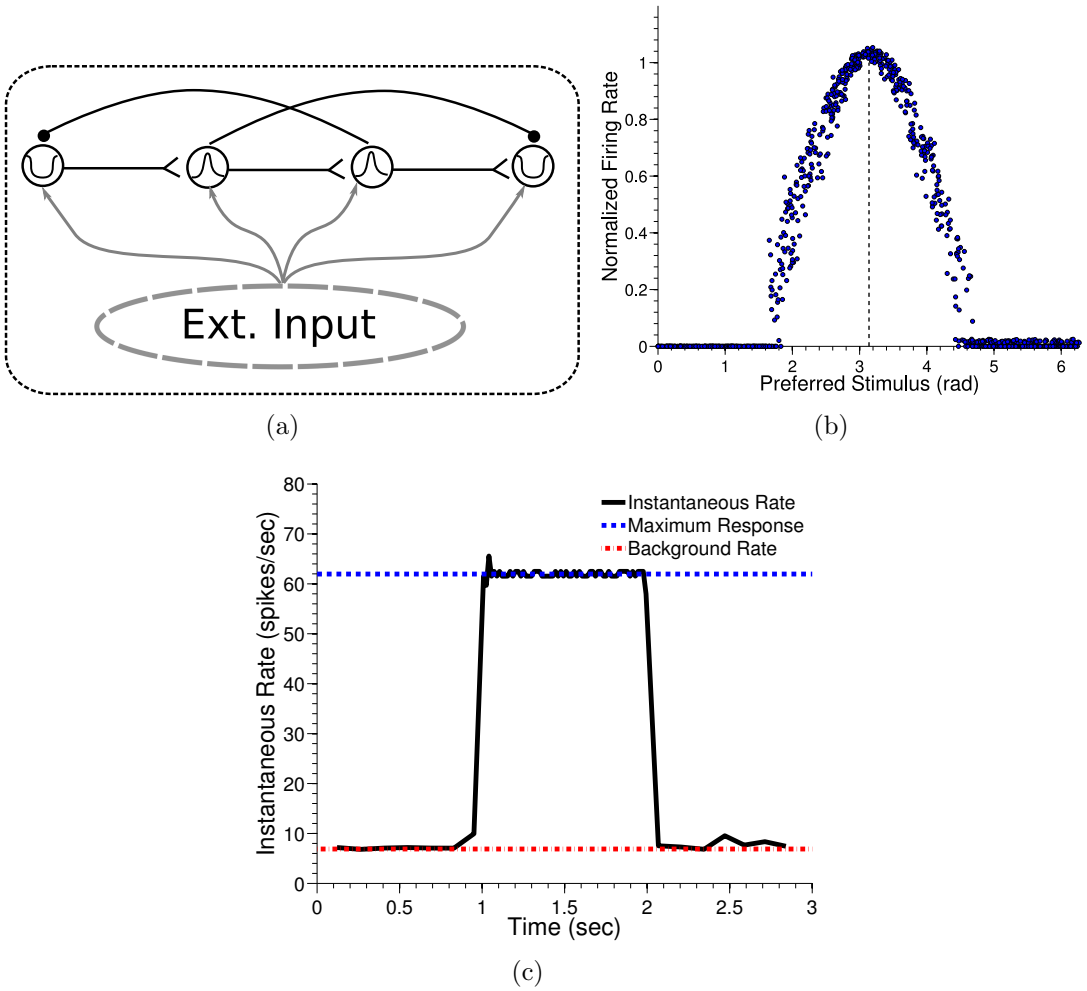


Figure 3.6: Neuron and population responses in a single layer recurrent spiking network. a) Schematic diagram depicting the structure of the network model. All neurons received an identical external input stimulus. Recurrent (lateral) connections existed between neurons in the population. b) Mean rate responses across the population for 1,000 randomly chosen neurons during presentation of a stimulus at π radians. Neuron rate responses are normalized to their respective maximum responses. c) Instantaneous firing rates, defined as the reciprocal of the interspike interval, plotted against time for a neuron which prefers a stimulus located at π radians. The stimulus was supplied from 1 to 2 seconds. The apparent anticipatory response of the neuron is due to the plotting of the instantaneous firing rates as a continuous function. The neuron does not actually spike until after the stimulus is supplied at 1 second.

position of the eye within the orbit, x_e, is known, the head-center coordinates of the object can be found by $x_h = x_r + x_e$.

Three neural populations, each consisting of 5,000 neurons, were used to encode head, eye, and retinotopic position of an object. Bidirectional connections

between these populations and an integration layer of 20,000 neurons followed difference of Gaussian profiles,

$$\begin{aligned} w_{ein} &= Ke^{\frac{-(S_{ei}^{pref}-S_{en}^{pref})^2}{2\sigma_n^2}} - e^{\frac{-(S_{ei}^{pref}-S_{en}^{pref})^2}{4.5\sigma_n^2}} \\ w_{rjn} &= Ke^{\frac{-(S_{rj}^{pref}-S_{rn}^{pref})^2}{2\sigma_n^2}} - e^{\frac{-(S_{rj}^{pref}-S_{rn}^{pref})^2}{4.5\sigma_n^2}} \\ w_{hkn} &= Ke^{\frac{-(S_{hk}^{pref}-S_{en}^{pref}-S_{rn}^{pref})^2}{2\sigma_n^2}} - e^{\frac{-(S_{hk}^{pref}-S_{en}^{pref}-S_{rn}^{pref})^2}{4.5\sigma_n^2}}, \end{aligned} \tag{3.28}$$

where S_{ei}^{pref}, S_{rj}^{pref}, and S_{hk}^{pref} correspond to the preferred eye, retinotopic, and head position of the i-th, j-th, and k-th neurons in the respective populations, w_{ein} indicates the synaptic weight from the i-th neuron in the eye position layer to the n-th neuron in the intermediate layer, and σ is the standard deviation of the connection topology associated with the destination population. Similarly, w_{rjn} denotes the connection for the j-th retinotopic layer neuron to the n-th intermediate layer neuron. Connection weights from the head-centered layer, w_{hkn}, were preferentially connected to neurons in the intermediate layer with preferred stimuli $S_e^{pref} + S_r^{pref}$. Figure 3.7 shows a schematic diagram of the neural network structure.

To facilitate comparisons with the rate based model from Deneve et al. (2001), each neuron was assigned a maximum response of 80 spikes/sec. We note, however, that the modeling framework does not required uniform maximum responses, as demonstrated by the first case study. Standard deviations in the stimulus response profile were uniformly distributed from $[\pi/16, \pi/8]$ radians across neurons. Coupled with randomly initialized membrane voltages as well as bias currents, this resulted in an initial noise that was greater than provided in Deneve et al. (2001) (Figure 3.8). All other neuron properties, were assigned as in the first case study. The specification of this simulation can be found in Appendix 7.3.

Neuron responses were simulated for one second, during which the driving input, J^d, supplied by Gaussian stimulus response profiles, provided a clamped

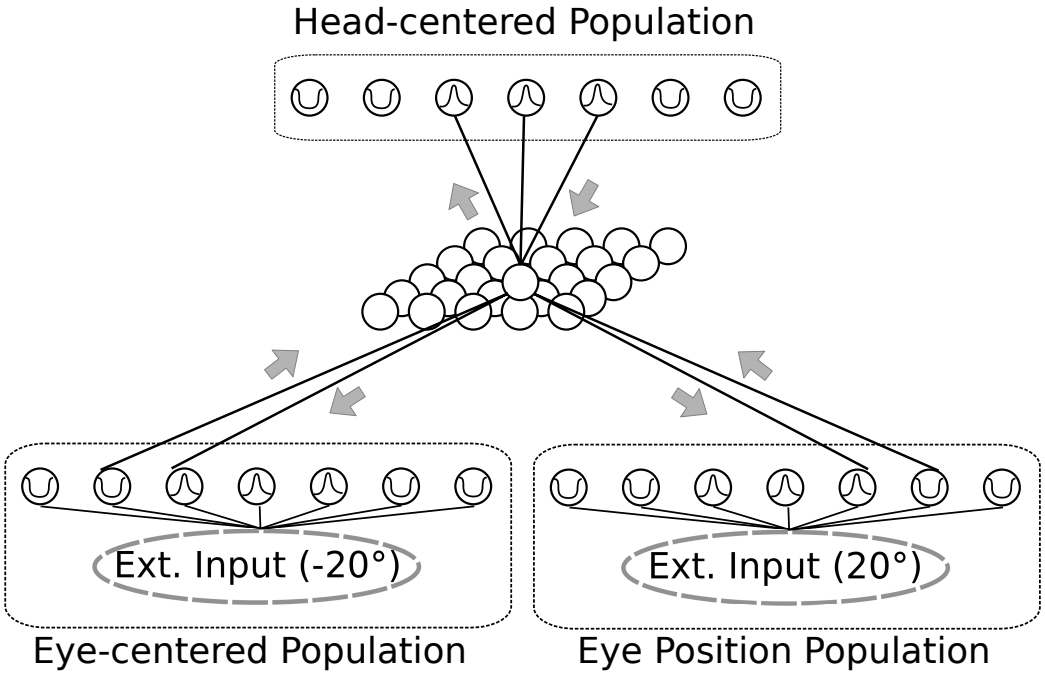


Figure 3.7: Schematic diagram of the neural network implemented to perform cue integration. Three unimodal populations encoded head, eye, and the retinotopic position of an object. An external stimulus was supplied to both the eye-centered and eye-position populations. Bidirectional connections existed between the three unimodal populations and an integration layer. The structure of the network was similar to a rate-based model of cue integration proposed by Deneve et al. (2001).

input of at most 20% when a neuron was supplied with its preferred stimulus. The remaining 80% of input at a neuron's preferred stimulus was supplied by lateral connections. This ratio of inputs is consistent with those presented in auxiliary simulations from Deneve et al. (2001) as well as cortical anatomy (Braitenberg and Schuz, 1991).

Figure 3.8 shows the mean firing rates for each of the neural populations when inputs to the eye-centered and eye position populations were -20° and 20°. The network was successfully able to perform cue integration, resulting in smooth hills of activity that stabilized within 50 ms.

There are two primary differences between the model supplied in Deneve et al. (2001) and the model implemented here: first, we used spiking neurons as opposed to rate-based basis function units. Second, the activity in the neural

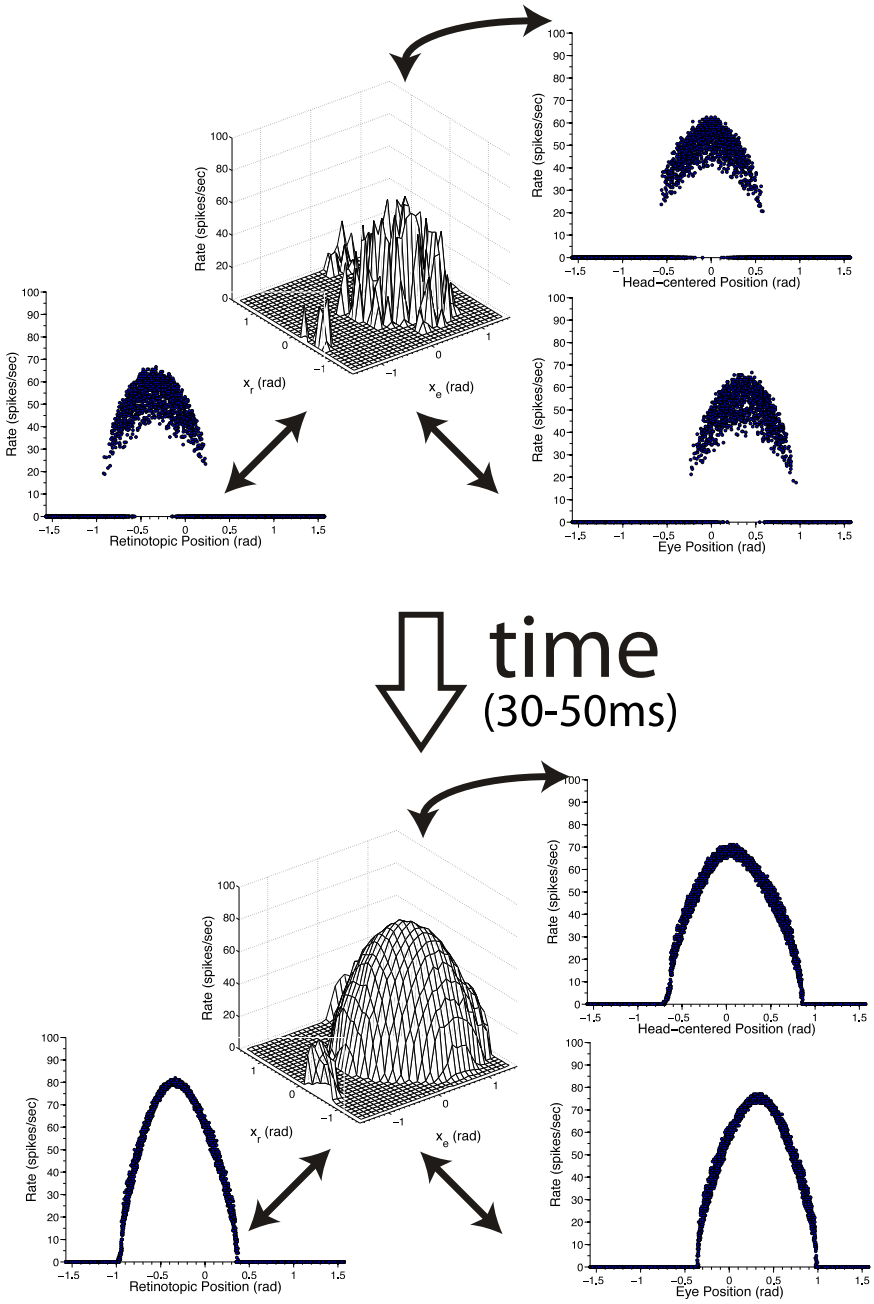


Figure 3.8: Neuron steady-state responses for a recurrently connected spiking network performing cue integration. The basic network structure has been described previously in Deneve et al. (2001). A driving input, Eq. 3.27, supplied a neuron with at most 20% of its total current at the preferred stimulus. The remaining 80% of the input at S^{pref} was supplied by recurrent connections. The network was initialized with noisy rate responses due to randomized initial voltages and a bias current, J^{noise}. Neural responses stabilized to smooth hills of activity in less than 50 milliseconds.

network described by Deneve et al. (2001) was initialized using a noisy probability density function and then allowed to relax over three iterations. These iterations cannot be directly linked to an absolute timescale; the spike-based model explicitly incorporates time, allowing future characterization of the temporal dynamics associated with cue integration tasks.

3.3.3 Conclusions

Using numerical simulations, we have developed a framework to specify connection weights in arbitrarily connected recurrent spiking networks, based on physiologically defined response characteristics. In particular, the background and stimulus-specific maximum responses of a neuron can be guaranteed when incident synaptic inputs are uncorrelated. This technique decouples the steady-state neuron responses from the connection topology, allowing *a priori* scaling of weights to elicit physiologic responses.

The primary assumption which allows us to dissociate neuron stimulus response profiles from the connection topology is that the neural population is sufficiently large to decouple incoming synaptic inputs. As the number of efferent connections per neuron increases, the contributions of secondary recurrent loops decreases quickly (i.e. $w_{ij}w_{jk} \to 0$, for arbitrary i, j, and k). We have found that population sizes on the order of thousands of neurons with several thousand efferent connections per neuron provides sufficiently small weights to support this assumption.

Most mean-field models of aggregate neural population activity assume that synaptic inputs are not correlated (Deco et al., 2008). Our approach validates this assumption in steady-state conditions for suitably large populations. More importantly, if facilitates the use of spike-based neural network models to characterize the dynamics of neural processing within populations. For example,

investigation of the temporal dynamics associated with the onset of stimulus presentation can be examined in an absolute timescale using the presented framework.

3.4 Modeling Framework Implementation

We implemented the modeling framework as a series of scripts which specify neural networks in terms of neuron steady-state responses. These scripts define abstract classes which are necessary for specification of simulation parameters. These classes include populations (groups of similar neurons), individual neurons, stimulus response relationships, and stimuli. The scripts provide several specific implementations of these abstract classes. For instance, the framework provides neuron models for both the LIF and Poisson distributed LIF neurons. Additional types of neurons can be specified by extending the abstract "neuron" class for the new model. The framework typically provides the user with a minimum of three files which can be later used to construct and simulate the network. First, an XML file which specifies all of the neuron parameters is required as an input to the simulator (refer to Section 4.2.1 for more information). In addition, when custom network connection functions are used (i.e. different $F(\mathbf{S};t)$), the implemented modeling framework typically outputs a file listing the weighted synaptic connections among neurons. Finally, one or more stimuli files are usually used as neuronal inputs during simulation runs. Examples of the scripts used to generate these files are provided in the Appendices.

4 COMPUTATIONAL FRAMEWORK

There are numerous existing neural simulation packages available, however most are focused on a particular spatial or temporal scale. Therefore, implementation and evaluation of the modeling framework described in Chapter 3 necessitated the creation of a novel simulator which allows construction of large populations of connected spiking neurons across various spatiotemporal regimes. The implementation of this simulator is intentionally generic, allowing the simulation of various network topologies using differing neuron models. Therefore, we refer to this simulation environment generically as a "computational framework."

Due to the computational demands associated with simulating large populations of spiking neurons, where each neuron is characterized by one or more differential equations, the computational framework is designed to take advantage of high performance computing abilities when available. Incorporation of high performance computing techniques provides two primary benefits compared with a single system/process implementation. First, the evaluation of the differential equations describing each neuron can be divided across multiple processes using message passing techniques. This can effectively reduce the time required to perform a simulation. Secondly, the memory requirements of the simulation can be divided across multiple physical processors or systems. This increases the total number of neurons that can be simulated. However, the creation of a simulation environment which uses message passing abilities has several drawbacks. First, the programming of the underlying simulator may be more complex than a single-process implementation. In addition, the time required to communicate messages between processes must be carefully weighted against the advantages of

performing computations in parallel.

This chapter describes the implementation of a computational framework which can simulate populations of spiking neurons specified by the modeling framework. We first describe the general simulation structure and then each of the components in detail. Particular emphasis is placed upon the portions of the computational framework which provide advantages in a high performance computing environment. Finally, we provide a series of benchmarks which evaluate the simulator for a number of overdriven network models. These benchmarks provide evidence for the improvements in speed possible using MPI on multiple processors compared with an identical single processor implementation.

4.1 Simulation Flow

Each simulation can be broken down into two general pieces: initialization and execution. Both of these pieces can be further subdivided into its constituent elements which will be described later. Figure 4.1 provides an overview of the basic steps involved in each simulation. In general, the initialization phase performs any setup necessary to perform execution of the simulation. This includes reading in a set of specification files which describe the initial states and characteristics of the neural populations. During this stage, synaptic connections between neurons are also constructed. Both the specification and connection files are typically created by the high level interface described in Section 3.4, as the output of the modeling framework.

After the neural populations and connection topology have been initialized, the execution phase begins. The simulation advances using a user-defined fixed timestep. Each timestep serves to update neuronal inputs and subsequently the states associated with the differential equations for each neuron. In addition, the

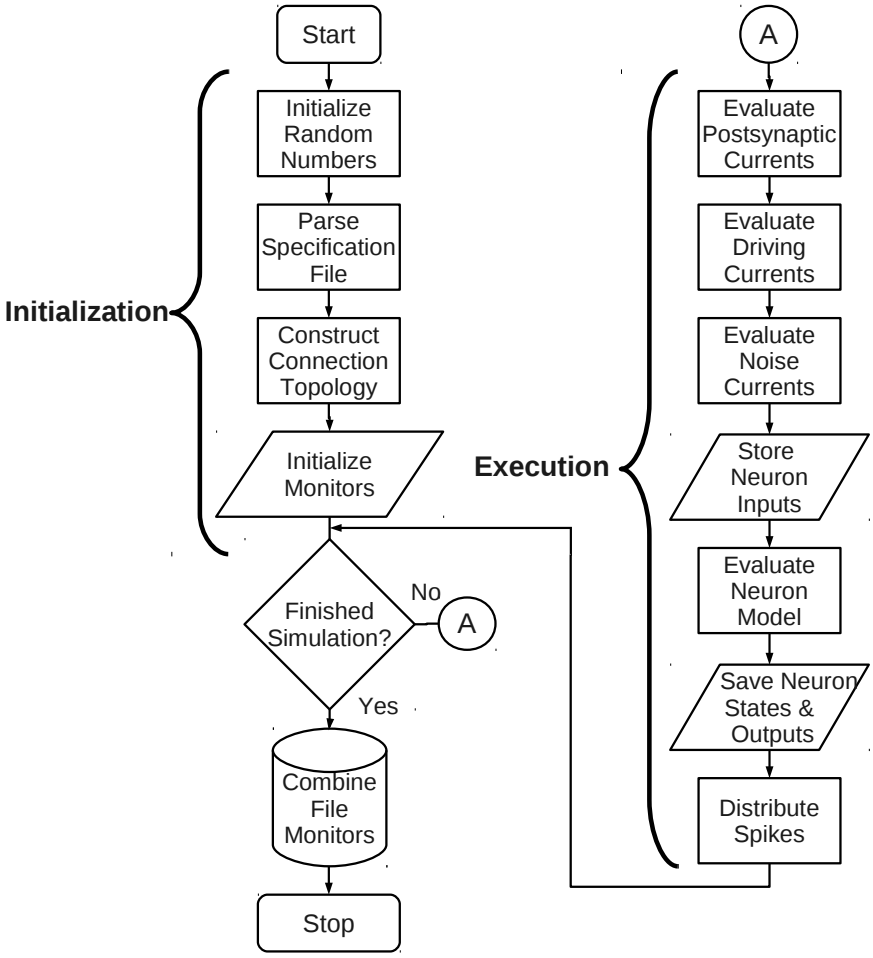


Figure 4.1: High-level schematic diagram indicating the flow of each simulation. Each simulation can be divided into two phases: initialization and execution. The execution phase proceeds using a fixed timestep until the total simulation duration has been reached. Following execution, there is a brief period of cleanup in which monitor files are combined.

spiking outputs for each neuron are distributed to all connected neurons. During the execution phases, any requested output values, which may include neuron inputs, states, or time-varying parameters are saved for later offline processing and review. The execution phase continues until the total simulation time has been reached. Once the execution phase has completed, a brief clean-up is performed on the simulation outputs for the convenience of the user.

4.2 Initialization

4.2.1 Simulation Input

The input to the simulation is created by the high-level implementation of the modeling framework, described in Section 3.4. This specification is stored using the Extensible Markup Language (XML), version 1.0. The document type definition (DTD) for the specification of a simulation is shown in Figure 4.2. The DTD is used to validate the XML specification file prior to parsing. Parsing of the XML document is performed using the `libxml2` parser and toolkit.

4.2.2 Neuron Models

In order to ensure that the computational framework is capable of simulating different types of neurons, the underlying code was designed to be independent of the specific neuron model. In Section 3.1.1 we outlined deterministic and Poisson LIF model neurons. These two neuron models satisfy the general requirements of inclusion of a neuron model in the computational framework. First, the neuron model must be capable of accepting synaptic current as its input. Secondly, the model must be capable of generating events as outputs. Since these events are assumed to be neuron spikes, which are stereotyped, the temporal profile of these

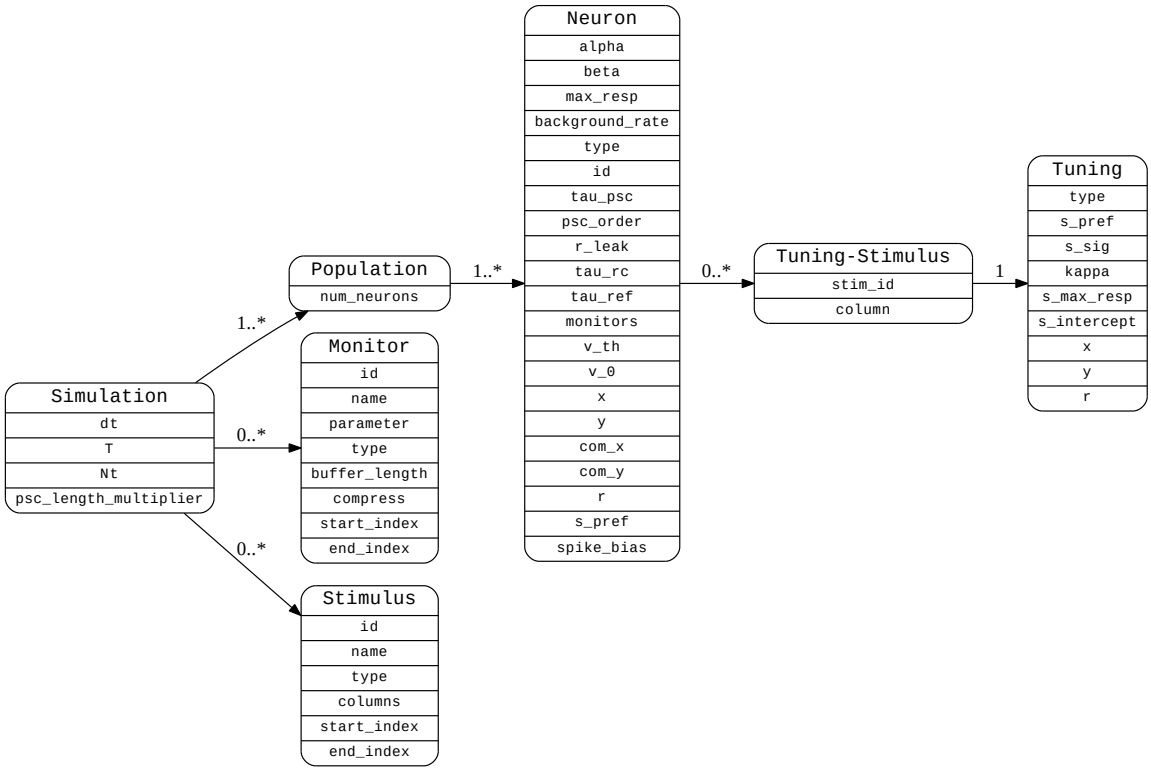


Figure 4.2: A graphical representation of the schema used to specify a simulation. The relationships between each of the DTD elements is provided as arrow labels. A label of "1" indicates a one-to-one relationship between the parent on the child, whereas a label of "0..*" indicates a one-to-many relationship. Finally a label of "1..*" indicates a one-to-many relationship where at least one child is required.

events is unimportant. Rather, the outputs of the neuron model must be capable of being represented using a single binary event, indicating the presence of absence of a spike.

4.2.3 Distribution of Neurons

When using multiple processes in a simulation run, the computational framework attempts to divide the work evenly across all available processes. This is accomplished primarily by distribution of the neurons across the grid of processes. Distribution is performed using a round-robin technique, where each neuron is assigned to a specific process with rank, `rank`, when `rank == i mod N`, where `i` is

the i-th neuron and `N` is the total number of processes in the communicator (i.e. the total number of processes involved in the MPI group). This distribution of neurons is naïve to the connection topology associated with the neurons as well as the neuron response properties. However, since the distribution of neurons is known *a priori*, construction of the structures required to describe the neural population can be performed in parallel on all processes in the parallel communicator.

4.2.4 Random Numbers

Many aspects of the computational framework rely on the use of random numbers. In particular, evaluation of spike times for Poisson LIF neurons (as described in Section 3.1.1) requires the use of a uniform pseudo-random number generator. In order to ensure that each of the processes involved in the computation are sufficiently randomized, a single process is used to define the seeds for every other process. A benefit of distributing seeds to every process is that it provides repeatability to the simulations, since only a single random seed need to be provided. If the user does not supply a random seed, a random seed is determined automatically using the current system time.

4.2.5 Connection Structure

The simulator contains several built-in connection topologies which can be used by specifying commandline options. In order to generalize to arbitrary connection topologies, however, the simulator also provides the ability to read in a connection structure from a file or pipe. The format of a single connection is shown below:

```
struct connection {
 uint32_t source; /* ID of source neuron */
 uint32_t destination; /* ID of destination neuron */
 REAL weight;
 REAL delay;
};
```

where `REAL` is a 4-byte floating point number and `uint32_t` is a 32-bit unsigned integer. Only positive delays are allowed for a connection. This connection structure allows connections between up to 2^{32} ($\approx 10^{10}$) neurons. The connection file is implemented as a binary file due to the file system cost associated with writing an ASCII equivalent file.

The storage of the connection topology represents a fundamental problem for the construction of a neural simulator. The problem is two fold: first, the underlying connection topology may require extensive amounts of memory to store. Second, since neurons are distributed across processors in a high performance environment, the connection structure must also be distributed.

In order to reduce the memory requirements associated with storing the connection topology, we make the assumption that a neural network can be *at most* fully connected. This implies that multiple connections with the same source and destination neurons are not allowed. This is a reasonable assumption given the duration of the postsynaptic current filters associated with the distribution of spikes as well as the characterization of neural spike as binary events. In the case where two connections exist from a single source to destination neuron with equivalent delays, the weights can be summed to provide equivalent input at the destination neuron without the cost of storing two connections. If the delays associated with the two connections are different, the weights of these connections can be summed as well. However, in this case, the delay associated with the equivalent connection would be the average of the two original temporal delays, provided each of the temporal delays is sufficiently smaller than the duration of the PSC filter.

In the case of a simulation which involves the use of multiple processes, connections are only stored in memory for the process where the destination neuron resides. Since the connection weight needs to be used to evaluate spiking currents at the destination neuron, it makes sense that the connection structure would only

reside with the process which is responsible for the destination neuron. For large populations, the connection structure represents the largest memory requirement of the simulation. For instance, a simulation of 100,000 neurons with 10% random connectivity requires approximately 19 GB of memory to store the connection topology. However, a simulation of this scale requires approximately 200 MB of memory to store the neuron parameters. Storage with the process that contains the destination neuron provides the ability to distribute this large memory requirement across all processes involved in the simulation.

For large networks, the number of spike events for a simulation can be very large. Therefore, performing a lookup of the weight associated with particular spike events for later application to the spiking current represents a potential performance bottleneck. In order to reduce the computational cost associated with spike application, the computational framework uses two joint data structures to store the connection information. First connections are stored in a splay tree, indexed by the source neuron id. Second, locally connected destination neuron identifiers are stored within the splay tree as an unrolled list.

A splay tree is a type of binary search tree which adjusts its structure based on access patterns. Binary search tree operations, such as insertions, removals, and searches are combined with a splay operation. This operation moves the accessed element from its original location in the binary search tree to the root of the tree. Since this splay operation is performed for every access to the splay tree, the more frequently used nodes will be located near the root of the tree; nodes which are not frequently access with be located at the leaves of the tree. For non-uniform access to the nodes, a splay tree can perform insertions, searches, and deletions in amortized $O(\log(n))$ time, where n is the number of elements in the tree. While the average time complexity is equivalent to classic binary search trees for these operations, the splay tree performs better than classic binary search trees for non-uniform access

patterns. However, since the splay search tree must satisfy the properties of binary search tree: 1) the left subtree contains nodes with keys that are strictly less than the node's key, 2) the right subtree contains nodes with keys that are strictly greater than the node's keys, and 3) the left and right subtrees also are also binary search trees, the performance of a splay tree can deteriorate to $O(n)$ time. This can occur when access to each of the nodes in the tree are performed in increasing order.

A list of the destination neurons for each source neuron are encapsulated in an unrolled list, contained in each element of the splay tree. The structure of an unrolled linked list is similar to a standard linked list, expect that multiple elements in the list are stored in the same node. A pictorial representation of this data structure is shown in Figure 4.3. This has several advantages over a standard linked list implementation. Primarily, the amount of memory required to store the entire list contents is reduced. In a typical linked list, a pointer to the location of the next list node in memory must be stored in each node. This pointer may require 4 or 8 bytes. Given the connection structure defined above, this pointer would account for a minimum of 4 bytes out of 20 total bytes (20% overhead). If the number of elements in each node is given by M, then an unrolled list need only store a pointer to the next node in the list N/M times, where N is the total number of elements in the list. In addition to requiring less overhead to store the connection topology, an unrolled list provides performance benefits if the total size of the elements in a node is less than the processor cache size (i.e. this significantly reduces the number of cache misses associated with a linear traversal of the linked list, since an entire node can be cached). The primary detriment of an unrolled list, however, is that memory can be wasted due to unfilled nodes. Given that the connection structure of the neural network does not change over the course of the simulation, this primary occurs in the final node of the unrolled list. This fragmentation can significantly affect memory when the number of neurons is large but the number of connections

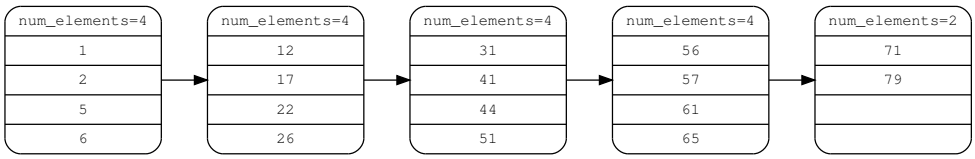


Figure 4.3: A pictorial diagram of an unrolled linked list, used to store synaptic connections to destination neurons who reside in memory on a particular process.

per neuron is small.

4.3 Execution

4.3.1 Distribution of Spikes

Since neurons may be divide across a number of independent processes, the communication of events (neural spikes) must occur in lock-step for each process. Therefore, at the end of every time step, a list of local neurons which posses a spike event are determined for every process. Each process then communicates the number of local events to every other process in the communicator. This allows the allocation of a single array which will eventually contain both the identifiers of the neurons which had events as well as the temporal location of the spike event relative to the current timestep. Following allocation of this data structure, a single collective MPI routine, `MPI_Allgratherv`, combines the independent list from all processes.

Once the list of spike events has been obtained, each of the processes must iterate over every potential source neuron in the event list to determine if there exist any locally connected destination neurons. Since this search must be performed for every event, the minimal computational cost associated with evaluating the distributed spikes is linearly related to the number of spike events. Therefore, networks which are “more active” (i.e. there is a higher number of spike events) will require more computational time in the application of the spikes. However, the

length of this list is limited by the number of neurons in the simulation, which places an upper bound on the total computational time associated distribution of spikes. If a connection to a local neuron is found by searching through the splay/unrolled list data structure, the connection weight is stored with the destination neuron, which will later be applied to the destination neuron's spiking current input, J^{spike}.

4.3.2 Evaluation of Currents

The computational framework calculates values for three separate currents for every neuron in the simulation. These currents correspond directly to the constituent elements of the total current, $J_j(t;\mathbf{S})$: the driving current associated with presentation of a stimulus, $J_j^d(t;\mathbf{S})$, spiking currents for modeled connected neurons, $J_j^{spike}(t)$, and a stimulus non-specific bias current, $J_j^{noise}(t)$.

Spiking Current

As described in Section 3.2, the post-synaptic current resulting from a spike is the convolution of a exponential neuron-specific post-synaptic current filter and the associated connection weight. A circular buffer for each neuron is used to store intermediate convolution results. In order to ensure that connections with associated delays are applied at the correct time, a delta queue is used to store this temporal information. A delta queue is a form of singly linked list in which successive elements in the list represent spike events that will be applied a given number of timesteps after the previous link. Insertions into the delta queue require $O(n)$ time. However, an advancement of the delta queue can be performed in constant time since only the delay associated with the first item in the queue needs to be decremented. Since the application of a spike for a particular neuron, in general, occurs less frequently than a timestep (advancement of the delta queue),

there is significant computational benefit to using a delta queue as opposed to a classic queue structure.

Stimulus Response Evaluation

The driving current input to each neuron is determined by evaluating the stimulus response profile given an input stimulus. The stimulus is most commonly read from a memory-mapped file. Several built-in stimulus response profiles are provided for the user, including Gaussian, von Mises, linear, cosine, and sigmoidal. If a leaky integrate-and-fire neuron in use in the simulation, the user is required to provide both the values of α and β, where α converts the normalized input to units of current, and β denotes the relative input of the driving current, $J_j^d(t;\mathbf{S})$, to the spiking current, $J_j^{spike}(t)$.

4.3.3 Monitors

The required output from simulations can differ significantly depending on the research question which the simulation is used to address. Therefore, numerous neuron inputs, states, and outputs are capable of being monitored over the course of the simulation. We refer to these time varying outputs generically as "monitors." The cummulative output of the neuron model, corresponding to the number of spike events generated by that neuron, are always provided to the user. These spike counts can be used to ensure that the network is performing as expected. For instance, the number of spikes plotted against the preferred direction of the neurons in the population can provided an estimate of the population response to a provided stimulus. The spike counts can also be coarsely compared against available neurophysiology to ensure that responses are physiologically plausible.

Additional monitors can be used to obtain fine grained information regarding the dynamics of the underlying neurons in the simulation. These monitors, such as

the various synaptic current inputs, membrane voltages, and spike trains, provide information at every time step of the simulation. Therefore, the majority of these monitors can be represented as a $N \times Nt$ matrix, where N is the number of monitored neurons and Nt is the total number of timesteps in the simulation ($T/\Delta t$). For simulations in which a large number of neuron parameters must be monitored or for simulations with a long duration, the size of the monitors can grow large. Therefore, the monitors are designed to overcome several of the limitations associated with performing disk I/O on large files.

First, monitors are written in a binary file format. This has the advantage of significantly reducing the required disk size compared with ASCII text output. In addition to being written in binary, the structure of the monitors also allows ordered writes over the duration of the simulation (i.e. binary seeking is not required to write out the monitors). Second, monitor output is buffered in memory to reduce the number of disk commits. Third, monitors are written out on a per-process basis, without the need for an underlying shared file system. The organization of the monitors allows for combination of per-process monitors into a global monitor after the simulation has completed. This global monitor can be created sequentially, without the need to perform binary seeks. Finally, per-process records can be written using a streaming compression library, since monitor output is written sequentially.

The global monitor consists of three different pieces. At the very bottom of the binary file, a binary footer is used to store the size of the binary data types and the number of records in the file:

```
struct io_size_footer {
 uint32_t num_records; /* Number of records */
 uint8_t sizeof_double; /* Size of a double */
 uint8_t sizeof_float; /* Size of a float */
 uint8_t sizeof_long; /* Size of a long */
 uint8_t sizeof_int; /* Size of an integer */
 uint8_t sizeof_short; /* Size of a short */
```

```
 uint8_t sizeof_char; /* Size of a char */
 uint8_t version; /* Revision of the binary footer */
 uint8_t checksum; /* File checksum */
};
```

The majority of the elements in the "size footer" provide the ability to read and write monitor files on two different machine architectures (i.e. 32 versus 64 bit systems). Multi-byte elements are always written in little-endian format. Since the size footer resides at the end of the file and always has a known size, it is straightforward to obtain the number of records in the file.

Immediately preceding the size footer in the binary file is a series of record footers. There are as many record footers in the file as there are records, noted by the `num_records` field in the size footer. Each record footer notes the location and matrix dimensions of a record in the file:

```
struct io_record_footer {
 uint64_t offset; /* Record offset from SEEK_SET (in bytes) */
 uint64_t length; /* Length (in bytes) of the record */
 uint32_t id; /* Record Id */
 uint32_t associated_record; /* Associated records */
 uint32_t x; /* Length of dimension #1 */
 uint32_t y; /* Length of dimension #2 */
 uint32_t z; /* Length of dimension #3 */
 uint8_t compressed; /* Record is compressed ? */
 uint8_t interleaved; /* Third dimension is interleaved with x and y */
 uint8_t element_size; /* Size of each element */
 uint8_t element_type; /* Float or integer */
};
```

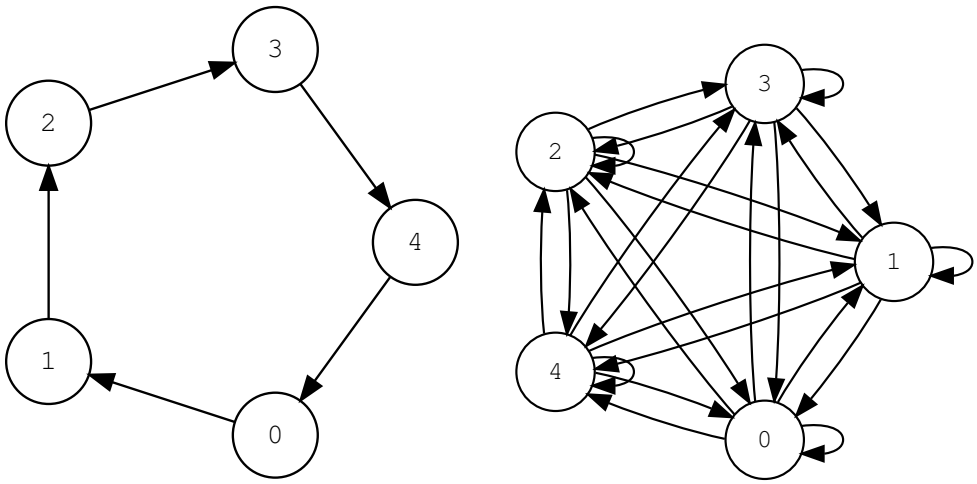
The `offset` field provides the location of the start of the record in the preceding binary file. The `length` field is used to define the number of bytes in the binary file occupied by the record in the file. When the file is uncompressed, this number is equal to `x` * `y` * `z` * `element_size`. However, if the record is compressed, the length of the record will be smaller than than this calculated quantity. Compression of the records is performed during the simulation run on each process by passing the binary data through the zlib compression library.

The size/record footer is used rather than a header to allow combination of per-process monitors using a series of sequential writes. Since a new `io_record_footer` needs to be added for every additional record in the file, using a header is impractical since insertions of this additional record footer would require the following binary data to shifted.

4.4 Performance Evaluation

The performance of the computational framework was evaluated as the time required to complete a simulation, measured using the Unix `time` command. The framework was run in a high performance environment using one, two, four, eight, and sixteen processors. Simulation runs which required eight or fewer processors were performed using a single octcore system. Simulations which used sixteen processors were divided across two physical systems. The time required to complete each function call in the computational framework was also determined for each simulation run using the GNU profiler. All simulations were 10 seconds in duration, using quarter millisecond timesteps. All reported results represents the mean time of 5 identical simulations. Input was provided to a single neuron in the network and synaptic weights were scaled such that a spike of the presynaptic neuron caused the postsynaptic neuron to spike at the following timestep. This results in an overdriven network in which the maximum number of spikes occurs given the assigned connection topology. All neuron characteristic were stereotyped and spiking events were evaluated using the deterministic LIF neuron model.

Two different network topologies were used to test the performance of the computational framework (Figure 4.4): ring and randomly connected. The ring topology has an equal number of neurons and connections. This network features constant communication overhead since at least one neuron will fire at every



(a) (b)

Figure 4.4: Topologies used to evaluate the performance of the computational framework. a) A ring topology in which neurons with sequential ID's are connected. b) A randomly connected network topology with a 100% probability of connection (fully connected). Neurons are connected to all other simulated neurons, including themselves.

timestep. Ring tests were performed for a variety of population/connection sizes from 100 to one million neurons. We limited the size of the population to one million neurons so that simulations would complete in a reasonable amout of time when evaluated using a single processor.

Various randomly connected neural network configurations were used. A fully connected network, in which all neural units are connected to all other units, is created when there is a 100% connection probability. This network has periods of high communication overhead followed by periods of no required communication. Since all neurons are connected to each other and neuron properties are stereotyped, the entire network spikes immediately after the first neuron spikes. This requires distribution of a spike for every neuron in the simulation. However, subsequent timesteps require no distribution of spikes since the entire network enters the absolute refractory period.

Additional randomly connected network topologies were used to evaluate the

performance of the computational framework with respect to the number of neurons and synapses. A fixed population size of 1,000 neurons was used to determine the effects of the network topology on the simulation time, where the percentage connectivity was systematically varied from 1% to 100% (fully connected, one million synapses). In a separate series of simulations, the number of synapses was fixed at 500,000 while the number of neurons was varied from 1,000 to one million neurons.

All of the timing simulations were performed on Marquette's Père cluster (NSF awards OCI-0923037 and CBET-0521602). Each node featured two quad-core Intel Nehalem X5550 processors, resulting in a total of 8 cores for each machine. The processors are clocked at 2.67GHz with 8MB cache per core. Hyperthreading on each node was disabled. Processor affinity was disabled. Each node had 24GB of dedicated system memory. All nodes featured RedHat Enterprise Linux with kernel 2.6.18-128. Communication between processes located on separate nodes used a 4X DDR infiniband interface, with a theoretical transfer limit of 20 Gbit/s.

4.4.1 Results

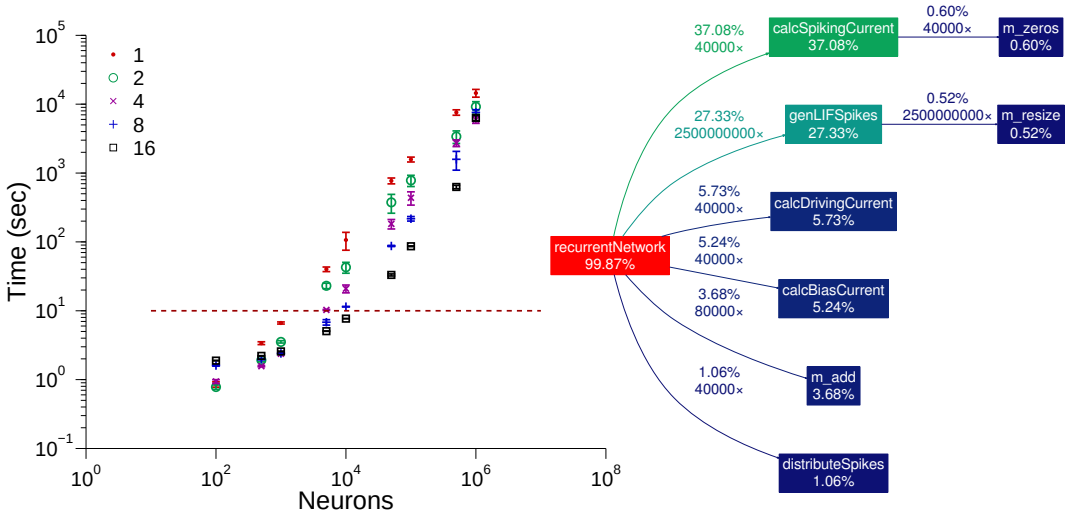
The results for the ring topology are shown in Figure 4.5 for population sizes ranging from one hundred to one million neurons. There is a linear relationship between the simulation time and population size. This linear relationship is valid for population sizes that are greater than 10,000 neurons. Populations that are smaller than 10,000 neurons, particularly in the case of 16 processors, saturate at a lower bound on the simulation time (<10 seconds). This lower bound is likely due to the disk I/O requirements involved in setting up the network topology and reading the provided stimulus from a file (i.e. initialization phase). In addition, there is communication overhead between two physical systems for the 16 processor cases. This communication overhead is the likely reason for the greater time required to

complete simulations between 100 and 500 neurons using 16 processors, compared with the single node jobs. Profiling of the computational framework for this topology shows that the majority of the time (75.38%) is spent calculating the current input (`calcSpikingCurrent`, `calcDrivingCurrent`, and `calcBiasCurrent`) and spiking output (`genLIFSpikes`) to the LIF model. The communication overhead associated with distributing the spikes is small for the ring topology since the number of spikes is small, accounting for 1.06% of the total simulation time.

The red horizontal line in Figure 4.5(a) denotes realtime performance (10 simulation seconds = 10 wallclock seconds). Simulations that appear below this line were completed faster than realtime. The results suggest that population sizes up to 50,000 neurons, when simulated using 16 processors, can be completed with faster than real time performance. The size of the population which can be completed in realtime can be increased, to a point, by increasing the number of processes devoted to the computation. However, as the number of processes in the communicator increases, so does the cost associated with communication as well as the overhead associated with initially constructing the network.

Timing results for the fully connected network topology are shown in Figure 4.6. Since the network is fully connected, as the number of neurons increases, the number of synapses increases with the square of the neurons. Therefore, a population of 10,000 neurons features one hundred million (100,000,000) synapses. Population sizes up to 1,000 neurons using 16 processors can be simulated in realtime. Similar to the ring topology, Figure 4.6(a) shows a linear increase in the simulation time as the number of neurons increases. This linear trend exists in all cases except for 16 process simulations at small population sizes. These simulations are dominated by communication overhead rather than computation.

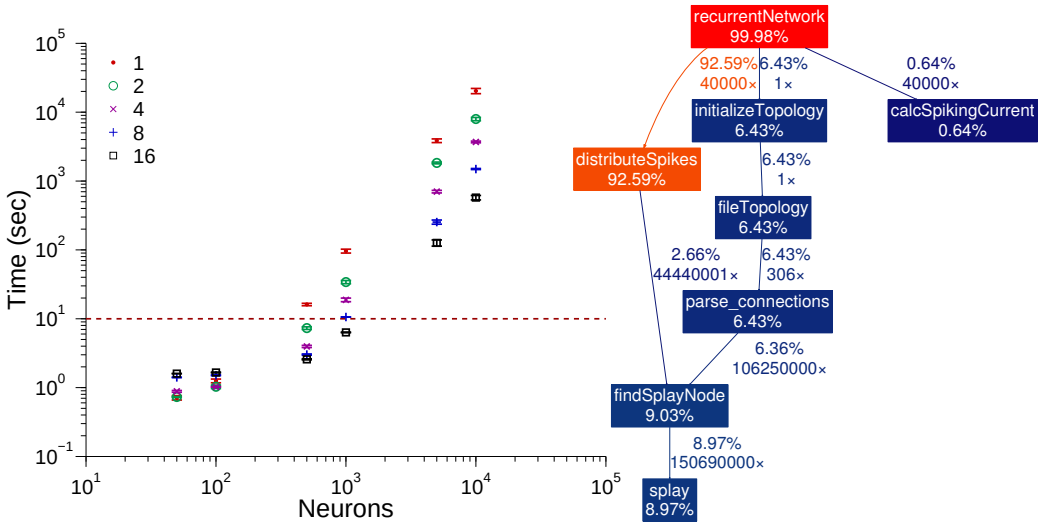
Unlike the profile of the overdriven ring presented in Figure 4.5(b), the performance of the fully connected profile is dominated by distribution of the spikes



(a) (b)

Figure 4.5: Performance of the computational framework for an overdriven ring topology with population sizes ranging from 100 to one million neurons. a) The time required to complete the simulation as a function of the neural population size. Symbols refer to the number of processors used for each simulation, ranging from 1 to 16. Error bars indicate standard deviation. The red horizontal line in the graph indicates realtime performance. b) A diagram of the simulator functions which contributed most to the overall simulation time. Percentages refer to the percent of total simulation time spent in the referenced function and its children. The number of times each function was called is also provided (numbers with a trailing "x"). Profiling was performed for a population of one million neurons using 16 processors.

at each timestep (accounting for 92.59% of the total simulation time). As noted in Section 4.3.1, the distribution of spikes involves a collective communication operation, which places a copy of the neuron ids that spiked on each process in the communicator. Then, each process must perform a lookup using the connection splay tree to determine if a connection between the source neuron and any local neurons exist. Since every neuron in the network spikes at the same timestep immediately following its refractory period, this overdriven fully connected network is dominated by the linear processing of all of the spikes that arrive after each

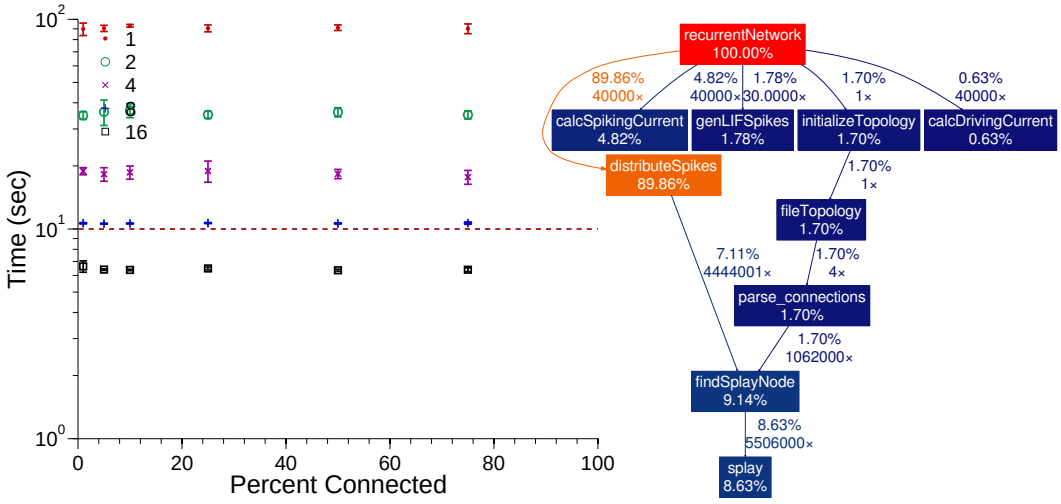


(a) (b)

Figure 4.6: Performance of the computational framework for an overdriven fully connected topology with population sizes ranging from 50 to 10,000 neurons. a) Time required to complete the simulation as a function of population size. Symbols are the same as Figure 4.5. b) Timing profile of the computational framework for a population of 10,000 fully connected neurons using 16 processes.

refractory period. Increasing the number of processes in the communicator can reduce this overhead since the length of the connection lists is related to the number of local neurons. Increasing the number of processes reduces the number of local neurons and, on average, reduces the time associated with processing spike events proportionally. For instance, 16 processors can simulate 10 seconds of a fully connected ring consisting of 10,000 neurons in 595.38±37.17 seconds. An identical network simulated using 32 processors completes in approximately half the time (311.67±7.46s). Likewise, a 64 process communicator again reduces the simulation time by approximately half (165.06±14.32s).

Figure 4.7 shows the effects of varying the number of synapses while keeping the number of neurons constant. A population size of 1,000 neurons was used to characterize performance with varying percentages of random connectivity between

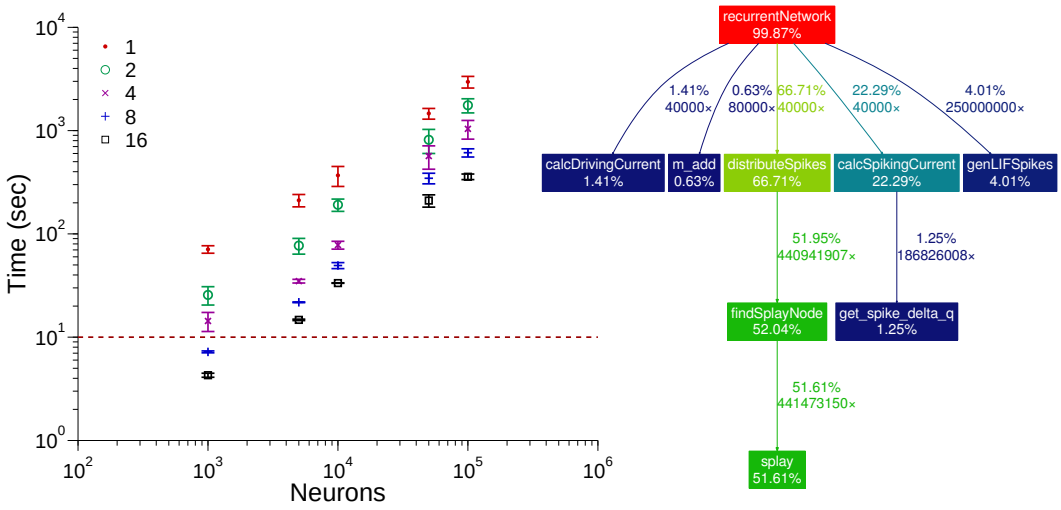


(a) (b)

Figure 4.7: Performance of the computational framework for an overdriven network consisting of 1,000 neurons. The percent connectivity was varied from 1% to 75%. a) Time required to complete the 10 second simulation as a function of percent random connectivity. b) Timing profile of the computational framework for a population of 1,000 neurons featuring 75% connectivity using 16 MPI processes.

one and 75%. A random connectivity of 1% consists of approximately 10,000 synapses whereas a 75% connected network has approximately 750,000 synapses. Figure 4.7 shows that the performance of the simulator is approximately constant for varying levels of connectivity. Profiling of the computational framework (Figure 4.7(b)) is very similar to the fully connected case above. Again, the speed of the computational framework is limited by the distribution and analysis of spike events (accounting for 89.86% of the total simulation time). Even at 1% connectivity, it is likely that the network dynamics are comparable to the fully connected case: all of the neurons in the network spike immediate after leaving their absolute refractory period.

Figure 4.8 shows the results of varying the number of neurons in the simulation while maintaining 500,000 synapses for all population sizes. The results



(a) (b)

Figure 4.8: The performance of the computational framework as a function of population size. The number of synapses in each simulation was held constant at 500,000. a) Time required to complete at 10 second simulation as a function of the population size independent of the number of synapses. b) Timing profile of the computational framework for a population of five hundred thousand neurons with 500,000 connections using 16 processes.

show that the total simulation time increases linearly with population size. The profiling results (Figure 4.8(b)) show that distribution and processing of spike events constitutes the majority of the simulation time (66.71%). However, unlike the fully and randomly connected networks presented above, the majority of the processing time for these events is spent performing the splay function (51.61% of the total simulation time). These timing results suggest that the splay lookup (`findSplayNode`), which determines if a destination neuron exists locally, is the most costly operation for this network. This is due to the fact that there are a large number of neurons per process, but the number of connections to each of these neurons is small (e.g. a population of 100,000 neurons is only 0.005% connected).

4.4.2 Discussion

The performance of the computational framework in a high performance computing environment is tied to the dynamics of the underlying network architecture. In all cases, the majority of the simulation time can be tied to the distribution, analysis, and subsequent application of spikes. The simulation time is linearly tied to the number of spike events. However, since the network can be, at most, fully connected, the upper limit on this relationship is determined by the number of simulated neurons. For instance, in the case of the randomly connected 1,000 neuron network presented in Figure 4.7, the time required to complete the simulation remains constant for all tested connectivity percentages. This constant time is due to the fact that a 1% connected network and a 75% connected network have approximately the same network dynamics when synaptic weights are large enough to evoke a postsynaptic spike whenever the presynaptic neuron spikes. In the case where the network is not overdriven, there would be a supralinear relationship between the simulation time and the number of neurons (since the number of spike events at each timestep will be significantly less than the total number of neurons).

The use of the Message Passing Interface provides several advantages over a single serial process. Provided the simulated network is large enough to overcome the communication and setup overhead associated with multiprocess simulations, the speed of the computational framework increases as additional processes are added to the communicator. This is most clearly evident in the fully connected network topology where doubling the number of MPI processes effectively halves the required simulation time. In addition, the use of multiple processes allows the distribution of memory costs across a number of physical systems. This is advantageous when either the number of neurons or synapses is large.

5 CASE STUDIES

In this chapter, we discuss two different models implemented using the described modeling and computational frameworks from Chapters 3 and 4. While previous examples have shown that the computational and modeling frameworks function as expected, the case studies in this chapter are intended to show the versatility of these frameworks when attempting to address novel scientific questions.

We first describe a model of the hemodynamic BOLD response for populations of spiking neurons. This model relates population activity, in the form of postsynaptic currents, to the BOLD response, which can be measured experimentally. This extension to the modeling framework proposed in Chapter 3 explicitly bridges the gap between spike-based neuron models and mean-field models, which have previously been used to investigate hemodynamic responses (Corchs and Deco, 2002; Bojak et al., 2010). We use the model to make experimental predictions relating functional changes among the interconnections between neurons to large-scale population responses capable of being measured via conventional imaging techniques, specifically fMRI.

Building on the hemodynamic response model, we also describe a model of visual motion processing, which encapsulates the neurophysiological properties associated with neurons in the middle temporal and medial superior temporal cortex. We validate this model by demonstrating that simulation results are consistent with neurophysiological responses in these areas. Using a Hebbian-style learning paradigm, we simulate the hemodynamic response changes that would accompany learning in these areas. Results will provide explicit predictions regarding changes in the hemodynamic response during perceptual learning, that

can be used to probe the sites of plasticity during learning (Wakde, 2011). In addition, this model allows investigation of the temporal dynamics associated with processing visual motion in areas MT and MST.

Several recent studies have investigated the relationship between the magnitude of the hemodynamic signal and the population rate activity derived from single and multi-unit electrode recordings (Heeger et al., 1999; 2000; Rees et al., 2000). The studies indicate that the magnitude of the BOLD response is directly proportional to the population rate response. While the constants of proportionality between the hemodynamic magnitude and the mean firing rate vary between studies, a positive linear relationship has been reported across studies.

Recently, Logothetis (2008) notes that changes in the magnitude of the measured fMRI signals may be due to neural activity that is not related to the mean firing rate across the population. Instead, Logothetis (2008) hypothesized that changes in the BOLD signal may be the result of changes in the balance of excitatory and inhibitory activity rather than mean firing rate. Therefore, increases in the magnitude of the hemodynamic response relative to baseline may be observed even if the aggregate firing rate of a population remains constant or decreases.

Elucidating the relationship between changes in neural processing and structure and the hemodyanmic response is required to construct computationally tractable cortical models. Numerous studies have used mean-field models to construct forward models of the hemodynamic response across cortical areas (Bojak et al., 2010; Corchs and Deco, 2002; Coombes, 2010, for a review). Models which simulate forward hemodynamic properties using the aggregate rate statistics of the underlying population may not provide accurate results if, as Logothetis (2008) suggests, hemodynamic responses are tied to the balance of excitation and inhibition within cortical subpopulations. Both models presented in this chapter seek to elucidate the relationship between the aggregate mean firing rate across a population

and the simulated hemodynamic response for large populations of spiking neurons.

5.1 Single-Area Hemodynamic Response Model

In this section, we outline a model for simulating the hemodynamic response of a cortical area. We model a cortical area using a populations of pyramidal cells and interneurons as a first order approximation to the processing that occurs within an area. We first outline background information regarding the link between individual spiking neurons, local field potentials, and the hemodynamic response. This section builds on the background information of Section 2.4, providing a quantitative link between neuron postsynaptic potentials and the hemodynamic response.

5.1.1 Model Background

As noted in Section 2.4, numerous studies have investigated the correlations between single unit activity (SUA), multi-unit activity (MUA), local field potentials (LFPs), and the BOLD response. Logothetis et al. (2001) presented rotating checkerboard patterns to anaesthetized monkeys while simultaneously recording electrophysiologic and fMRI signals. Single and multi-unit activity was determined by bandpass filtering the recorded electrode signals using a zero phase filter between 300 and 3,000 Hz and subsequently low pass filtering at 150 Hz to obtain an envelope of the SUA and MUA. The SUA and MUA are typically associated with the spiking activity of neurons within 300-400 micrometers of the placed electrode. The magnitude of the LFP signal is typically associated with the weighted average of synchronized input signals (postsynaptic currents) of a neurons located within 1-3 millimeters of the electrode tip, and was determined by bandpass filtering the acquired data between 10 and 130 Hz using a 36 db oct^{-1} zero phase filter.

Using a least-squares regression approach, Logothetis et al. (2001) found that

the average LFP response of a spatially localized cortical area gave a better estimate of the measured BOLD contrast than either SUA or MUA. The greater correspondence of the LFP to the hemodynamic response is in general agreement with the theory that a greater percentage of the hemodynamic response is related to energetically demanding synaptic activity rather than the spiking output of a population (Pellerin and Magistretti, 1994). Logothetis et al. (2001) also noted that spiking activity will usually correlate with the pre- and postsynaptic current inputs, although this may not be the case across all cortical regions.

Lippert et al. (2010) extended the results of Logothetis and colleagues by investigating the correlation between LFP signal amplitude in distinct frequency bands and the hemodynamic response. In their study, electrophysical recordings of area MT were obtained while simultaneously recording the hemodynamic response to drifting sine-wave gratings. Electrophysiological recordings were first filtering into low and high frequency regimes, corresponding to LFP (fourth order low pass filter, 100 Hz) and MUA regions (fourth order high pass filter, 400 Hz). Different frequency bands in the LFP region where obtained by applying a series of bandpass finite impulse response (FIR) filters with frequency ranges of 4-8 Hz, 12-40 Hz, and 40-60 Hz. In agreement with Logothetis et al. (2001), Lippert noted that LFP bands between 12 and 60 Hz provide a better correlation between the hemodynamic response than did MUA. Using a general linear model which allowed investigation of the variance accounted for by each frequency band, Lippert and colleagues noted that the 40 to 60 Hz band most accurately predicated the BOLD signal in areas with stimulus-specific responses (i.e. regions in which neuronal firing rates were sensitive to the direction of stimulus presentation).

Together, the results of Lippert et al. (2010) and Logothetis et al. (2001) suggest that the hemodynamic response for a spatially localized neural population can be accurately described by convolution of a hemodynamic response function

with the power of the LFP in the 40-60 Hz band. Several studies have proposed models which relate the postsynaptic current of spiking neurons to the LFP (Mazzoni et al., 2008; Rasch et al., 2009; Mattia et al., 2010). Others have used the average membrane potential of a population of neurons as an estimate of the LFP (Ursino and La Cara, 2006), although neurophysiological studies generally note the correspondence between LFPs and the aggregate postsynaptic currents (Logothetis, 2003).

In multi-compartmental neuron models, the LFP can be determined at every point in space by computing the weighted sum of extracellular potentials over a large number of compartments (Pettersen et al., 2008). As a computational simplification, here we use a point-source neuron model, similar to Mazzoni et al. (2008), which does not attempt to incorporate the spatial organization of the underlying cortical areas. Instead, we model LFP generation at a point assuming that each neuron acts as a spatial point source, contributing equally to the overall LFP amplitude within a local region (< 3 mm) of cortex. In order to account for the fact that pyramidal cells account for the majority of the hemodynamic signal (Leung, 1991), we assume that current flows into a cell through apical excitatory synapses and flows out through basal inhibitory connections(Mazzoni et al., 2008). Therefore, the magnitude of the LFP for neurons without spatial extent can be modeled as the sum of AMPA and GABA currents, which are incident upon the population of pyramidal cells. Mazzoni et al. (2008) noted that multiple linear combinations of unsigned AMPA and GABA currents can provide similar LFP results, therefore we resort to the simplest of these models in which AMPA and GABA currents contribute equally to the overall magnitude of the LFP.

Figure 5.1 illustrates the extension of the modeling framework proposed in Chapter 3 to provide an estimate of neurovascular coupling. Using neurons modeled as a spatial point-source, the magnitude of the local field potential can be estimated

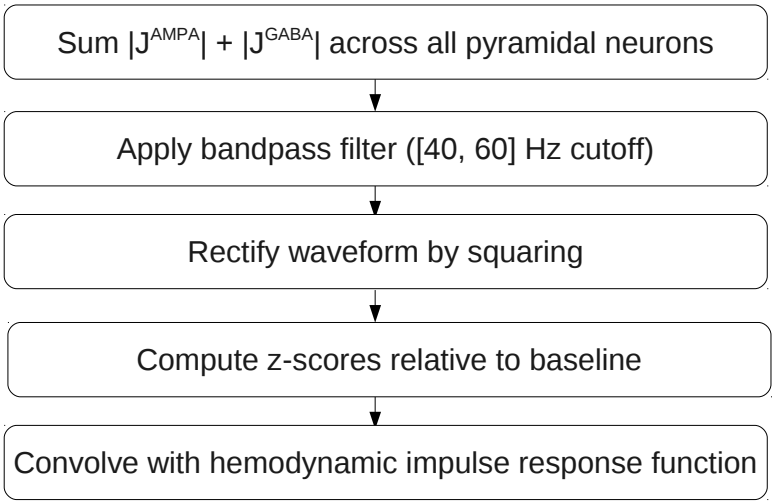


Figure 5.1: Mathematical model relating the postsynaptic current to the hemodynamic response. The postsynaptic current is first related to the magnitude of the local field potential (LFP) by summation of the absolute magnitudes of AMPA and GABA currents incident on pyramidal neurons. The magnitude of the LFP frequencies between 40 and 60 Hz is then related to the BOLD signal via a gamma response function.

as the sum of the unsigned AMPA and GABA currents incident on the pyramidal cells. The AMPA and GABA currents correspond to the excitatory and inhibitory portions of the spiking input current, respectively,

$$J_j^{spike}(t) = J_j^{AMPA} + J_j^{GABA} \ . \tag{5.1}$$

The voltage measured via an electrode probe, $V^E(t)$, is then related to sum of the AMPA and GABA currents,

$$V^E(t) = \sum_j^N R_j^E \left[|J_j^{AMPA}| + |J_j^{GABA}| \right] \ , \tag{5.2}$$

where R_j^E is the extracellular resistance between the j-th neuron and the voltage measured at the electrode tip.

Consistent with Lippert et al. (2010), we use the magnitude of the LFP in the 40 to 60 Hz band to predict the magnitude of the BOLD signal for the modeled

population. We use the hemodyanmic impulse response function, $g(t)$, as described by Boynton et al. (1996), to relate these two quantities:

$$g(t-\epsilon) = \frac{(t/\tau)^{n-1} e^{-(1/\tau)}}{\tau(n-1)!} \ , \tag{5.3}$$

where $n = 3$, $\tau = 1.25$ s, and $\epsilon = 2.5$ s.

5.1.2 Methods

We implemented a model which consisted of a single generic cortical area. Conceptually the area was divided into two distinct populations, consisting of pyramidal and inhibitory interneurons, respectively. The overall structure of the model, shown in Figure 5.2, is computationally similar to that proposed by Mazzoni et al. (2008) for LFP generation. Pyramidal neurons projected purely excitatory (AMPAergic) synaptic connections. These AMPAergic connections existed laterally, both among neurons in the pyramidal population and in the pyramidal projections to the inhibitory interneuron population. The interneurons projected purely inhibitory (GABAergic) synaptic connections to other neurons in the interneuron population as well as to pyramidal neurons. Together, the pyramidal and inhibitory interneuron populations composed a single laterally connected cortical area.

The overall model parameters were similar to those described in the Section 3.3.1. The model consisted of 5,000 pyramidal and 5,000 inhibitory interneurons. We used LIF neurons with Poisson distributed spike trains as the encoding model in both neural populations. Each neuron was assigned a Gaussian stimulus response profile, with preferred directions uniformly distributed throughout a [0, 2π] polar space. Standard deviations in the Gaussian profile were distributed from [$\pi/8$, $\pi/4$] radians for pyramidal cells and [$\pi/4$, $\pi/2$] for inhibitory neurons. We simulated a total of 10 seconds of neuron responses, in which no stimulus was

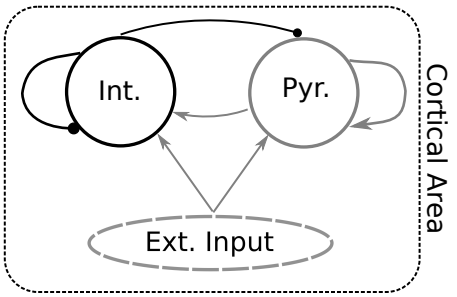


Figure 5.2: Neural network model used to characterize the hemodynamic response of a single cortical area. The model was similar in structure to a model of LFP generation proposed by Mazzoni et al. (2008). External input via a driving current was supplied to both the interneuron and pyramidal neuron populations. Lateral connections existed among neurons in both the pyramidal and inhibitory interneuron populations. Additional connections linked the pyramidal and inhibitory populations.

supplied for the first 5 seconds, allowing neurons to fire at their background firing rates. The remaining 5 seconds of simulation time featured a stimulus located at π radians.

Neurons were assigned absolute refractory periods (τ_j^{ref}) chosen randomly from a uniform distribution between 2 and 5 ms, and RC membrane time constants were assigned randomly from 10 and 30 milliseconds. Neuron background responses were assigned randomly from 2 to 4 Hz for pyramidal cells and from 4 to 8 Hz for interneurons. All neurons were assigned maximum responses drawn from a uniform distribution between 40 and 80 Hz. The time constant associated with the postsynaptic current filters (Equation 2.11) were assigned randomly from 4 to 6 milliseconds.

A connection between any pair of neurons was established randomly, with a 60% probability of a synapse. Therefore, each neuron featured an average of 6,000 incident connections, with equal percentages of GABAergic and AMPAergic synapses. Synaptic weighting followed a series of Gaussian distributions, whose standard deviation was matched to the standard deviation of the destination neuron's stimulus response profile. GABAeric connections, in which the i-th source

neuron resided in the inhibitory interneuron population, were characterized by

$$w_{ij} = -A_j^{GABA} \exp\left[\frac{-\min(S_i^{pref} - S_j^{pref})^2}{2\sigma_j^2}\right], \tag{5.4}$$

whose amplitude is characterized by A_j^{GABA}. Connections from pyramidal neurons followed a similar Gaussian profile, with a different amplitude coefficient, A_j^{AMPA}. Rather than equalizing the background input to each neuron by adding an offset to each weight, as described in Section 3.2.1, we instead modified the amplitude coefficients, A^{AMPA} and A^{GABA}. Performing this multiplicative manipulation of the weight profile ensured that all AMPAergic connections remained positive and GABAergic connections were assigned a negative weight. The synaptic weights were then modified to elicit maximum responses as described in Chapter 3. The source code use to specify model parameters can be found in Appendix 7.4.

Unlike the models described in Chapter 3 where spiking activity of the presynaptic neurons was "instantly" incorporated into the postsynaptic neuron's membrane current at the next time step, the current model incorporated synaptic delays between neurons. The delays simulated the transmission delay of action potentials down the presynaptic neuron axon, and were assigned randomly between 0.25 and 1 milliseconds for excitatory connections and between 0.25 and 3 milliseconds for inhibitory connections. While these synaptic delays do not affect the steady-state responses of the neuron, they may play a role in high frequency neural network dynamics.

The hemodynamic response for a cortical region, modeled as a spatial point-source, was found by summing the AMPA and GABA currents for the pyramidal cells. The resulting signal was bandpass filtered with corner frequencies at 40 and 60 Hz (zero phase, 36 db oct^{-1}), and squared to obtain the net current magnitude. The standard deviation of the signal was determined for baseline

stimulus conditions. Z-scores were computed for every point in the temporal sequence, such that the resulting waveform related signal intensity in units of standard deviations (SD) relative to background. The time-varying hemodynamic response function was found by convolving the z-scored signal with the hemodynamic impulse response function. The maximum value of the convolved signal is referred to as the "peak hemodynamic response."

In order to quantify the level of activation of the neural population due to the presentation of the stimulus, we define an additional metric: the aggregate mean firing rate (AMFR), $\bar{r}$. In many mean-field models, neural populations may be replaced by an "equivalent neural unit" which describes the mean firing rate activity for the entire population. The AMFR represents the rate response of this equivalent neural unit, and can be defined as

$$\bar{r} = \frac{1}{NT} \sum_{j}^{N} \int_{0}^{T} a_j(t; \mathbf{S}) dt \ , \tag{5.5}$$

where a_j is the spike train of the j-th neuron within a population of N neurons.

Hebbian-style Learning

To determine the effects of synaptic plasticity on the magnitude of the hemodynamic response, we simulated Hebbian-style learning by *post hoc* modification of the synaptic weights. An estimate of each neuron's rate response to the stimulus, R_j^{est}, was determined by twenty successive presentations of a stimulus located at π radians. Using this stimulus-specific rate estimate, synaptic weights were modified using a variation of the Hebbian learning rule,

$$w'_{ij} = w_{ij} + \Delta w_{ij} \ , \tag{5.6}$$

where

$$\Delta w_{ij} = \zeta w_{ij} \frac{R_j^{est} R_i^{est}}{R_j^{max} R_i^{max}} \ , \tag{5.7}$$

in which ζ can be interpreted as the learning rate in Hebb's rule. However, since weight modification occurs only one and is not tied to a specific time scale, we use the term "learning coefficient" to avoid confusion. When $\zeta = 0$, no change in the synaptic weights occurs. If $\zeta > 0$ and the source and destination neurons fire near their maximum responses, then the synaptic weight is increased.

5.1.3 Results

Figure 5.3 shows the average stimulus-specific response of the pyramidal neurons to twenty successive presentations of a stimulus at π radians. The average rate responses of 1,000 randomly chosen pyramidal neurons is shown, normalized to their respective maximum responses, R_j^{max} for three different learning coefficientss ($\zeta = 0, 1, 2$). In the case where $\zeta = 0$, responses peaked at a value of 1 ($R_j = R_j^{max}$) for preferred stimuli near π radians. As the value of ζ increased, the peak of the curve also increased.

Figure 5.4 shows both the aggregate mean firing rate and the magnitude of the hemodynamic response, relative to baseline BOLD response. The AMFR of the population increased linearly with the learning coefficient, from a baseline of 22 spikes/sec to 38 spikes/sec when $\zeta = 2$, ($\bar{r} = 7.18\zeta + 22.98$, $R^2 = 0.98$). The magnitude of the hemodynamic response (in units of standard deviation) decreased linearly as a function of the learning coefficient, with a magnitude that was well-characterized by a linear relationship with a negative slope ($-2.36\zeta + 4.94$, $R^2 = 0.72$). The decrease in the BOLD response with increased plasticity was significant, $F(1, 39) = 101.37, p < 0.01$.

Figure 5.5 shows the change in the standard deviation of the baseline signal

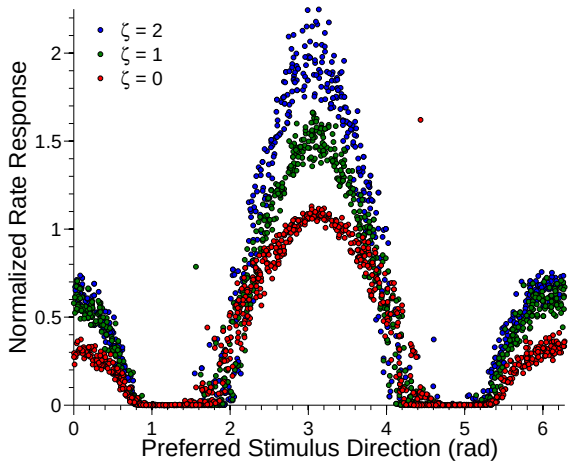


Figure 5.3: Stimulus-specific neuron responses for a population of pyramidal neurons as a function of preferred stimulus direction for learning coefficients, ζ, of 0, 1, and 2. One thousand randomly chosen neuron responses are shown, averaged over twenty stimulus repetitions. The rate response was normalized to the maximum response, R_j^{max} of each neuron.

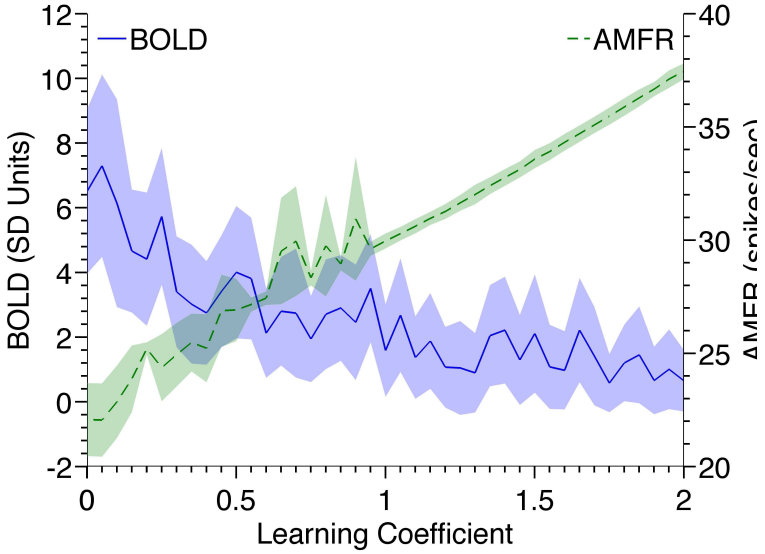


Figure 5.4: Peak hemodynamic response for a single interconnected neural network composed of 5,000 pyramidal cells and 5,000 interneuron. The peak hemodynamic response relative to baseline is plotted on the left axis (solid line) for a range of learning coefficients, ζ. The aggregate mean firing rate (AMFR) of the pyramidal population is plotted on the right axis (dashed line). Shaded regions denote ± 1 standard error.

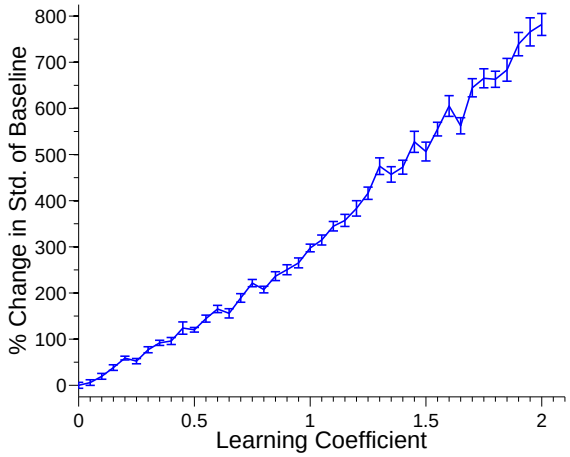


Figure 5.5: Change in the standard deviation of the baseline signal for a single-area neural network. The percentage change in the standard deviation of the baseline signal relative to a learning coefficient value of zero is shown as a function of Hebbian-style learning. The standard deviation of the signal was computed over the first five seconds of each simulation. Error bars indicate ±1 standard deviation.

as a function of the learning coefficient. The standard deviation of the baseline response of the population within the 40-60 Hz bandwidth consistently increases as the learning coefficient increases. In addition, as the learning rate increases, so does the standard deviation of the measured change in baseline activity (i.e. the magnitude of the error bars increase as the learning coefficient increases).

5.2 Visual Motion Processing Model

Extending the single-area model described in Section 5.1, we constructed a model of two prominent visual motion processing areas which we use to investigate the effects of task-dependent learning on the hemodynamic response. In the following section, we provide background material relating to, and previous models of the visual motion processing pathway. Together, this information is used to construct a spike-based model of visual motion processing for complex motions, whose steady state responses align well with available neurophysiology and pyschophysical literature.

5.2.1 Model Background

The movement of an individual through the environment results in the movement of the scene on the retina. The apparent movement of these images across the retina during locomotion of the observer, termed optic flow, is important for interaction with the environment. Spiking activity in several cortical areas has been shown to be correlated with components of optic flow as well as the perception of self-motion by the observer (Ohlendorf et al., 2008; Raffi and Siegel, 2007). While the processing of optic flow does not necessitate a hierarchical cortical structure, the numerous intracortical connections between visual areas and the increasingly complexity of the motion information as signals propogate across areas suggests a rough hierarchy for the processing of flow information tied to self-motion. In the remainder of this section, we describe the physiological and psychophysical characteristics of several cortical areas involved in the processing of self-motion. We place particular emphasis on the neuron responses to visual motion which are explicitly incorporated into the model.

Visual Motion Processing

Visual information at the retina is projected via the lateral geniculate nucleus to the primary visual cortex (V1), located at the occipital pole (Felleman and Van Essen, 1991). Neurons in V1 respond to elementary properties, such as brightness, contrast, color opponency, spatial and temporal frequency, and are limited in spatial extent, each sampling 1-3° of the visual field. The dorsal stream of visual processing continues via afferent connections from V1 to the middle temporal (MT) area. In general, neurons in area MT are constrained to the contralateral visual field and have larger receptive fields than those in V1 (approximately 10°), and thus serve to integrate motion signals over larger areas of the visual field. Various

neurophysiology studies have shown that neurons in MT are among the first to be specifically tuned for visual motion, exhibiting preferences for both the speed and direction of motion (Newsome and Pare, 1988; Movshon and Newsome, 1996; Andersen, 1997, for review). Neurons in MT, in turn, project to the medial superior temporal (MST) area. Neurons in MST integrate motion over larger areas of the visual field (approximately 60°), which may encompass both the contralateral and ipsilateral visual fields. The transition from V1 to MT and subsequently to MST results in changes in the responses of neurons from spatially localized regions of spatiotemporal frequency, to planar motion, to more complex forms of motion. Neurons in the dorsal division of the medial superior temporal (MSTd) area respond preferentially to wide-field patterns of optic flow, including circular, radial, spiral and planar motions (Tanaka et al., 1989; Duffy and Wurtz, 1991; Graziano et al., 1994; Paolini et al., 2000).

Existing Models

Computational models have become increasingly important for understanding how the results of neurophysiological and psychophysical experiments relate to the integration of visual motion information across cortical areas. Several models of human visual processing, which examine various components of the hierarchy (V1, MT, MSTd), have been proposed. Many of the early models focused primarily on feed-forward computations. For instance, Wang (1995) created a competitive network which featured hierarchical connections between MT and MST populations. Using competitive techniques, the authors identified a continuum of preferred motions in hidden unit layers, which corresponded loosely to MST. Zemel and Sejnowski (1998) obtained similar results using an unsupervised network using complex motion stimuli.

Beardsley and Vaina (1998,2003) proposed a feed-forward model of

connections between MT and MSTd, whose connections were determined using back-propagation network training. The training ensured that MT and MSTd responses remained consistent with experimentally observed responses to complex motion. As an extension to their earlier work, Beardsley and Vaina (2001) examined the effect of lateral connections with MSTd. The authors showed that, with lateral connections that excited neurons with similar preferred motions and inhibited neurons with opposing preferred motions, the representation of motion patterns across a population of MST neurons is sufficient to account for human performance in a task. Most recently, Beardsley and Vaina (2004), explicitly defined a lateral connection structure for MST whose magnitude varies as a function of distance between receptive field centers and preferred motion patterns of the neurons. The inclusion of spatial specificity enabled the extraction of perceptual discrimination thresholds that were consistent with experimentally observed psychophysical results across a broader range of tasks.

5.2.2 Methods

We implemented a model of visual motion processing which explicitly incorporated neuronal populations in the middle temporal and medial superior temporal areas (Figure 5.6). Similar to the single area model described in Section 5.1, we explicitly divided each area into two distinct populations. The first population contained pyramidal neurons with purely excitatory projections (AMPAergic). The second population was comprised of interneurons that projected purely inhibitory synapses (GABAergic), and whose neuron response properties matched those in the pyramidal population. The overall model structure is similar to a two-area model of LFP generation proposed by Mattia et al. (2010).

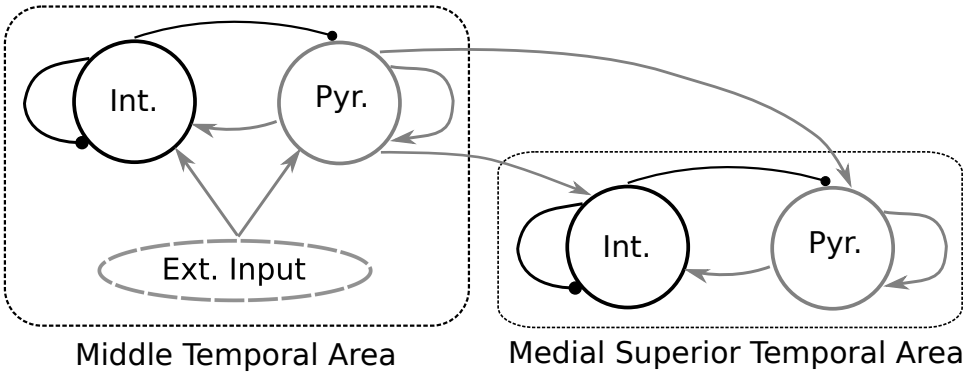


Figure 5.6: Neural network model used to characterize the hemodynamic response to visual motion stimuli processed in the middle temporal (MT) and medial superior temporal (MST) areas. The model is similar in structure to a two-area model of LFP generation proposed by Mattia et al. (2010). External input via a driving current is supplied to both the interneuron and pyramidal neuron populations of area MT. Lateral connections exist amongst neurons in both the pyramidal and inhibitory interneuron populations in both cortical areas. Additional feed-forward connections link the neural populations in MT and MST.

Middle Temporal Area

We pseudorandomly placed 10,000 MT neurons within a 2D visual field, extending $\pm 30°$ vertically and horizontally. The radius of each neuron's receptive field was chosen from a normal distribution with a mean of $5°$ and a standard deviation of $0.67°$, identical to Beardsley and Vaina (1998).

The populations corresponding to area MT featured stimulus response profiles characterized by

$$F(S) = \frac{1}{X} \sum_{m=1}^{X} \exp\left(\frac{-\min(S_m - S_j^{pref})^2}{2\sigma_j^2} \right) , \tag{5.8}$$

where S_m is the m-th stimulus motion direction vector, min is the minimum angular distance between the direction vector and the j-th neuron's preferred direction, S_j^{pref}, and X is the total number of motion direction vectors in the neuron's receptive field. This response function simulates the feed-forward projections of V1 and their subsequent effect on MT responses, and accounts for the effects of motion

opponency, in which the neuron's response to its preferred stimulus is significantly reduced when a second stimulus incorporating the opposite motion direction is superimposed on the original stimulus. Neuron preferred stimulus directions, S_j^{pref}, were chosen randomly from the [0, 2π] polar stimulus space of possible motion directions.

While there is little available literature which explicitly defines the structure of lateral connections among MT neurons, the existence of such connections is highly likely given the significant role of MT in psychophysical studies. Using a unidirectional motion stimulus, Adini et al. (1997) described a connection structure whose weighting pattern followed a Gaussian distribution in space. We chose to implement the recurrent weighting in MT such that it complemented the stimulus response profile in terms of the relationship between identically placed MT neurons with differing preferred stimulus directions. Therefore, excitatory lateral connections among the MT layers followed a Gaussian curve, whose value varied with the distance between the neuron's preferred directions . The AMPAergic lateral connection structure was described by,

$$w_{ij} = A_j^{AMPA} \exp\left(\frac{-\min(S_i^{pref} - S_j^{pref})^2}{2\sigma_{jd}^2}\right) \exp\left(\frac{-[(x_i - x_j)^2 + (y_i - y_j)^2]}{2\sigma_{js}^2}\right) , \tag{5.9}$$

where σ_{jd} corresponds to the standard deviation of the destination neuron in preferred direction of motion, and σ_{js} corresponds to the standard deviation of the Gaussian curve whose amplitude is a function of the distance between neuron receptive field centers (x, y), The value of σ_{jd} was 24.7° for all neurons (Beardsley and Vaina, 2004). The spatial standard deviation, σ_{js} was matched to the destination neuron's receptive field radius.

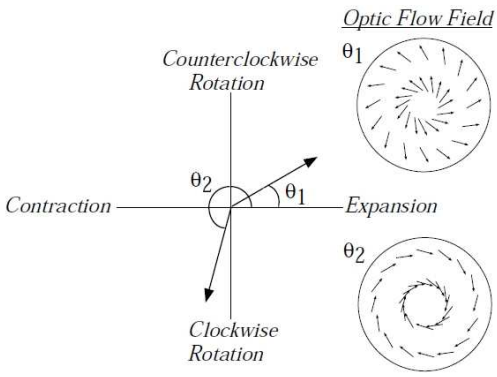


Figure 5.7: 2D stimulus space for MST neurons. The magnitude of the vector represents the mean flow speed across the motion field and the flow angle, θ, defines the type of motion. Off-axis regions correspond to spiral motion. Reproduced from Beardsley and Vaina (1998), with permission.

Medial Superior Temporal Area

We pseudorandomly placed 8,000 MST neurons within the same visual field as the MT neurons. The diameter of the receptive field of each MST neuron was assigned to be 63.0°, which is in agreement with the values used by Beardsley and Vaina (1998). Similar to the model of area MT, MST neurons were divided into two populations: pyramidal and interneurons. All neurons were randomly assigned a preferred 2D optic flow direction, defined by the angle from [0, 2π] radians within the optic flow space (Figure 5.7) (Beardsley and Vaina, 1998). Neurons where pseudorandomly assigned a preferred center of motion (COM) within the visual field whose coordinates were drawn from a normal distribution centered on the neuron's receptive field and whose standard deviation was characterized by the neuron's radius.

Extending the modeling framework outlined by Beardsley et al. (Unpublished), MT neurons which reside in the receptive field of and MST neurons were connected to the MST neuron using a standard Gaussian function,

$$w_{ij} = A_w(x,y)\exp\left(\frac{-[\min(\theta_j^{pref} - \mu_{ij}(x,y))]^2}{2\sigma_w^2(x,y)}\right) + O_w(x,y)\ , \qquad (5.10)$$

where $A_w(x,y)$ is the weight amplitude, $O_w(x,y)$ is the weight offset, $\sigma_w(x,y)$ is the standard deviation in the connection weight profile, and $\mu_{ij}(x,y)$ is the mean of the Gaussian distribution. The mean of the Gaussian distribution, $\mu_{ij}(x,y)$, was computed using the preferred optic flow direction of the MST unit, ϕ_i^{pref}, the spatial location of the MT neuron's receptive field, (x_i, y_i), and the preferred center of motion of the MST neuron, (c_j^x, c_j^x) such that

$$\begin{aligned} \mu_{ij}(x,y) &= \theta_s + \phi_j^{pref} \\ \theta_s &= \tan^{-1}\left(\frac{y_i - c_j^y}{x_i - c_j^x}\right) . \end{aligned} \tag{5.11}$$

Excitatory recurrent connections, whose source resided in the pyramidal population of MST, featured synaptic weights whose strength varied using a Gaussian function,

$$w_{ij} = S_R \exp\left(\frac{-[(x_j - c_i^x)^2 + (y_j - c_i^y)^2]}{2\sigma_{Re}^2}\right) , \tag{5.12}$$

where S_R is the amplitude of the Gaussian, assigned to be 0.04, and σ_{Re} was assigned to be 10° (Beardsley and Vaina, 2004). This results in a weighting profile that varied with the distance between the receptive field location of the source neuron and the COM location of the destination neuron. Inhibitory weights projected from interneurons were characterized by a difference of Gaussian curve,

$$\begin{aligned} w_{ij} = & \\ & -\frac{S_R}{2}\exp\left(\frac{-[(x_j - c_i^x)^2 + (y_j - c_i^y)^2]}{2\sigma_{Ri}^2}\right) \\ & -S_\phi \exp\left(\frac{-min(\phi_i - \phi_j)^2}{2\sigma_I^2}\right) , \end{aligned} \tag{5.13}$$

where synaptic weight amplitude varied with the distance between the COM of the

destination neuron and the center of the receptive field of the source neuron as well as the angular distance between the preferred optic flow directions of the neurons. The amplitude coefficient, S_ϕ, was chosen to be 0.055, and the standard deviation of the Gaussian curves, σ_{Ri} and σ_I, were both assigned to be 80° (Beardsley and Vaina, 2004).

Hemodynamic Response

We determined the magnitude of the hemodynamic response for the MT and MST populations using the process outlined in Section 5.1. Specifically, the AMPA and GABA currents incident upon MT and MST pyramidal cells were recorded and used to determined the peak magnitude of the hemodynamic curve in units of standard deviations. The aggregate mean firing rate of all neurons in both MT and MST due to stimulus presentation were calculated using Equation 5.5.

Hebbian-Style Learning

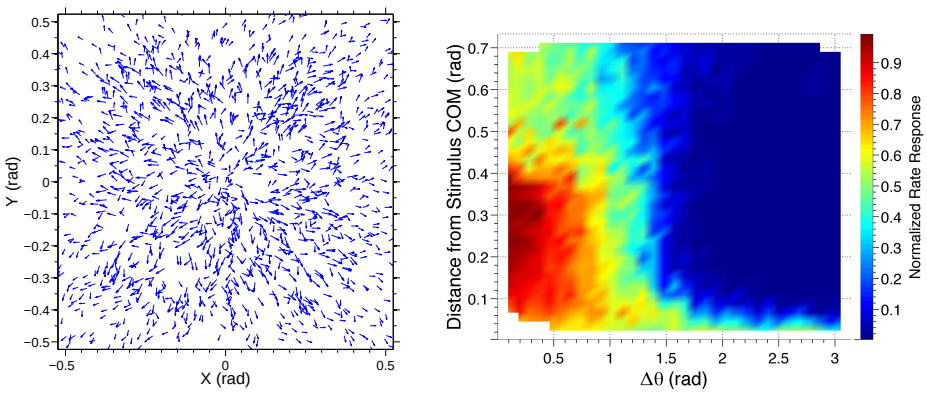
We investigated the effects of Hebbian-style weight changes at various stages in the visual motion processing hierarchy. Specifically, we examined Hebbian-style synaptic plasticity in the recurrent connections of MT and MST separately, in the feed-forward synaptic weights between MT and MST, and across the entire MT-MST visual motion complex. Similar to the procedure described in Section 5.1, we estimated each neuron's rate response to an expansion stimulus (i.e. optic flow direction of zero radians) over twenty successive trials. These response estimates were then used to perform *post hoc* weight modifications for a range of learning coefficients, ζ, according to Equation 5.7.

5.2.3 Results

All population hemodynamic and AMFR responses were determined for twenty successive presentations of an identical stimulus. The stimulus featured five seconds without a supplied stimulus, allowing all neurons to fire near their background rates. An additional five seconds of simulation time featured an expansion stimulus ($\phi = 0$ rad). The center of the expansion motion was tied to the center of the visual field. Unit motion vectors were placed pseudorandomly within the central 35° of the visual field, spanning ±17.5° vertically and horizontally, with with an approximate density of 0.5 dots per square degree. Twenty-five percent of all vectors were randomly relocated at a frame rate of 60 Hz.

Figure 5.8 shows the average responses of the MT neural population to the expansion stimulus. The responses of MT show that neurons which preferred the local motion used to characterize an expansion fired near their assigned maximum response. Figure 5.8(b) provides the normalized rate response of the MT population as a function of both the distance from the neuron's receptive field to the center of the stimulus motion as well as the difference between the neuron's preferred direction and the supplied stimulus direction ($\Delta\theta$). The plot shows that neurons whose receptive field centers were near the edge of the supplied stimulus ranges (-17.5 to 17.5° in either direction) and whose $\Delta\theta$ was close to zero showed responses near their maximum. Neurons whose receptive field centers were close the center of motion of the stimulus had lower responses due to the averaging effect of the stimulus response profile (Equation 5.8).

Figure 5.9 provides the BOLD response (SD units) and the aggregate mean firing rate of the MT and MST pyramidal populations. In the case where synaptic learning is limited to the lateral connections among MT neurons, both the AMFRs of the MT and MST populations increased with the learning coefficient. The AMFR



(a) (b)

Figure 5.8: Response of the MT pyramidal neuron population to an expansion stimulus. a) Spatial location of MT receptive field centers in visual coordinates. The direction of the arrow corresponds to the preferred motion direction. Vector lengths correspond to the normalized rate response of each neuron in response to an expansion stimulus centered in the visual field. Only neurons which exceeded a normalized rate response of 0.75 are shown. b) Surface plot showing the normalized maximum rate response of the MT neural population as a function of distance from the stimulus center of motion and the minimum angular distance between neurons' preferred stimulus motion directions and the presented motion direction ($\Delta\theta$).

of area MT increased faster slope= 0.93 spikes/sec/ζ, than did MST, slope= 0.17 spikes/sec/ζ. However, the magnitude of the hemodynamic response was not significantly related to ζ in either MT, $F(1,19) = 0.61, p > 0.05$, or MST, $F(1,19) = 1.07, p > 0.05$. When weight changes were limited to the feed-forward connections between MT and MST, there was no significant relationship between ζ and the magnitude of the hemodynamic response in MT, $F(1,19) = 1.18, p > 0.05$. The AMFR of MT was also invariant to changes in the learning coefficient, with firing rates remaining at approximately 15.5 spikes/sec for all values of ζ. However, both the AMFR and magnitude of the hemodynamic response of the MST population were significantly affected by the learning coefficient. The hemodynamic response increased with ζ, $F(1,19) = 14.94, p < 0.01$ as did the AMFR. Hebbian-like changes in the lateral connections within the MST population resulted in changes to the AMFR for the MST area only. The AMFR and hemodynamic

response of the MT population remained constant across all values of ζ, $F(1, 19) = 0.21, 0.61, p > 0.05$. While the AMFR of the MST population increased from 20 to 45 spikes/sec, the hemodynamic response remained largely constant, $F(1, 19) = 1.07, p > 0.05$. In the case where synaptic changes occurred at all stages in the model, the aggregate mean firing rate of both the MT and MST populations increased, and the magnitudes of the hemodynamic responses for both populations decreased for Hebbian-style changes in the synaptic efficacy of the connection between neurons (MT: $F(1, 19) = 7.83, p < 0.05$; MST: $F(1, 19) = 15.76, p < 0.01$).

5.3 Discussion

Our results indicate that, even in relatively simple single area network topologies, the aggregate mean firing rate of the population is not necessarily an accurate indicator of the hemodynamic response. Conversely, the magnitude of the hemodynamic response relative to baseline may not be an accurate predictor of underlying population spiking activity. For instance, in the case of a recurrently connected network representing a single cortical area, large changes in the AMFR (> 20 spikes/sec) coincided with a significant decrease in the hemodynamic response (Figure 5.4). In addition, some cases in which synaptic plasticity is constrained to lateral connections show that AMFR changes may not result in a significant change in the BOLD response (Figure 5.9(a,c)).

Interestingly, it appears that the location of the learning can be, at least, partially inferred from hemodynamic changes. For instance, Hebbian-style synaptic modification in the feed-forward connections between areas resulted in an increase in the BOLD response of upstream areas (Figure 5.9b). However, Hebbian-style changes in the efficacy of recurrent connections with learning tended to result in either a decrease or consistent hemodynamic response when compared with

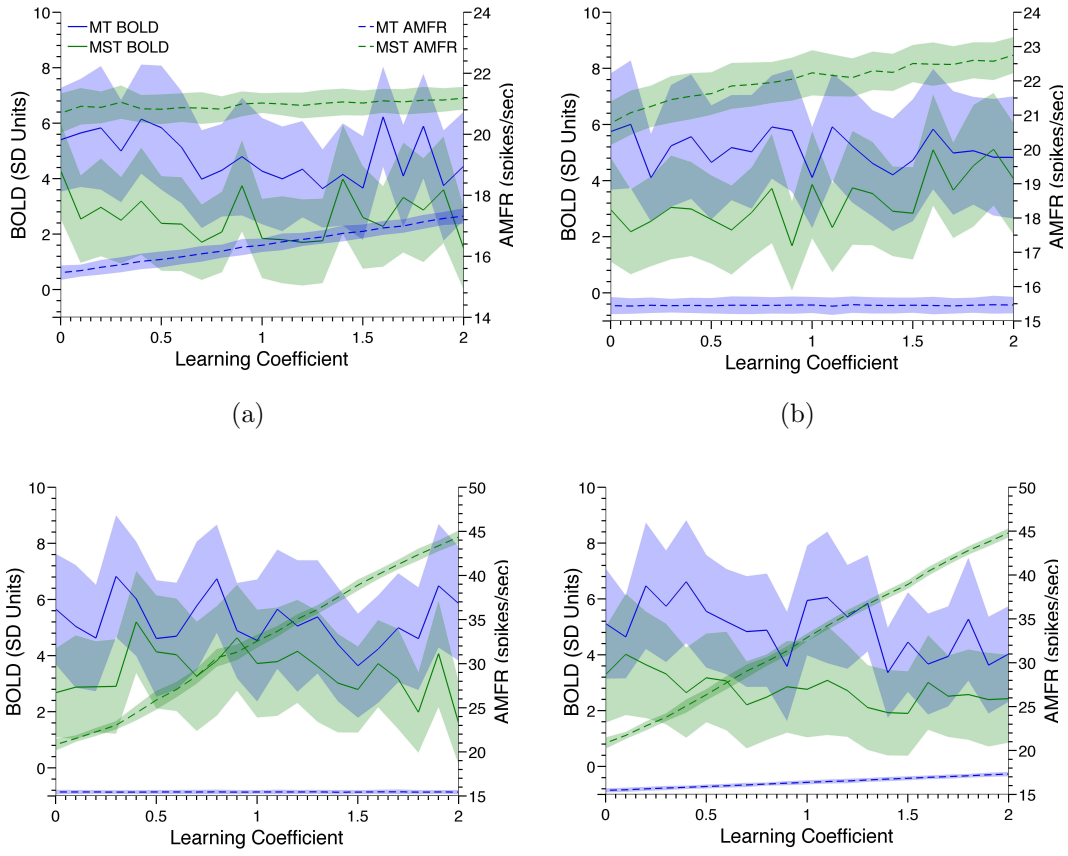


Figure 5.9: Hemodynamic responses and aggregate mean firing rates for MT and and MST pyramidal populations after *post hoc* weight modification due to Hebbian-like learning. a) Hemodynamic response magnitudes and AMFR values in MT and MST as a function of the learning coefficient, ζ, for plasticity constrained to the recurrent connections in MT. Solid and dashed blue lines correspond to the hemodynamic magnitude (SD units) and AMFR (spikes/sec) for the MT neural population, respectively. BOLD and AMFR results for MST are shown in green. Shaded regions denote standard error. b) Hemodynamic and AMFR results for Hebbian-style learning in the feed-forward connections between MT and MST. c) BOLD and AMFR results when Hebbian-style weight modification is constrained to recurrent connections within MST. d) AMFR and BOLD results for simultaneous Hebbian-style weight modifications across all connections in MT and MST between areas (i.e. recurrent connections in MT and MST as well as feed-forward connections between areas).

pre-learning hemodynamic estimates. We hypothesize that learning which occurs in the lateral connections within an area tends to improve the signal to noise ratio associated with stimulus estimation (i.e. an improvement in maximum likelihood estimates). While the spiking output of the population tends to increase in these situations, the magnitude of the LFP signal within the 40-60 Hz bandwidth decreases relative to baseline. The decrease in 40-60 Hz oscillations may indicate a more consistent estimate of the original signal with decreased metabolic cost. In contrast, when learning is constrained to feed-forward connections between areas, there is no improvement in the maximum likelihood estimate of the presented stimulus. In this case, the weight changes act as a gain on the upstream population, producing increased firing rates without a considerable improvement in the signal to noise ratio. This would generally lead to increased metabolic demands in upstream populations and higher BOLD response.

Since the hemodynamic response is determined relative to baseline activity, it is important to note that changes in baseline population dynamics can have an impact on interpretation of the magnitude of the hemodynamic response. For instance, as shown in Figure 5.5, the standard deviation of the baseline signal within the 40-60 Hz bandwidth consistently increase in response to Hebbian-like synaptic plasticity. In this case, if the population estimate of the presented stimulus remains relatively constant (i.e. encoding of the signal is not significantly altered) but the baseline activity of the population increases, the overall magnitude of the hemodynamic response relative to baseline would decrease. This suggests that the change in BOLD magnitude as a result of synaptic plasticity are tied to changes in the background activity of the population. In this case, plasticity dependent changes in baseline BOLD activity would need to be taken into account as part of the fMRI analysis to relate changes in BOLD to underlying neuronal activity.

Our results generally support the hypothesis from Logothetis (2008),

suggesting that changes in the BOLD response may be tied to the ratio of excitation/inhibition in neural subpopulations rather than to the aggregate rate response or MUA/SUA. We extend this hypothesis, noting that improvements in the maximum likelihood estimates due to Hebbian-style modification of lateral connections weights will show a systematic decrease in the magnitude of the BOLD signal with learning, while changes to feed-forward weights will result in increases in the overall BOLD response of upstream areas. Together, these results have implications for the use of mean-field based forward models of the BOLD response; indicating that the inhibitory and excitatory neural dynamics within the modeled population should be incorporated to accurately predict hemodynamic response properties.

6 CONCLUSIONS & FUTURE DIRECTIONS

Recurrent neural networks have been used extensively in the literature to describe various neurophysiological phenomena (Deneve et al., 1999; 2001). Here, we presented a modeling framework which allows *a priori* specification of synaptic weights to elicit neuron spiking responses to stimuli drawn from available neurophysiology literature. The specification of synaptic weights is predicated on the assumption that synaptic inputs are uncorrelated for sufficiently large populations of neurons. We note that this assumption is common to mean-field models of cortical populations.

The specification of large populations of connected spiking neurons necessitated the creation of a novel computational framework. We described how this framework was designed to take advantage of high performance computational architectures. In particular, the use of the Message Passing Interface (MPI) allows the computational work and memory requirements of the simulation to be divided across a number of independent processors. For large populations of spiking neurons performance improves as additional processes are added to the MPI communicator. In addition, we showed that the performance of the simulator is linearly related to the number of neuron spike events. Since the number of spike events is constrained by the total number of simulated neurons, this indicates that simulation time in overdriven network scales with neurons.

Finally, we used the joint computational and modeling frameworks to investigate the relationship between the magnitude of the hemodynamic BOLD response and aggregate population firing rates. In a preliminary series of simulations, we showed that the BOLD response relative to baseline may not be an

accurate predicator of the underlying population firing rate in recurrently connected neural networks. Instead, changes in the BOLD response are likely tied to the ratio of excitatory/inhibitory activity in neural subpopulations. This has implications for the use of mean-field based forward models of the BOLD response; indicating that the inhibitory and excitatory neural dynamics within the modeled population should be incorporated in order to obtain an accurate prediction of the hemodynamic curve.

As demonstrated by the case studies of Chapters 3 and 5, the described computational and modeling frameworks allows rapid construction and simulation of neural networks which are based on available neurophysiological responses. We anticipate that that this framework will allow researchers to quickly test hypotheses regarding underlying neural dynamics. In addition, the modeling and computational framework explicitly bridge the experimental responses of spiking neurons with physiological measures of neural activity across the population. This allows analysis, for instance, of the underlying neural dynamics associated with mean-field models.

6.1 Future Directions

The speed of the high performance simulation environment, outlined in Chapter 4, is tied directly to the slowest process in the communicator. This implies that an overscheduled machine (in terms of neurons or spike events) may severely impact the overall speed of the simulation environment. Since distribution of neurons across the ranks are performed in a round-robin fashion, the implemented framework assumes that available processors in the communicator are homogeneous. Speed increases could be accomplished by allowing neurons and their associated synapses to cross process boundaries when an uneven distribution compromises the speed of the simulation. For instance, a simple algorithm which monitors the time between the synchronization required for distribution of spike events could be used to

determine if migration of neurons/synapses is necessary. If the time for an individual process fell well outside the mean time (e.g. ± 2 standard deviations), a small subset of its assigned neurons and associated synapses would be transferred to the processor with the shortest time. This rudimentary load-balancing algorithm could provide dramatic increases in the speed of processing for simulations with long durations. However, the current scheme to monitor neuron current, voltage, etc. makes this algorithm difficult to implement due to in-memory buffering of costly I/O operations.

In addition, the current simulation framework does not inherently handle learning paradigms. However, a true Hebbian based learning algorithm would be relatively straight forward to implement. This extension to the existing framework would allow the user to specify which synapses were allowed to modulate in a Hebbian fashion, as well as specify the learning rate of change in efficacy associated with the synapse (i.e. learning rate). There is some danger in implementing such a learning algorithm in the existing framework, particularly when dealing with a recurrent network, due to the ability of the weights to grow without bound. Care would have to be taken when implementing this feature to ensure that Hebbian learning would not lead to network instability or complete depression.

In all of the models presented here, either deterministic or Poisson leaky integrate-and-fire neurons were used. However, using the scalar value α, the computational framework allows various neuron encoding models to be used. The only stipulations for inclusion of these alternate neuron models in the model framework are 1) that the neuron's generate discrete action potentials capable of being described as binary events and 2) the input to the neuron model is membrane current. However, rigorous examination of the network dynamics associated with underlying neuron model has not been performed. In future studies, it would be beneficial to ensure that network dynamics under steady-state conditions are

invariant to the underlying neuron model.

Finally, we note that all of the implemented models specify the functional structure of the connection topology. The synaptic weights are then normalized to elicit maximum responses and offset to ensure background firing rates when no stimulus is supplied. While neuron response profiles can be found by evaluation of the weight and stimulus response profiles, this can be cumbersome. In addition, neurophysiology or modeling literature which specifies synaptic weight structures may be not readily available for some areas. Instead, a modeling framework could be constructed which specifies weights given neuron response profiles. For instance, consider a neuron response profile, $G_j(\mathbf{S})$, that is well-defined across all dimensions of $\mathbf{S}$ (to ease nomenclature, we use a unidimensional stimulus, S, for the remainder or this section). A series of M synaptic connections for every neuron, can be described by a vector of weights,

$$\mathbf{w}_j = [w_{j1}, w_{j2}, ..., w_{jM}] \ . \tag{6.1}$$

If the neuron response function is evaluated at M distinct points for all dimensions, the neuron response rate $\mathbf{R}$,

$$\mathbf{R}_j = [G_j(S_1), G_j(S_2), ...G_j(S_M)] \ , \tag{6.2}$$

can be converted to units of current, $\mathbf{I}_j(S)$, through inversion of the chosen neuron model. If this is done for each connected neuron, an $M \times M$ matrix containing the evaluation of the M-th connected neuron's response profile at the j-th neuron's

sampled stimuli can be constructed:

$$\mathbf{A}_j = \begin{bmatrix} G_1(S_1) & G_1(S_2) & \cdots & G_1(S_M) \\ G_2(S_1) & G_2(S_2) & \cdots & G_2(S_M) \\ \vdots & \vdots & \vdots & \vdots \\ G_M(S_1) & G_M(S_2) & \cdots & G_M(S_M) \end{bmatrix}. \tag{6.3}$$

The weights to elicit steady-state responses can then be found by solving $\mathbf{w} = \mathbf{I}\mathbf{A}^{-1}$. This framework would require that both neuron response properties and an inverse model of neuron dynamics were well-defined. However, the functional structure of the synaptic weights would not be needed. Given this framework, it would be interesting to compare the synaptic weight structure used to elicit physiologically defined neuron response profiles with those used to define the functional network topologies used in Chapter 5.

7 APPENDIX

7.1 Unconnected Poisson LIF Neuron Model

```
#!/usr/bin/env python
import sys
from recurrent import *
from math import pi
import random

def main():
    # Create a new simulation
    sim = Simulation(T=50, dt=2.5e-4)
    N = 1000
    max_resp = [40, 100] # Range of maximum responses
    tau_ref = [2e-3, 5e-3]
    tau_rc = [10e-3, 30e-3] # RC Time constant
    pref_stim = [0, 2*pi] # Prefered stimulus
    std_stim = [pi/4.0, pi/2.0] # Standard deviation of tuning

    # Set the PSC length multiplier
    sim.set_psc_length_multiplier(5) # 5x PSC filter length

    # Monitors for the population
    spike_mon = FileMonitor('spiketrain', 'spiketrain', 4000)
    sim.add_sim_monitor(spike_mon)

    population = sim.add_population(0) # Initially no neurons
    for i in range(0, N):
        # Neuron receives all of its input from the stimulus
        maximum_rate = random.uniform(*max_resp)
        background_rate = 0.10 * maximum_rate
        neuron = LIFNeuron(maximum_rate=maximum_rate,
            background_rate=background_rate,
                   beta=1.0, tau_ref=tau_ref, tau_rc=tau_rc,
                       initial_voltage=[0, 1])
        neuron.set_poisson_spiking(True)
        population.add_neuron(neuron)
```

```
    population.add_neuron_monitor(spike_mon)

    # Generate unique ids
    sim.gen_unique_neuron_ids()

    # Create a stimulus at pi degrees
    stimulus = FileStimulus('stimulus.bin', sim.Nt)
    sim.add_sim_stimulus(stimulus)
    stimulus.add_row([pi for i in range(stimulus.Nt)])

    # Add tuning functions
    neuron = population.get_neurons()
    for i in range(0, N):
        neuron[i].add_tuning(tuning=Gaussian(pref_stim=pref_stim,
           std_stim=std_stim),
                       stimulus=stimulus)

    # Write the specification file
    f = open('unconnected_poisson.xml', 'w')
    f.write(sim.write_xml())
    f.close()
    sys.exit(0) # Done!

if __name__ == '__main__':
    main()
```

7.2 Single Layer Connected Model

```
#!/usr/bin/env python
import sys
from recurrent import *
from math import pi, exp, ceil
import random

# The average connections per neuron
average_connections = 10000

def main():
    # Create a new simulation
    sim = Simulation(T=3, dt=2.5e-4)
    N = 100000
    maximum_rate = [40, 80] # Range of maximum responses
    background_percent = 0.10
```

```
tau_ref = [2e-3, 5e-3]
tau_rc = [10e-3, 30e-3] # RC Time constant
pref_stim = [0, 2*pi] # Prefered stimulus
std_stim = [pi/8.0, pi/4.0] # Standard deviation of tuning

# Set the PSC length multiplier
sim.set_psc_length_multiplier(5) # 5x PSC filter length

# Monitors for the population
spike_mon = FileMonitor('spiketrain.bin', 'spiketrain', 1000)
sim.add_sim_monitor(spike_mon)

# Create a stimulus at pi radians
stimulus = FileStimulus('stimulus.bin', int(ceil(1/sim.dt)),
   start_index=ceil(1/sim.dt), end_index=2*ceil(1/sim.dt))
stimulus.add_row([pi for i in range(stimulus.Nt)])
#stimulus = NoStimulus()
sim.add_sim_stimulus(stimulus)

population = sim.add_population(0) # Initially no neurons
for i in range(0, N):
    s_pref = random.uniform(*pref_stim)
    s_std = random.uniform(*std_stim)
    max_resp = random.uniform(*maximum_rate)
    background_rate = background_percent * max_resp
    # Neuron receives all of its input from the stimulus
    neuron = LIFNeuron(maximum_rate=max_resp,
       background_rate=background_rate, beta=0.5,
                   tau_ref=tau_ref, tau_rc=tau_rc,
                      initial_voltage=[0, 1])
    # Add Gaussian tuning
    neuron.add_tuning(tuning=Gaussian(pref_stim=s_pref,
       std_stim=s_std),
        stimulus=stimulus)
    #neuron.set_poisson_spiking(True)
    neuron.set_unadvertised_attr('s_pref', s_pref)
    neuron.set_unadvertised_attr('s_std', s_std)
    neuron.set_unadvertised_attr('recurrent_beta', 0.5)
    population.add_neuron(neuron)

# Add the spike train monitor to the entire population
population.add_neuron_monitor(spike_mon)

# Generate unique ids
sim.gen_unique_neuron_ids()
```

```
    # Write the specification file
    f = open('connected_single_layer.xml', 'w')
    f.write(sim.write_xml())
    f.close()

    # Create the connection file
    f = open('connections.bin', 'wb')
    gen_recurrent_connections(sim, population.get_neurons(), f)
    f.close()

    sys.exit(0) # Done!

#--------------------------------------------------------------------------
# CONNECTIONS
#--------------------------------------------------------------------------
def gaussian(x, mean, std):
    """Returns a weight from a gaussian profile with the given
       ``mean'' and
    ``std''."""
    return exp(-(x - mean)**2 / (2 * std**2))

def gen_recurrent_connections(sim, neurons, connection_file):
    # Connect the neurons using a gaussian connection topology

    # Determine the probability of a connection
    connection_probability = float(average_connections) / len(neurons)
    print connection_probability

    for dest_neuron in neurons:
        print dest_neuron.id
        background_rate = 0 # Sum of all background rates
        background_input = 0 # Sum of input at background
        connections = [] # Empty list to store connections
        for src_neuron in neurons:
            if src_neuron is dest_neuron:
                continue # No self excitation
            if random.uniform(0, 1) > connection_probability:
                continue
            # Determine the weight
            weight = gaussian(src_neuron.s_pref, dest_neuron.s_pref,
               dest_neuron.s_std)
            # Determine the input at background
            background_rate += src_neuron.background_rate
            background_input += src_neuron.background_rate * weight *
```

```
                sim.dt
            # Determine the activity of the src_neuron at the
                dest_neuron's preferred direction
            src_rate_resp = (src_neuron.max_resp) * \
                gaussian(dest_neuron.s_pref, src_neuron.s_pref,
                    src_neuron.s_std) + \
                src_neuron.background_rate
            connections.append([Connection(src_neuron, dest_neuron,
                weight), src_rate_resp])
        # Equalize the weights be subtracting off the background
            activity
        mean_background_rate = background_rate / len(connections) #
            r_bar
        for connection, src_rate_resp in connections:
            connection.weight = connection.weight - (background_input /
                \
                (sim.dt * mean_background_rate * len(connections)))
        # Normalize input
        total_input = 0
        for connection, src_rate_resp in connections:
            total_input += connection.weight * sim.dt * src_rate_resp
        assert total_input > 0
        for connection, src_rate_resp in connections:
            connection.weight *= (dest_neuron.recurrent_beta) /
                total_input
        # Write the connections to a file
        write_connections_list([i[0] for i in connections],
            connection_file)
        print len(connections)
        del connections

if __name__ == '__main__':
    main()
```

7.3 Cue Integration Model

```
#!/usr/bin/env python
import sys
import os
import time
import multiprocessing
import shutil
from recurrent import *
from initialize_new_sim import *
```

```
from math import *
import random

def main():
    # Create a new simulation
    sim = Simulation(T=sim_time, dt=sim_dt)

    # Set the PSC length multiplier
    sim.set_psc_length_multiplier(psc_filter_multiplier)

    # Create monitors
    spiketrain_monitor = FileMonitor('spiketrains.bin', "spiketrain",
        1000)
    sim.add_sim_monitor(spiketrain_monitor)

    # Create a stimulus for the ``eye-centered'' population
    eye_centered_pos_stim = FileStimulus('eye_centered_pos.stim',
        sim.Nt)
    sim.add_sim_stimulus(eye_centered_pos_stim)
    # -45 degrees
    eye_centered_pos_stim.add_row([-20*pi/180 for i in
        range(eye_centered_pos_stim.Nt)])
    # Create a stimulus for the ``eye position'' population
    eye_pos_stim = FileStimulus('eye_pos.stim', sim.Nt)
    sim.add_sim_stimulus(eye_pos_stim)
    eye_pos_stim.add_row([20*pi/180 for i in range(eye_pos_stim.Nt)])
    # Create a (NULL) head centered stimulus
    #head_centered_pos_stim = NoStimulus()
    head_centered_pos_stim = FileStimulus('head_centered_pos.stim',
        sim.Nt)
    sim.add_sim_stimulus(head_centered_pos_stim)
    head_centered_pos_stim.add_row([0*pi/180 for i in
        range(head_centered_pos_stim.Nt)])

    #---
    # Create the ``eye-centered'' position population
    #---
    pop_eye_centered_pos = sim.add_population(0)
    for i in range(pop_eye_centered_pos_num_neurons):
        max_resp = random.uniform(*pop_eye_centered_pos_max_resp)
        s_pref = random.uniform(*pop_eye_centered_pos_spref)
        s_std = random.uniform(*pop_eye_centered_pos_sdev)
        neuron = LIFNeuron(maximum_rate=max_resp,
                           background_rate=max_resp *
                               pop_eye_centered_pos_background_percent,
```

```
                    tau_ref=tau_ref_range,
                    tau_rc=tau_rc_range,
                    v_th=v_th,
                    r_leak=r_leak,
                    tau_psc=tau_psc_range,
                    beta=pop_eye_centered_pos_driving_beta,
                    initial_voltage=[0, v_th])
    neuron.add_tuning(tuning=Gaussian(pref_stim=s_pref,
       std_stim=s_std),
                  stimulus=eye_centered_pos_stim,
                  column=0)
    # Neuron's preferred direction
    neuron.set_unadvertised_attr('s_pref', s_pref)
    neuron.set_unadvertised_attr('s_std', s_std)
    neuron.set_unadvertised_attr('recurrent_beta',
       pop_eye_centered_pos_recurrent_beta)
    neuron.set_unadvertised_attr('fb_beta',
       pop_eye_centered_pos_fb_beta)
    pop_eye_centered_pos.add_neuron(neuron)

#---
# Create the ``head-centered'' position population
#---
pop_head_centered_pos = sim.add_population(0)
for i in range(pop_head_centered_pos_num_neurons):
    max_resp = random.uniform(*pop_head_centered_pos_max_resp)
    s_pref = random.uniform(*pop_head_centered_pos_spref)
    s_std = random.uniform(*pop_head_centered_pos_sdev)
    neuron = LIFNeuron(maximum_rate=max_resp,
                    background_rate=max_resp *
                       pop_head_centered_pos_background_percent,
                    tau_ref=tau_ref_range,
                    tau_rc=tau_rc_range,
                    v_th=v_th,
                    r_leak=r_leak,
                    tau_psc=tau_psc_range,
                    beta=pop_head_centered_pos_driving_beta,
                    initial_voltage=[0, v_th])
    neuron.add_tuning(tuning=Gaussian(pref_stim=s_pref,
       std_stim=s_std),
                  stimulus=head_centered_pos_stim,
                  column=0)
    # Neuron's preferred direction
    neuron.set_unadvertised_attr('s_pref', s_pref)
    neuron.set_unadvertised_attr('s_std', s_std)
```

```
    neuron.set_unadvertised_attr('recurrent_beta',
       pop_head_centered_pos_recurrent_beta)
    neuron.set_unadvertised_attr('fb_beta',
       pop_head_centered_pos_fb_beta)
    pop_head_centered_pos.add_neuron(neuron)

#---
# Create the ``eye'' position population
#---
pop_eye_pos = sim.add_population(0)
for i in range(pop_eye_pos_num_neurons):
    max_resp = random.uniform(*pop_eye_pos_max_resp)
    s_pref = random.uniform(*pop_eye_pos_spref)
    s_std = random.uniform(*pop_eye_pos_sdev)
    neuron = LIFNeuron(maximum_rate=max_resp,
                       background_rate=max_resp *
                          pop_eye_pos_background_percent,
                       tau_ref=tau_ref_range,
                       tau_rc=tau_rc_range,
                       v_th=v_th,
                       r_leak=r_leak,
                       tau_psc=tau_psc_range,
                       beta=pop_eye_pos_driving_beta,
                       initial_voltage=[0, v_th])
    neuron.add_tuning(tuning=Gaussian(pref_stim=s_pref,
       std_stim=s_std),
                      stimulus=eye_pos_stim,
                      column=0)
    # Neuron's preferred direction
    neuron.set_unadvertised_attr('s_pref', s_pref)
    neuron.set_unadvertised_attr('s_std', s_std)
    neuron.set_unadvertised_attr('recurrent_beta',
       pop_eye_pos_recurrent_beta)
    neuron.set_unadvertised_attr('fb_beta', pop_eye_pos_fb_beta)
    pop_eye_pos.add_neuron(neuron)

#---
# Create the integration population
#---
pop_integration = sim.add_population(0)
for i in range(pop_integration_num_neurons):
    max_resp = random.uniform(*pop_integration_max_resp)
    x_s_pref = random.uniform(*pop_eye_pos_spref)
    y_s_pref = random.uniform(*pop_eye_centered_pos_spref)
    x_s_std = random.uniform(*pop_eye_pos_sdev)
```

```
    y_s_std = random.uniform(*pop_eye_centered_pos_sdev)
    neuron = LIFNeuron(maximum_rate=max_resp,
                       background_rate=max_resp *
                          pop_integration_background_percent,
                       tau_ref=tau_ref_range,
                       tau_rc=tau_rc_range,
                       v_th=v_th,
                       r_leak=r_leak,
                       tau_psc=tau_psc_range,
                       beta=pop_integration_driving_beta,
                       initial_voltage=[0, v_th])
    # Neuron's preferred direction
    neuron.x = x_s_pref
    neuron.y = y_s_pref
    neuron.set_unadvertised_attr('x_s_pref', x_s_pref)
    neuron.set_unadvertised_attr('y_s_pref', y_s_pref)
    neuron.set_unadvertised_attr('x_s_std', x_s_std)
    neuron.set_unadvertised_attr('y_s_std', y_s_std)
    neuron.set_unadvertised_attr('recurrent_beta',
       pop_integration_recurrent_beta)
    neuron.set_unadvertised_attr('ff_beta',
       pop_integration_ff_beta)
    pop_integration.add_neuron(neuron)


# Add the spike train monitor to all neurons in the simulation
sim.add_neuron_monitor(spiketrain_monitor)

# Generate a unique ID for all neurons in the simulation
sim.gen_unique_neuron_ids()

# Generate connections
processes = []
connections_file_1 = open("connections1.bin", "wb")
p = multiprocessing.Process(target=gen_ff_connections,
       args=(sim, pop_eye_centered_pos, pop_head_centered_pos,
          pop_eye_pos,
           pop_integration, connections_file_1))
p.daemon = True
p.start()
processes.append(p)

#gen_ff_connections(sim, pop_eye_centered_pos,
   pop_head_centered_pos, pop_eye_pos,
# pop_integration, connections_file)
```

```
#gen_fb_connections(sim, pop_eye_centered_pos,
   pop_head_centered_pos, pop_eye_pos,
# pop_integration, connections_file)
connections_file_2 = open("connections2.bin", "wb")
p = multiprocessing.Process(target=gen_fb_connections,
       args=(sim, pop_eye_centered_pos, pop_head_centered_pos,
          pop_eye_pos,
           pop_integration, connections_file_2))
p.daemon = True
p.start()
processes.append(p)

connections_file_3 = open("connections3.bin", "wb")
p = multiprocessing.Process(target=gen_lateral_connections,
       args=(sim, pop_eye_centered_pos, pop_head_centered_pos,
          pop_eye_pos,
           pop_integration, connections_file_3))
p.daemon = True
p.start()
processes.append(p)

for i in processes:
    while i.pid is None:
        time.sleep(0.25)
    i.join()

connections_file_1.close()
connections_file_2.close()
# Concatenate the files
connections_file = open('connections.bin', 'wb')
shutil.copyfileobj(open('connections1.bin', 'rb'),
   connections_file)
shutil.copyfileobj(open('connections2.bin', 'rb'),
   connections_file)
shutil.copyfileobj(open('connections3.bin', 'rb'),
   connections_file)
connections_file.close()
os.remove('connections1.bin')
os.remove('connections2.bin')
os.remove('connections3.bin')

# Write the specifications file
f = open("pouget_spiking.xml", "w")
f.write(sim.write_xml())
f.close()
```

```
    sys.exit(0) # Done!

#------------------------------------------------------------------------
# CONNECTIONS
#------------------------------------------------------------------------
def dog_radians(x, mean, std1, std2, a1, a2):
    """Computes the value of a DOG with standard deviations std1 and
       std2. The
    incomming values are assumed to be in radians. THIS FUNCTION DOES
       NOT WRAP
    ANGULAR MEASURES! The max value is normalized to 1. The parameters
    ``a1'' and ``a2'' are the gains of the first and second
       gaussians"""
    assert std1 < std2
    assert a1 > a2
    return exp(-(x - mean)**2 / (2 * std1**2))
    return ((a1) * exp(-(x - mean)**2 / (2 * std1**2)) - \
          (a2) * exp(-(x - mean)**2 / (2 * std2**2))) / \
          (a1 - a2)

def gaussian(x, mean, std1):
    """Computes the value of a Gaussian with standard deviation std1.
    incomming values are assumed to be in radians. THIS FUNCTION DOES
       NOT WRAP
    ANGULAR MEASURES! The max value is normalized to 1."""
    return (exp(-(x - mean)**2 / (2 * std1**2)))

def equalize_and_normalize(total_background_rate, background_input,
   connections, dt, beta=1):
    """Equalize and normalize a set of weights"""
    mean_background_rate = total_background_rate / len(connections)
    for connection, src_rate_resp in connections:
        connection.weight = connection.weight - background_input /
           (mean_background_rate * dt * len(connections))

    total_input = 0
    for connection, src_rate_resp in connections:
        total_input += connection.weight * src_rate_resp * dt
    assert total_input > 0
    for connection, src_rate_resp in connections:
        connection.weight *= abs(beta / total_input)
    return connections

def gen_ff_connections(sim, pop_eye_centered_pos,
   pop_head_centered_pos,
```

```
pop_eye_pos, pop_integration, connections_file):
all_src_neurons = pop_eye_centered_pos.get_neurons() + \
    pop_eye_pos.get_neurons() + pop_head_centered_pos.get_neurons()

# Generate connections for (destination) neurons in the
   integration layer
for dest_neuron in pop_integration.get_neurons():
    connection_likelihood =
       float(pop_integration_average_connections *
       pop_integration_ff_beta) / len(all_src_neurons)
    connections = []
    background_rate = 0
    background_input = 0
    for src_neuron in pop_eye_centered_pos.get_neurons():
        if random.uniform(0, 1) > connection_likelihood:
            continue
        if dest_neuron is src_neuron:
            continue
        # Compute the weight to the eye-centered population
        # y_s_pref
        weight = dog_radians(src_neuron.s_pref,
           dest_neuron.y_s_pref, dest_neuron.y_s_std, 2 *
           dest_neuron.y_s_std, 1, 0.2)
        # Get the rate response for the source neuron at the
           destination
        # neuron's preferred direction
        src_rate_resp = (src_neuron.max_resp) * \
            exp(-((src_neuron.s_pref - dest_neuron.y_s_pref)**2) /
               (2 * (src_neuron.s_std)**2)) \
            + src_neuron.background_rate
        background_input += weight * src_neuron.background_rate *
           sim.dt
        background_rate += src_neuron.background_rate
        connections.append([Connection(src_neuron, dest_neuron,
           weight), src_rate_resp])

    for src_neuron in pop_eye_pos.get_neurons():
        if random.uniform(0, 1) > connection_likelihood:
            continue
        if dest_neuron is src_neuron:
            continue
        # Compute the weight to the eye population
        # x_s_pref
        weight = dog_radians(src_neuron.s_pref,
           dest_neuron.x_s_pref, dest_neuron.x_s_std, 2 *
```

```
            dest_neuron.x_s_std, 1, 0.2)
        # Get the rate response for the source neuron at the
            destination
        # neuron's preferred direction
        src_rate_resp = (src_neuron.max_resp) * \
            exp(-((src_neuron.s_pref - dest_neuron.x_s_pref)**2) /
                (2 * (src_neuron.s_std)**2)) \
            + src_neuron.background_rate
        background_input += weight * src_neuron.background_rate
            *sim.dt
        background_rate += src_neuron.background_rate
        connections.append([Connection(src_neuron, dest_neuron,
            weight), src_rate_resp])

    for src_neuron in pop_head_centered_pos.get_neurons():
        if random.uniform(0, 1) > connection_likelihood:
            continue
        if dest_neuron is src_neuron:
            continue
        # Compute the weight to the head-centered population
        # neurons in the source layer are most strongly connected
            to the
        # y_s_pref - x_s_pref
        weight = dog_radians(src_neuron.s_pref,
            (dest_neuron.y_s_pref + dest_neuron.x_s_pref),
             dest_neuron.x_s_std, 2 * dest_neuron.x_s_std, 1, 0.2)
        # Get the rate response for the source neuron at the
            destination
        # neuron's preferred direction
        src_rate_resp = (src_neuron.max_resp) * \
            exp(-((src_neuron.s_pref - (dest_neuron.y_s_pref +
                dest_neuron.x_s_pref))**2) / (2 *
                (src_neuron.s_std)**2)) \
            + src_neuron.background_rate
        background_input += weight * src_neuron.background_rate *
            sim.dt
        background_rate += src_neuron.background_rate
        connections.append([Connection(src_neuron, dest_neuron,
            weight), src_rate_resp])

    # Determine the equalized/normalized weights
    connections = equalize_and_normalize(background_rate,
        background_input, connections, sim.dt, dest_neuron.ff_beta)
    write_connections_list([i[0] for i in connections],
        connections_file)
```

```
        del connections

def gen_fb_connections(sim, pop_eye_centered_pos,
    pop_head_centered_pos,
    pop_eye_pos, pop_integration, connections_file):
    # Generate connections for (destination) neurons in the unimodal
        layers
    for dest_neuron in pop_eye_centered_pos.get_neurons():
        connection_likelihood =
            float(pop_eye_centered_pos_average_connections *
            pop_eye_centered_pos_fb_beta) /
            len(pop_integration.get_neurons())
        connections = []
        background_rate = 0
        background_input = 0
        for src_neuron in pop_integration.get_neurons():
            if random.uniform(0, 1) > connection_likelihood:
                continue
            if dest_neuron is src_neuron:
                continue
            # Compute the weight to the eye-centered population
            # y_s_pref
            weight = dog_radians(src_neuron.y_s_pref,
                dest_neuron.s_pref, dest_neuron.s_std, 2.0 *
                dest_neuron.s_std, 1, 0.2)
            # Get the rate response for the source neuron at the
                destination
            # neuron's preferred direction
            # The 0.5 scalar is required since neurons in the
                intermediate layer have 2 PREFERRED DIRECTIONS.
                Therefore,
            # evaluation at the destination neuron's preferred
                direction will only provide half of the desired input.
            src_rate_resp = 0.5 * (src_neuron.max_resp) * \
                exp(-((src_neuron.y_s_pref - dest_neuron.s_pref)**2) /
                    (2 * (src_neuron.y_s_std)**2)) \
                + src_neuron.background_rate
            background_input += weight * src_neuron.background_rate *
                sim.dt
            background_rate += src_neuron.background_rate
            connections.append([Connection(src_neuron, dest_neuron,
                weight), src_rate_resp])

        # Determine the equalized/normalized weights
        connections = equalize_and_normalize(background_rate,
```

```
            background_input, connections, sim.dt, dest_neuron.fb_beta)
        write_connections_list([i[0] for i in connections],
            connections_file)
        del connections

    for dest_neuron in pop_eye_pos.get_neurons():
        connection_likelihood = float(pop_eye_pos_average_connections *
             pop_eye_centered_pos_fb_beta) /
            len(pop_integration.get_neurons())
        connections = []
        background_rate = 0
        background_input = 0
        for src_neuron in pop_integration.get_neurons():
            if random.uniform(0, 1) > connection_likelihood:
                continue
            if dest_neuron is src_neuron:
                continue
            # Compute the weight to the eye-centered population
            # y_s_pref
            weight = dog_radians(src_neuron.x_s_pref,
                dest_neuron.s_pref, dest_neuron.s_std, 2 *
                dest_neuron.s_std, 1, 0.2)
            # Get the rate response for the source neuron at the
                destination
            # neuron's preferred direction
            # The 0.5 scalar is required since neurons in the
                intermediate layer have 2 PREFERRED DIRECTIONS.
                Therefore,
            # evaluation at the destination neuron's preferred
                direction will only provide half of the desired input.
            src_rate_resp = 0.5 * (src_neuron.max_resp) * \
                exp(-((src_neuron.x_s_pref - dest_neuron.s_pref)**2) /
                    (2 * (src_neuron.x_s_std)**2)) \
                + src_neuron.background_rate
            background_input += weight * src_neuron.background_rate *
                sim.dt
            background_rate += src_neuron.background_rate
            connections.append([Connection(src_neuron, dest_neuron,
                weight), src_rate_resp])

        # Determine the equalized/normalized weights
        connections = equalize_and_normalize(background_rate,
            background_input, connections, sim.dt, dest_neuron.fb_beta)
        write_connections_list([i[0] for i in connections],
            connections_file)
```

```
    del connections

# Head centered neuron weights
for dest_neuron in pop_head_centered_pos.get_neurons():
    connection_likelihood =
       float(pop_head_centered_pos_average_connections *
       pop_head_centered_pos_fb_beta) /
       len(pop_integration.get_neurons())
    connections = []
    background_rate = 0
    background_input = 0
    for src_neuron in pop_integration.get_neurons():
        if random.uniform(0, 1) > connection_likelihood:
            continue
        if dest_neuron is src_neuron:
            continue
        # Compute the weight to the eye-centered population
        # y_s_pref
        weight = dog_radians(src_neuron.x_s_pref +
           src_neuron.y_s_pref, dest_neuron.s_pref,
           dest_neuron.s_std, 2 * dest_neuron.s_std, 1, 0.2)
        # Get the rate response for the source neuron at the
           destination
        # neuron's preferred direction (take the mean of the
           standard
        # deviations in both directions
        # The 0.66 scalar is required since neurons in the
           intermediate layer have 2 PREFERRED DIRECTIONS.
           Therefore,
        # evaluation at the destination neuron's preferred
           direction will only provide 2/3 of the desired input
        src_rate_resp = 0.66 * (src_neuron.max_resp) * \
            exp(-((dest_neuron.s_pref - (src_neuron.x_s_pref +
               src_neuron.y_s_pref))**2) / (2 *
               (src_neuron.x_s_std)**2)) \
            + src_neuron.background_rate
        background_input += weight * src_neuron.background_rate *
           sim.dt
        background_rate += src_neuron.background_rate
        connections.append([Connection(src_neuron, dest_neuron,
           weight), src_rate_resp])

    # Determine the equalized/normalized weights
    connections = equalize_and_normalize(background_rate,
       background_input, connections, sim.dt, dest_neuron.fb_beta)
```

```
        write_connections_list([i[0] for i in connections],
            connections_file)
        del connections

def gen_lateral_connections(sim, pop_eye_centered_pos,
    pop_head_centered_pos,
    pop_eye_pos, pop_integration, connections_file):
    # Generate connections for (destination) neurons in the unimodal
        layers
    for dest_neuron in pop_eye_centered_pos.get_neurons():
        connection_likelihood =
            float(pop_eye_centered_pos_average_connections *
            pop_eye_centered_pos_recurrent_beta) /
            len(pop_eye_centered_pos.get_neurons())
        connections = []
        background_rate = 0
        background_input = 0
        for src_neuron in pop_eye_centered_pos.get_neurons():
            if random.uniform(0, 1) > connection_likelihood:
                continue
            if dest_neuron is src_neuron:
                continue
            # Compute the lateral weight
            #weight = gaussian(src_neuron.s_pref, dest_neuron.s_pref,
                dest_neuron.s_std)
            weight = dog_radians(src_neuron.s_pref, dest_neuron.s_pref,
                dest_neuron.s_std, 2 * dest_neuron.s_std, 1, 0.2)
            # Get the rate response for the source neuron at the
                destination
            # neuron's preferred direction (take the mean of the
                standard
            # deviations in both directions
            src_rate_resp = (src_neuron.max_resp) * \
                exp(-((dest_neuron.s_pref - src_neuron.s_pref)**2) / (2
                    * (src_neuron.s_std)**2)) \
                + src_neuron.background_rate
            background_input += weight * src_neuron.background_rate *
                sim.dt
            background_rate += src_neuron.background_rate
            connections.append([Connection(src_neuron, dest_neuron,
                weight), src_rate_resp])

        # Determine the equalized/normalized weights
        connections = equalize_and_normalize(background_rate,
            background_input, connections, sim.dt,
```

```
            dest_neuron.recurrent_beta)
        write_connections_list([i[0] for i in connections],
            connections_file)
        del connections

    for dest_neuron in pop_eye_pos.get_neurons():
        connection_likelihood = float(pop_eye_pos_average_connections *
             pop_eye_pos_recurrent_beta) /
            len(pop_eye_pos.get_neurons())
        connections = []
        background_rate = 0
        background_input = 0
        for src_neuron in pop_eye_pos.get_neurons():
            if random.uniform(0, 1) > connection_likelihood:
                continue
            if dest_neuron is src_neuron:
                continue
            # Compute the lateral weight
            #weight = gaussian(src_neuron.s_pref, dest_neuron.s_pref,
                dest_neuron.s_std)
            weight = dog_radians(src_neuron.s_pref, dest_neuron.s_pref,
                 dest_neuron.s_std, 2 * dest_neuron.s_std, 1, 0.2)
            # Get the rate response for the source neuron at the
                destination
            # neuron's preferred direction (take the mean of the
                standard
            # deviations in both directions
            src_rate_resp = (src_neuron.max_resp) * \
                exp(-((dest_neuron.s_pref - src_neuron.s_pref)**2) / (2
                    * (src_neuron.s_std)**2)) \
                + src_neuron.background_rate
            background_input += weight * src_neuron.background_rate *
                sim.dt
            background_rate += src_neuron.background_rate
            connections.append([Connection(src_neuron, dest_neuron,
                weight), src_rate_resp])

        # Determine the equalized/normalized weights
        connections = equalize_and_normalize(background_rate,
            background_input, connections, sim.dt,
            dest_neuron.recurrent_beta)
        write_connections_list([i[0] for i in connections],
            connections_file)
        del connections
```

```
for dest_neuron in pop_head_centered_pos.get_neurons():
    connection_likelihood =
       float(pop_head_centered_pos_average_connections *
       pop_head_centered_pos_recurrent_beta) /
       len(pop_head_centered_pos.get_neurons())
    connections = []
    background_rate = 0
    background_input = 0
    for src_neuron in pop_head_centered_pos.get_neurons():
        if random.uniform(0, 1) > connection_likelihood:
            continue
        if dest_neuron is src_neuron:
            continue
        # Compute the lateral weight
        #weight = gaussian(src_neuron.s_pref, dest_neuron.s_pref,
           dest_neuron.s_std)
        weight = dog_radians(src_neuron.s_pref, dest_neuron.s_pref,
            dest_neuron.s_std, 2 * dest_neuron.s_std, 1, 0.2)
        # Get the rate response for the source neuron at the
           destination
        # neuron's preferred direction (take the mean of the
           standard
        # deviations in both directions
        src_rate_resp = (src_neuron.max_resp) * \
            exp(-((dest_neuron.s_pref - src_neuron.s_pref)**2) / (2
                * (src_neuron.s_std)**2)) \
            + src_neuron.background_rate
        background_input += weight * src_neuron.background_rate *
           sim.dt
        background_rate += src_neuron.background_rate
        connections.append([Connection(src_neuron, dest_neuron,
           weight), src_rate_resp])

    # Determine the equalized/normalized weights
    connections = equalize_and_normalize(background_rate,
       background_input, connections, sim.dt,
       dest_neuron.recurrent_beta)
    write_connections_list([i[0] for i in connections],
       connections_file)
    del connections

for dest_neuron in pop_integration.get_neurons():
    connection_likelihood =
       float(pop_integration_average_connections *
       pop_integration_recurrent_beta) /
```

```
    len(pop_integration.get_neurons())
connections = []
background_rate = 0
background_input = 0
for src_neuron in pop_integration.get_neurons():
    if random.uniform(0, 1) > connection_likelihood:
        continue
    if dest_neuron is src_neuron:
        continue
    # Compute the lateral weight
    #weight = gaussian(src_neuron.y_s_pref,
       dest_neuron.y_s_pref, dest_neuron.y_s_std) *
       gaussian(src_neuron.x_s_pref, dest_neuron.x_s_pref,
       dest_neuron.x_s_std)
    weight = dog_radians(src_neuron.y_s_pref,
       dest_neuron.y_s_pref, dest_neuron.y_s_std, 2 *
       dest_neuron.y_s_std, 1, 0.2) * \
        dog_radians(src_neuron.x_s_pref, dest_neuron.x_s_pref,
           dest_neuron.x_s_std, 2 * dest_neuron.x_s_std, 1,
           0.2)
    # Get the rate response for the source neuron at the
       destination
    # neuron's preferred direction (take the mean of the
       standard
    # deviations in both directions
    src_rate_resp = (src_neuron.max_resp) * \
        exp(-((dest_neuron.x_s_pref - src_neuron.x_s_pref)**2)
           / (2 * (src_neuron.x_s_std)**2)) * \
        exp(-((dest_neuron.y_s_pref - src_neuron.y_s_pref)**2)
           / (2 * (src_neuron.y_s_std)**2)) \
        + src_neuron.background_rate
    background_input += weight * src_neuron.background_rate *
       sim.dt
    background_rate += src_neuron.background_rate
    connections.append([Connection(src_neuron, dest_neuron,
       weight), src_rate_resp])

# Determine the equalized/normalized weights
connections = equalize_and_normalize(background_rate,
   background_input, connections, sim.dt,
   dest_neuron.recurrent_beta)
write_connections_list([i[0] for i in connections],
   connections_file)
del connections
```

```
if __name__ == '__main__':
    main()
```

7.3.1 Simulation Parameters

```
#!/usr/bin/env python
import math

#---
# Simulation-Wide Parameters
#---
# Who many times longer than tau_psc should our filters be ?
psc_filter_multiplier = 5
sim_time = 2.0 # Seconds
sim_dt = 2.5e-4

#---
# Neuron Parameters
#---
tau_psc_range = [0.005, 0.010] # Seconds
tau_ref_range = [0.002, 0.005] # Seconds
tau_rc_range = [0.010, 0.030] # Seconds
v_th = 1
r_leak = 1

#---
# ``Eye-Centered'' Position Population
#---
pop_eye_centered_pos_num_neurons = 5000
pop_eye_centered_pos_max_resp = [80, 80] # Hz
pop_eye_centered_pos_background_percent = 0.10 # i.e. 10%
pop_eye_centered_pos_average_connections = 10000
pop_eye_centered_pos_driving_beta = 0.20
pop_eye_centered_pos_recurrent_beta = 0.40
pop_eye_centered_pos_fb_beta = 0.40
pop_eye_centered_pos_spref = [-90*math.pi/180, 90*math.pi/180] #
   Radians
pop_eye_centered_pos_sdev = [math.pi/12, math.pi/12] # Radians

#---
# ``Head-Centered'' Position Population
#---
pop_head_centered_pos_num_neurons = 5000
pop_head_centered_pos_max_resp = [80, 80] # Hz
```

```
pop_head_centered_pos_background_percent = 0.10 # i.e. 10%
pop_head_centered_pos_average_connections = 10000
pop_head_centered_pos_driving_beta = 0.20
pop_head_centered_pos_recurrent_beta = 0.40
pop_head_centered_pos_fb_beta = 0.40
pop_head_centered_pos_spref = [-90*math.pi/180, 90*math.pi/180] #
   Radians
pop_head_centered_pos_sdev = [math.pi/12, math.pi/12] # Radians

#---
# ``Eye'' Position Population
#---
pop_eye_pos_num_neurons = 5000
pop_eye_pos_max_resp = [80, 80] # Hz
pop_eye_pos_background_percent = 0.10 # i.e. 10%
pop_eye_pos_average_connections = 10000
pop_eye_pos_driving_beta = 0.20
pop_eye_pos_recurrent_beta = 0.40
pop_eye_pos_fb_beta = 0.40
pop_eye_pos_spref = [-90*math.pi/180, 90*math.pi/180] # Radians
pop_eye_pos_sdev = [math.pi/12, math.pi/12] # Radians

#---
# ``Integration'' Population
#---
pop_integration_num_neurons = 20000
pop_integration_max_resp = [80, 80] # Hz
pop_integration_background_percent = 0.10 # i.e. 10%
pop_integration_average_connections = 10000
pop_integration_driving_beta = 0.0
pop_integration_recurrent_beta = 0.5
pop_integration_ff_beta = 0.5
```

7.4 Single Area Pyramidal-Interneuron Model

```
#!/usr/bin/env python
import sys
from recurrent import *
from initialize_new_sim import *
from math import *
import random

import multiprocessing
import shutil
```

```
from glob import glob


def main():
    files = glob('connections*.bin')
    for f in files:
        try:
            os.remove(f)
        except:
            pass

    # Create a new simulation
    sim = Simulation(T=sim_time, dt=sim_dt)

    # Set the PSC length multiplier
    sim.set_psc_length_multiplier(psc_filter_multiplier)

    # Monitors for the A population

    gaba_current_monitor = FileMonitor('gaba_current', "gaba_current",
        4000, compress=True)
    sim.add_sim_monitor(gaba_current_monitor)
    # Monitors for population B
    ampa_current_monitor = FileMonitor('ampa_current', "ampa_current",
        4000, compress=True)
    sim.add_sim_monitor(ampa_current_monitor)
    #voltage_monitor = FileMonitor('voltage', "voltage", 4000,
       compress=True)
    #sim.add_sim_monitor(voltage_monitor)

    spike_train_monitor = FileMonitor('spike_train', 'spiketrain',
       4000, compress=True)
    sim.add_sim_monitor(spike_train_monitor)

    # Create the excitatory (pyramidal) population
    excitatory = sim.add_population(0)
    for i in range(excitatory_num_neurons):
        max_resp = random.uniform(*excitatory_max_resp_range)
        neuron = LIFNeuron(maximum_rate=max_resp,
                           background_rate=random.uniform(*excitatory_background_rate),
                           tau_ref=excitatory_tau_ref,
                           tau_rc=excitatory_tau_rc,
                           v_th=v_th,
                           r_leak=r_leak,
                           tau_psc=random.uniform(*excitatory_tau_psc),
```

```
                beta=excitatory_driving_beta,
                initial_voltage=[0, v_th])
    neuron.set_poisson_spiking(True)
    neuron.set_unadvertised_attr('recurrent_beta',
       excitatory_recurrent_beta)
    neuron.set_unadvertised_attr('ff_beta', excitatory_ff_beta)
    neuron.set_unadvertised_attr('fb_beta', excitatory_fb_beta)
    neuron.is_inhibitory = False
    excitatory.add_neuron(neuron)

#excitatory.add_neuron_monitor(gaba_current_monitor)
#excitatory.add_neuron_monitor(ampa_current_monitor)
excitatory.add_neuron_monitor(spike_train_monitor)
#excitatory.add_neuron_monitor(voltage_monitor)

# Create a new "B" population
inhibitory = sim.add_population(0) # No neuron in pop A
for i in range(inhibitory_num_neurons):
    max_resp = random.uniform(*inhibitory_max_resp_range)
    neuron = LIFNeuron(maximum_rate=max_resp,
                background_rate=random.uniform(*inhibitory_background_rate),
                tau_ref=inhibitory_tau_ref,
                tau_rc=inhibitory_tau_rc,
                v_th=v_th,
                r_leak=r_leak,
                tau_psc=random.uniform(*inhibitory_tau_psc),
                beta=inhibitory_driving_beta,
                initial_voltage=[0, v_th])
    neuron.set_poisson_spiking(True)
    neuron.set_unadvertised_attr('recurrent_beta',
       inhibitory_recurrent_beta)
    neuron.set_unadvertised_attr('ff_beta', inhibitory_ff_beta)
    neuron.set_unadvertised_attr('fb_beta', inhibitory_fb_beta)
    neuron.is_inhibitory = True
    inhibitory.add_neuron(neuron)

#inhibitory.add_neuron_monitor(ampa_current_monitor)
#inhibitory.add_neuron_monitor(gaba_current_monitor)
inhibitory.add_neuron_monitor(spike_train_monitor)

# Generate a unique ID for all neurons in the simulation
sim.gen_unique_neuron_ids()

# Create a new stimulus for population a
# Stimulus is on for times [5, 10] seconds
```

```
injection_stimulus = FileStimulus('injection.bin',
   int(ceil(5/sim.dt)), start_index=int(ceil(5/sim.dt)),
   end_index=int(ceil(10/sim.dt)))
# Add the stimulus to the simulation
sim.add_sim_stimulus(injection_stimulus)
# Generate the stimulus (constant with pi radians)
injection_stimulus.add_row([pi for i in
   range(injection_stimulus.Nt)])
# Add Gaussian tuning to all neurons.
for neuron in excitatory.get_neurons():
    s_pref = random.uniform(*excitatory_spref_range)
    s_std = random.uniform(*excitatory_std_range)
    neuron.add_tuning(tuning=Gaussian(pref_stim=s_pref,
       std_stim=s_std),
                      stimulus=injection_stimulus,
                      column=0)
    neuron.s_pref = s_pref
    neuron.s_std = s_std
for neuron in inhibitory.get_neurons():
    s_pref = random.uniform(*inhibitory_spref_range)
    s_std = random.uniform(*inhibitory_std_range)
    neuron.add_tuning(tuning=Gaussian(pref_stim=s_pref,
       std_stim=s_std),
                      stimulus=injection_stimulus,
                      column=0)
    neuron.s_pref = s_pref
    neuron.s_std = s_std

# Create the connections file
gen_connections(sim, inhibitory.get_neurons() +
   excitatory.get_neurons(),
 inhibitory.get_neurons() + excitatory.get_neurons(),
    gen_random_connections)
connections_file = open("connections.bin", "wb")
for i in range(0, multiprocessing.cpu_count()):
    shutil.copyfileobj(open('connections_%d.bin' % (i), 'rb'),
       connections_file)
    os.remove('connections_%d.bin' % (i))
connections_file.close()

# Write the specifications file
f = open("lfp.xml", "w")
f.write(sim.write_xml())
f.close()
sys.exit(0) # Done!
```

```
#---------------------------------------------------------------------------
# CONNECTIONS
#---------------------------------------------------------------------------
def equalize_and_normalize(total_background_rate, background_input,
    connections, dt, beta=1):
    """Equalize and normalize a set of weights"""
    mean_background_rate = total_background_rate / len(connections)
    for connection, src_rate_resp in connections:
        connection.weight = connection.weight - background_input /
            (mean_background_rate * dt * len(connections))

    total_input = 0
    for connection, src_rate_resp in connections:
        total_input += connection.weight * src_rate_resp * dt
    assert total_input > 0
    for connection, src_rate_resp in connections:
        connection.weight *= abs(beta / total_input)
    return connections

def gaussian(x, mean, std):
    return exp(-common.angle_mod(x, mean)**2 / (2 * std**2))

def gen_connections(sim, src_neurons, dest_neurons, function):
    fp = []
    cpus = multiprocessing.cpu_count()
    p = []
    start = 0
    step = int(len(dest_neurons)/float(cpus))
    for i in range(0, cpus):
        fp.append(open('connections_%d.bin' % (i), 'ab'))
        end = start + step
        if i == cpus - 1:
            end = len(dest_neurons)
        print start, end
        p.append(multiprocessing.Process(target=function,
        args=(sim, fp[-1], src_neurons,
        [dest_neurons[j] for j in range(start, end)])))
        p[-1].start()
        start = end
    for i in range(0, len(p)):
        p[i].join()
        if p[i].exitcode != 0:
            for j in range(i+1, len(p)):
                p[j].terminate()
```

```
            sys.exit(1)
        fp[i].close()

def gen_random_connections(sim, connections_file, src_neurons,
    dest_neurons):
    for dest_neuron in dest_neurons:
        connections = []
        inhibitory_background_rate = 0
        inhibitory_background_input = 0
        excitatory_background_rate = 0
        excitatory_background_input = 0

        # Choose 60% of the neurons randomly
        indices = [random.randint(0, len(src_neurons)-1) for i in
            range(0, int(0.60 * len(src_neurons)))]
        for i in indices:
            src_neuron = src_neurons[i]
            if src_neuron.is_inhibitory:
                # Gaussian centered on the antipreferred
                weight = -1.0 * gaussian(src_neuron.s_pref,
                    dest_neuron.s_pref, dest_neuron.s_std)
            else: # Src neuron is excitatory
                # Gaussian centered on the preferred
                weight = gaussian(src_neuron.s_pref,
                    dest_neuron.s_pref, dest_neuron.s_std)
            # Get the rate response of the source neuron at the
               destination
            # neuron's preferred direction
            src_rate_resp = (src_neuron.max_resp) * \
                gaussian(src_neuron.s_pref, dest_neuron.s_pref,
                    src_neuron.s_std) \
                + src_neuron.background_rate
            if src_neuron.is_inhibitory:
                inhibitory_background_input += weight *
                    src_neuron.background_rate * sim.dt
                inhibitory_background_rate +=
                    src_neuron.background_rate
            else:
                excitatory_background_input += weight *
                    src_neuron.background_rate * sim.dt
                excitatory_background_rate +=
                    src_neuron.background_rate
            connections.append([Connection(src_neuron, dest_neuron,
               weight, random.uniform(*excitatory_synaptic_delay) if
               weight > 0 else
```

```
                random.uniform(*inhibitory_synaptic_delay)),
                src_rate_resp])

        # NOTE: Since the sign of the weights must be mainted, we scale
            the
        # inhibitory connection weights to equal the excitatory weights
            at background
        # (i.e. ``partial'' normalization rather than equalization
        inhibitory_scaling = abs(excitatory_background_input) /
           abs(inhibitory_background_input)
        total_input = 0
        for connection, src_rate_resp in connections:
            if connection.weight <= 0:
                # This is inhibitory
                connection.weight = connection.weight *
                   inhibitory_scaling
            total_input += connection.weight * src_rate_resp * sim.dt
        assert total_input > 0
        for connection, src_rate_resp in connections:
            connection.weight *= abs(dest_neuron.recurrent_beta /
               total_input)
        write_connections_list([i[0] for i in connections],
           connections_file, pruning_constant)

if __name__ == '__main__':
    main()
```

7.4.1 Simulation Parameters

```
#!/usr/bin/env python
from math import pi

#---
# Simulation-Wide Parameters
#---
# Who many times longer than tau_psc should our filters be ?
psc_filter_multiplier = 5
sim_time = 10 # Seconds
sim_dt = 2.5e-4

#---
# Excitatory neuron (Pyramidal neurons)
#---
excitatory_tau_ref = [0.002, 0.005] # Seconds (2 ms)
excitatory_tau_rc = [0.010, 0.030] # Seconds (20 ms)
```

```
excitatory_tau_psc = [0.004, 0.006] # Seconds (5 ms)
excitatory_num_neurons = 5000
excitatory_max_resp_range = [40, 80]
excitatory_background_rate = [2, 4]
excitatory_spref_range = [0, 2*pi]
excitatory_std_range = [pi/8, pi/4]
excitatory_recurrent_beta = 0.5
excitatory_driving_beta = 0.5
excitatory_ff_beta = 0.0
excitatory_fb_beta = 0.0

#---
# Inhibitory interneurons (GABA-type)
#---
inhibitory_tau_ref = [0.002, 0.005] # Seconds (1 ms)
inhibitory_tau_rc = [0.010, 0.030] # Seconds (10 ms)
inhibitory_tau_psc = [0.004, 0.006] # Seconds (5 ms)
inhibitory_num_neurons = 5000
inhibitory_max_resp_range = [40, 80]
inhibitory_background_rate = [4, 8]
inhibitory_spref_range = [0, 2*pi]
inhibitory_std_range = [pi/4, pi/2]
inhibitory_recurrent_beta = 0.5
inhibitory_driving_beta = 0.5
inhibitory_ff_beta = 0.0
inhibitory_fb_beta = 0.0

#---
# Common parameters
#---
v_th = 1
r_leak = 1
# Use synaptic delays from Mattia et al. NeuroImage, 2010
inhibitory_synaptic_delay = [sim_dt, 0.003] # Seconds
excitatory_synaptic_delay = [sim_dt, 0.001] # Seconds

#---
# Trimming Factor
#---
pruning_constant = 1e-5
```

BIBLIOGRAPHY

M. Abeles, Y. Prut, H. Bergman, and E. Vaadia. Synchronization in neuronal transmission and its importance for information processing. *Prog. Brain Res.*, 102:395–404, 1994.

Y. Adini, D. Sagi, and M. Tsodyks. Excitatory-inhibitory network in the visual cortex: psychophysical evidence. *Proc. Natl. Acad. Sci. U.S.A.*, 94: 10426–10431, Sep 1997.

E. D. Adrian. The impulses produced by sensory nerve endings: Part I. *J. Physiol. (Lond.)*, 61:49–72, Mar 1926.

E. D. Adrian and Y. Zotterman. The impulses produced by sensory nerve-endings: Part II. The response of a Single End-Organ. *J. Physiol. (Lond.)*, 61:151–171, Apr 1926.

T. D. Albright. Direction and orientation selectivity of neurons in visual area MT of the macaque. *J. Neurophysiol.*, 52:1106–1130, Dec 1984.

T.D. Albright and R. Desimone. Local precision of visuotopic organization in the middle temporal area (MT) of the macaque. *Exp Brain Res*, 65:582–592, 1987.

B. Amirikian and A. P. Georgopoulos. Directional tuning profiles of motor cortical cells. *Neurosci Res*, 36(1):73–9, 2000.

R.A. Andersen. Neural mechanisms of visual motion perception in primates. *Neuron*, 18:865–872, Jun 1997.

Michael A. Arbib, editor. *The Handbook of Brain Theory and Neural Networks*. MIT Press, Cambridge, MA, USA, 2nd edition, 2002. ISBN 0262011972.

D. Attwell and C. Iadecola. The neural basis of functional brain imaging signals. *Trends Neurosci.*, 25:621–625, Dec 2002.

M. Avillac, S. Deneve, E. Olivier, A. Pouget, and J. R. Duhamel. Reference frames for representing visual and tactile locations in parietal cortex. *Nat. Neurosci.*, 8:941–949, Jul 2005.

P. A. Bandettini, A. Jesmanowicz, E. C. Wong, and J. S. Hyde. Processing strategies for time-course data sets in functional MRI of the human brain. *Magn Reson Med*, 30:161–173, Aug 1993.

Alain Barrat, Marc Barthlemy, and Alessandro Vespignani. *Dynamical Processes on Complex Networks*. Cambridge University Press, New York, NY, USA, 1st edition, 2008. ISBN 0521879507, 9780521879507.

A. Bartels, N. K. Logothetis, and K. Moutoussis. fMRI and its interpretations: an illustration on directional selectivity in area V5/MT. *Trends Neurosci.*, 31: 444–453, Sep 2008.

S. A. Beardsley and L. M. Vaina. Computational modelling of optic flow selectivity in MSTd neurons. *Network*, 9:467–493, Nov 1998.

S. A. Beardsley and L. M. Vaina. A laterally interconnected neural architecture in MST accounts for psychophysical discrimination of complex motion patterns. *J Comput Neurosci*, 10:255–280, 2001.

S. A. Beardsley and L. M. Vaina. A functional architecture for motion pattern processing in mstd. 2004.

S. A. Beardsley, R. L. Ward, and L. M. Vaina. A neural network model of spiral-planar motion tuning in MSTd. *Vision Res.*, 43:577–595, Mar 2003.

S. A. Beardsley, E. M. Sikoglu, and L. M. Vaina. A generalized mathematical model of feed-forward visual motion processing in the mt-mst complex. Unpublished manuscript, Unpublished.

C. Beck and H. Neumann. Interactions of motion and form in visual cortex - A neural model. *J. Physiol. Paris*, 104:61–70, 2010.

R. M. Birn, Z. S. Saad, and P. A. Bandettini. Spatial heterogeneity of the nonlinear

dynamics in the FMRI BOLD response. *Neuroimage*, 14:817–826, Oct 2001.

I. Bojak, T. F. Oostendorp, A. T. Reid, and R. Kotter. Connecting mean field models of neural activity to EEG and fMRI data. *Brain Topogr*, 23:139–149, Jun 2010.

James M. Bower and David Beeman. *The book of GENESIS (2nd ed.): exploring realistic neural models with the GEneral NEural SImulation System*. Springer-Verlag New York, Inc., New York, NY, USA, 1998. ISBN 0-387-94938-0.

G. M. Boynton, S. A. Engel, G. H. Glover, and D. J. Heeger. Linear systems analysis of functional magnetic resonance imaging in human V1. *J. Neurosci.*, 16:4207–4221, Jul 1996.

Valentino. Braitenberg and A. Schuz. *Anatomy of the cortex*. Springer-Verlag, Berlin, New York, 1991.

A. N. Burkitt. A review of the integrate-and-fire neuron model: II. Inhomogeneous synaptic input and network properties. *Biol Cybern*, 95:97–112, Aug 2006.

Nicholas T. Carnevale and Michael L. Hines. *The NEURON Book*. Cambridge University Press, February 2006. ISBN 0521843219.

B. Cauli, X. K. Tong, A. Rancillac, N. Serluca, B. Lambolez, J. Rossier, and E. Hamel. Cortical GABA interneurons in neurovascular coupling: relays for subcortical vasoactive pathways. *J. Neurosci.*, 24:8940–8949, Oct 2004.

F. S. Chance, L. F. Abbott, and A. D. Reyes. Gain modulation from background synaptic input. *Neuron*, 35:773–782, Aug 2002.

S. Coombes. Large-scale neural dynamics: simple and complex. *Neuroimage*, 52: 731–739, Sep 2010.

L.N. Cooper and C.L. Scofield. Mean-field theory of a neural network. *Proc. Natl. Acad. Sci. U.S.A.*, 85:1973–1977, Mar 1988.

S. Corchs and G. Deco. Large-scale neural model for visual attention: integration of experimental single-cell and fMRI data. *Cereb. Cortex*, 12:339–348, Apr 2002.

A. M. Dale and R. L. Buckner. Selective averaging of rapidly presented individual trials using fMRI. *Hum Brain Mapp*, 5:329–340, 1997.

Peter Dayan and L. F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press, 1st edition, December 2001.

G. Deco, V. K. Jirsa, P. A. Robinson, M. Breakspear, and K. Friston. The dynamic brain: from spiking neurons to neural masses and cortical fields. *PLoS Comput. Biol.*, 4:e1000092, 2008.

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

S. Deneve, P. E. Latham, and A. Pouget. Efficient computation and cue integration with noisy population codes. *Nat. Neurosci.*, 4:826–831, Aug 2001.

Markus Diesmann, Max planck Inst FÃijr StrÃűmungsforschung, and Marc oliver Gewaltig. Nest: An environment for neural systems simulations. In *In T. Plesser and V. Macho (Eds.), Forschung und wisschenschaftliches Rechnen, Beitrage zum Heinz-Billing-Preis 2001, Volume 58 of GWDG-Bericht*, pages 43–70, 2002.

R. J. Douglas, C. Koch, M. Mahowald, K. A. Martin, and H. H. Suarez. Recurrent excitation in neocortical circuits. *Science*, 269:981–985, Aug 1995.

C. J. Duffy and R. H. Wurtz. Sensitivity of MST neurons to optic flow stimuli. I. A continuum of response selectivity to large-field stimuli. *J. Neurophysiol.*, 65: 1329–1345, Jun 1991.

Chris Eliasmith and Charles H. Anderson. *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. The MIT Press, Cambridge, 2002.

D. J. Felleman and D. C. Van Essen. Distributed hierarchical processing in the primate cerebral cortex. *Cereb. Cortex*, 1:1–47, 1991.

A. Fergus and K. S. Lee. GABAergic regulation of cerebral microvascular tone in the rat. *J. Cereb. Blood Flow Metab.*, 17:992–1003, Sep 1997.

R. Fitzhugh. Impulses and Physiological States in Theoretical Models of Nerve Membrane. *Biophys. J.*, 1:445–466, Jul 1961.

M. S. Graziano, G. S. Yap, and C. G. Gross. Coding of visual space by premotor neurons. *Science*, 266:1054–1057, Nov 1994.

B.S. Gutkin and G.B. Ermentrout. Dynamics of membrane excitability determine interspike interval variability: a link between spike generation mechanisms and cortical spike train statistics. *Neural Comput*, 10:1047–1065, Jul 1998.

D. J. Heeger, G. M. Boynton, J. B. Demb, E. Seidemann, and W. T. Newsome. Motion opponency in visual cortex. *J. Neurosci.*, 19:7162–7174, Aug 1999.

D. J. Heeger, A. C. Huk, W. S. Geisler, and D. G. Albrecht. Spikes versus BOLD: what does neuroimaging tell us about neuronal activity? *Nat. Neurosci.*, 3: 631–633, Jul 2000.

Rolf Hempel. *The MPI Standard for Message Passing*. Springer-Verlag, London, UK, 1994.

John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the theory of neural computation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1991.

D.J. Herzfeld and S.A. Beardsley. Improved multi-unit decoding at the brain-machine interface using population temporal linear filtering. *J Neural Eng*, 7:046012, Aug 2010.

A. L. Hodgkin and A. F. Huxley. Propagation of electrical signals along giant nerve fibers. *Proc. R. Soc. Lond., B, Biol. Sci.*, 140:177–183, Oct 1952.

E. M. Izhikevich. Simple model of spiking neurons. *IEEE Trans Neural Netw*, 14: 1569–1572, 2003.

E. M. Izhikevich and G. M. Edelman. Large-scale model of mammalian thalamocortical systems. *Proc. Natl. Acad. Sci. U.S.A.*, 105:3593–3598, Mar 2008.

E. Juergens, A. Guettler, and R. Eckhorn. Visual stimulation elicits locked and induced gamma oscillations in monkey intracortical- and EEG-potentials, but not in human EEG. *Exp Brain Res*, 129:247–259, Nov 1999.

S. G. Kim, W. Richter, and K. U?urbil. Limitations of temporal resolution in functional MRI. *Magn Reson Med*, 37:631–636, Apr 1997.

Christof Koch. *Biophysics of Computation: Information Processing in Single Neurons (Computational Neuroscience)*. Oxford University Press, USA, October 2004. ISBN 0195181999.

S. Koyama and R.E. Kass. Spike train probability models for stimulus-driven leaky integrate-and-fire neurons. *Neural Comput*, 20:1776–1795, Jul 2008.

Lai-Wo Stan Leung. Field potentials in the central nervous system. In Wolfgang Walz, Alan A. Boulton, Glen B. Baker, and Case H. Vanderwolf, editors, *Neurophysiological Techniques*, volume 15 of *Neuromethods*, pages 277–312. Humana Press, 1991.

M.T. Lippert, T. Steudel, F. Ohl, N.K. Logothetis, and C. Kayser. Coupling of neural activity and fMRI-BOLD in the motion area MT. *Magn Reson Imaging*, 28:1087–1094, Oct 2010.

Z. Liu, C. Rios, N. Zhang, L. Yang, W. Chen, and B. He. Linear and nonlinear relationships between visual stimuli, EEG and BOLD fMRI signals. *Neuroimage*, 50:1054–1066, Apr 2010.

N. K. Logothetis. What we can do and what we cannot do with fMRI. *Nature*, 453: 869–878, Jun 2008.

N. K. Logothetis, J. Pauls, M. Augath, T. Trinath, and A. Oeltermann. Neurophysiological investigation of the basis of the fMRI signal. *Nature*, 412: 150–157, Jul 2001.

N.K. Logothetis. The underpinnings of the BOLD functional magnetic resonance imaging signal. *J. Neurosci.*, 23:3963–3971, May 2003.

M. Mattia, S. Ferraina, and P. Del Giudice. Dissociated multi-unit activity and local field potentials: a theory inspired analysis of a motor decision task. *Neuroimage*, 52:812–823, Sep 2010.

A. Mazzoni, S. Panzeri, N.K. Logothetis, and N. Brunel. Encoding of naturalistic stimuli by local field potential spectra in networks of excitatory and inhibitory neurons. *PLoS Comput. Biol.*, 4:e1000239, Dec 2008.

C. Meunier and I. Segev. Playing the devil's advocate: is the Hodgkin-Huxley model useful? *Trends Neurosci.*, 25:558–563, Nov 2002.

U. Mitzdorf. Current source-density method and application in cat cerebral cortex: investigation of evoked potentials and EEG phenomena. *Physiol. Rev.*, 65: 37–100, Jan 1985.

D. W. Moran and A. B. Schwartz. Motor cortical representation of speed and direction during reaching. *J Neurophysiol*, 82(5):2676–92, 1999.

V. B. Mountcastle, P. W. Davies, and A. L. Berman. Response properties of neurons of cat's somatic sensory cortex to peripheral stimuli. *J. Neurophysiol.*, 20:374–407, Jul 1957.

J. A. Movshon and W. T. Newsome. Visual response properties of striate cortical neurons projecting to area MT in macaque monkeys. *J. Neurosci.*, 16: 7733–7741, Dec 1996.

A. Nabi and J. Moehlis. Time optimal control of spiking neurons. Jun 2011.

D.G. Nair. About being BOLD. *Brain Res. Brain Res. Rev.*, 50:229–243, Dec 2005.

W. T. Newsome and E. B. Pare. A selective impairment of motion perception following lesions of the middle temporal visual area (MT). *J. Neurosci.*, 8: 2201–2211, Jun 1988.

D. Noble and R. B. Stein. The threshold conditions for initiation of action potentials by excitable cells. *J. Physiol. (Lond.)*, 187:129–162, Nov 1966.

S. J. Nowlan and T. J. Sejnowski. A selection model for motion processing in area MT of primates. *J. Neurosci.*, 15:1195–1214, Feb 1995.

D. Q. Nykamp and D. Tranchina. A population density approach that facilitates large-scale modeling of neural networks: extension to slow inhibitory synapses. *Neural Comput*, 13:511–546, Mar 2001.

S. Ohlendorf, A. Sprenger, O. Speck, S. Haller, and H. Kimmig. Optic flow stimuli in and near the visual field centre: a group FMRI study of motion sensitive regions. *PLoS ONE*, 3:e4043, 2008.

L. Paninski, M. R. Fellows, N. G. Hatsopoulos, and J. P. Donoghue. Spatiotemporal tuning of motor cortical neurons for hand position and velocity. *J Neurophysiol*, 91(1):515–32, 2004.

M. Paolini, C. Distler, F. Bremmer, M. Lappe, and K.P. Hoffmann. Responses to continuously changing optic flow in area MST. *J. Neurophysiol.*, 84:730–743, Aug 2000.

L. D. Partridge. A possible source of nerve signal distortion arising in pulse rate encoding of signals. *J. Theor. Biol.*, 11:257–281, Jul 1966.

L. Pellerin and P.J. Magistretti. Glutamate uptake into astrocytes stimulates aerobic glycolysis: a mechanism coupling neuronal activity to glucose utilization. *Proc. Natl. Acad. Sci. U.S.A.*, 91:10625–10629, Oct 1994.

K.H. Pettersen, E. Hagen, and G.T. Einevoll. Estimation of population firing rates and current source densities from laminar electrode recordings. *J Comput Neurosci*, 24:291–313, Jun 2008.

A. Pouget, Terrence, and T. J. Sejnowski. Spatial representations in the parietal cortex may use basis functions. In *Advances in Neural Information Processing Systems 7*, pages 157–164. MIT Press, 1995.

M. Raffi and R.M. Siegel. A functional architecture of optic flow in the inferior parietal lobule of the behaving monkey. *PLoS ONE*, 2:e200, 2007.

M. Rasch, N.K. Logothetis, and G. Kreiman. From neurons to circuits: linear estimation of local field potentials. *J. Neurosci.*, 29:13785–13796, Nov 2009.

G. Rees, K. Friston, and C. Koch. A direct quantitative relationship between the functional properties of human and macaque V5. *Nat. Neurosci.*, 3:716–723, Jul 2000.

A. Roberts and B. M. H. Bush. Neurones without impulses. *Experimental Physiology*, 66(3):347–348, 1981.

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

A. Shmuel, M. Augath, A. Oeltermann, and N. K. Logothetis. Negative functional MRI response correlates with decreases in neuronal activity in monkey visual area V1. *Nat. Neurosci.*, 9:569–577, Apr 2006.

B. Stefanovic, J. M. Warnking, and G. B. Pike. Hemodynamic and metabolic responses to neuronal inhibition. *Neuroimage*, 22:771–778, Jun 2004.

B. E. Stein, M. A. Meredith, and M. T. Wallace. The visually responsive neuron and beyond: multisensory integration in cat and monkey. *Prog. Brain Res.*, 95: 79–90, 1993.

S. M. Stringer, E. T. Rolls, T. P. Trappenberg, and I. E. de Araujo. Self-organizing continuous attractor networks and path integration: two-dimensional models of place cells. *Network*, 13:429–446, Nov 2002a.

S. M. Stringer, T. P. Trappenberg, E. T. Rolls, and I. E. de Araujo. Self-organizing continuous attractor networks and path integration: one-dimensional models of

head direction cells. *Network*, 13:217–242, May 2002b.

K. Tanaka, Y. Fukada, and H. A. Saito. Underlying mechanisms of the response specificity of expansion/contraction and rotation cells in the dorsal part of the medial superior temporal area of the macaque monkey. *J. Neurophysiol.*, 62: 642–656, Sep 1989.

A. Thielscher and H. Neumann. Globally consistent depth sorting of overlapping 2D surfaces in a model using local recurrent interactions. *Biol Cybern*, 98:305–337, Apr 2008.

J.D. Touboul and G.B. Ermentrout. Finite-size and correlation-induced effects in mean-field dynamics. Mar 2011.

M. Ursino and G.E. La Cara. Travelling waves and EEG patterns during epileptic seizure: analysis with an integrate-and-fire neural network. *J. Theor. Biol.*, 242:171–187, Sep 2006.

S. Wakde. Asymmetric transfer of task-dependent perceptual learning in visual motion processing. Master's thesis, Marquette University, Milwaukee, Wisconsin, June 2011.

R. Wang. A simple competitive account of some response properties of visual neurons in area MSTd. *Neural Comput*, 7:290–306, Mar 1995.

J. T. Weber, C. I. De Zeeuw, D. J. Linden, and C. Hansel. Long-term depression of climbing fiber-evoked calcium transients in Purkinje cell dendrites. *Proc. Natl. Acad. Sci. U.S.A.*, 100:2878–2883, Mar 2003.

X. S. Wu, J. Y. Sun, A. S. Evers, M. Crowder, and L. G. Wu. Isoflurane inhibits transmitter release and the presynaptic action potential. *Anesthesiology*, 100: 663–670, Mar 2004.

X. Xie, R. H. Hahnloser, and H. S. Seung. Double-ring network model of the head-direction system. *Phys Rev E Stat Nonlin Soft Matter Phys*, 66:041902, Oct 2002.

R. S. Zemel and T. J. Sejnowski. A model for encoding multiple object motions and self-motion in area MST of primate visual cortex. *J. Neurosci.*, 18:531–547, Jan 1998.