University of Nevada, Reno

# Computational Neuroscience: Theory, Development and Applications in Modeling The Basal Ganglia

A dissertation submitted in partial fulfillment of the
the requirements for the degree Doctor of Philosophy in
Biomedical Engineering

by

Corey M. Thibeault

Dr. Frederick C. Harris, Jr. / Dissertation Advisor
and
Dr. Narayan Srinivasa / Dissertation Co-Advisor

December, 2012


THE GRADUATE SCHOOL

We recommend that the dissertation
prepared under our supervision by

**COREY M. THIBEAULT**

entitled

**Computational Neuroscience: Theory, Development And Applications In Modeling The Basal Ganglia**

be accepted in partial fulfillment of the
requirements for the degree of

**DOCTOR OF PHILOSOPHY**

Frederick C. Harris, Jr., Ph. D., Advisor

Narayan Srinivasa, Ph. D., Committee Member

James Kenyon, Ph. D., Committee Member

Michael Crognale, Ph. D., Committee Member

Nelson Publicover, Ph. D., Committee Member

Normand Leblanc, Ph. D., Graduate School Representative

Marsha H. Read, Ph. D., Dean, Graduate School

December, 2012

# Abstract

Inspiring entire computing paradigms, hardware platforms and theories of nervous system function, the field of computational neuroscience has grown steadily since its emergence in the mid-1980s. The motivation behind it is to mathematically describe the nervous system in terms of how the structures process information. Simulating the brain this way can be done at varying levels of abstraction and biological realism; providing insight into the function of the nervous system or supporting empirical evidence. This dissertation presents a snapshot of the computational neuroscience landscape. It begins begins with the mathematical theory, moving to implementation, and finally ending with its application. It is by no means a complete picture but provides a basic understanding of how mathematical modeling contributes to neuroscience.

This begins by presenting the design considerations behind high-performance neural simulation environments. A concept which is then extended with a novel implementation for the exchange of spiking information in high-performance cluster environments. A framework for creating virtual environments for embodied modeling is then developed and discussed. Finally, a toolkit for efficiently analyzing the large amounts of data generated by these spiking models is presented.

Once these tools are established the focus is shifted to models of the basal ganglia. After a brief background, spiking models capable of action-selection through reinforce-

ment learning are described. These borrow from the basal ganglia but are developed for implementation on neuromorphic hardware and are therefore necessarily simplified. The networks are embodied in virtual environments and their performance based on two tasks is explored under varying conditions. Finally, the use of a simple hybrid neuron is explored in several published models of the basal ganglia; demonstrating the first example of a hybrid neuron in biologically faithful models of the basal ganglia.

To Dashiel and Amber without whose never-ending support, encouragement, and dirty diapers this dissertation would have been finished years ago and for half the cost.

To my Grandmother Barbara Smith who, despite her desire for me to get a real job, still supported me both emotionally and financially.

To Phil Goodman who, although the only thing we could ever agree on was scotch, I would never have gotten to this point without.

# Acknowledgments

There are a lot of people who have helped me on my way to finishing this dissertation. First and foremost, my wife Amber and son Dashiel have been incredibly supportive; putting up with my absences due to my research as well as my frequent mental absences, incidentally also due to my research.

My mother, Lorre, has been a consistent source of encouragement and cheerleading, almost to the point of annoyance; however, it was all greatly appreciated.

My father and stepmother, John and Laura, were encouraging and positive through this process; even genuinely pretending to care about the minutia of graduate school life.

My grandfather, Stanley B. Smith, Jr., who I have idolized since I was old enough to realize computers were cool, has been a never-ending source of intellectual and emotional support. Words cannot express how much that has meant to me.

My grandmother, Barbara Smith, who passed away during my graduate study, was a phenomenal presence in my life. Her absence during these times is difficult but her support of me has never been forgotten.

Academically, there have been several people that have supported and guided me through graduate school. While at MNSU, Dr. Patrick Tebbe encouraged me to pursue graduate education and helped me prepare for it. Similarly, Dr. Jonathan Page exposed me to the world of scientific research, nurturing my curiosity and humoring many of my bad experi-

ment ideas. Without the support of these professors I would never have made it to UNR.

My career at UNR began with Dr. Phil Goodman in the Brain Lab. He was responsible for my admittance, exposing me to computational neuroscience, and continually challenging me intellectually. His passing in 2010 has left a huge whole in the field of neuroscience and in all who knew him.

My experience in the UNR Brain Lab was a truly positive one. Thanks are due to the many people I had the pleasure of working with there, including John Kenyon, Gareth Ferneyhough, Casey Howard, Josh Hegie, Sridhar Anumandla, Kevin Cassiday, Bryce Prescott, Rashid Makhmudov, Nick Ceglia, Mathias Quoy, Roger Hoang and Laurence Jayet Bray, as well as the invaluable interactions I had with people from many other labs, such as Ramon Ayon, Chad Feller, Del Jackson, Cody White, Matt Sgambati, and Joe Mahsman.

My graduate committee members at UNR have been tremendous assets both during my graduate work and my dissertation. I can only hope to be half as successful as each of them has been.

I only had the opportunity to listen to Dr. Micheal Crognale speak on a few occasions. Despite not really knowing me, he was kind enough to serve on my committee and provided pointed questions and suggestions that were invaluable during my qualifier and defense.

I had the pleasure of learning electrophysiology from Dr. James Kenyon; he was even kind enough to lecture in the first class I taught. I have been continually amazed at the depth of his knowledge and I consider having him serve on my committee as a distinction.

Dr. Normand Leblanc taught me nearly everything I know about ion channels and excitability in cells. During the year before I came to HRL he provided me with lab space and my own patch clamp setup. I still consider working in his lab as a highlight of my time at UNR. The combination of computational science with empirical science should never be

dismissed; that experience has cemented that belief for me.

Dr. Nelson Publicover has acted as teacher, department head and committee member for me during my time at UNR. Whether teaching how to objectively judge what good science is or counseling me during those inevitable periods of academic uncertainty, he has gone out of his way to make a positive impact. I have encountered very few people who have been so successful in doing that.

Dr. Frederick C. Harris, Jr. who stepped in as my adviser and director of the Brain Lab has been a constant advocate. He not only taught me a tremendous amount about high-performance computing but also the politics of research and subtleties of grant writing. He supported my research when no one else at UNR would and supported my move to HRL even though it negatively impacted him.

I am forever grateful to my other adviser Dr. Narayan Srinivasa of HRL Labs, who not only employed me to do this work but let me define what it was going to be and trusted me to complete it. The subtle suggestions and encouragement were the right amount of guidance I needed to finish this dissertation (and let me think I was actually in control of it).

There are numerous people at HRL who have helped me complete this work, either through useful discussions (Mike O'Brien) or literally hundreds of hours of coding and debugging (Kirill Minkovich). In addition, the efforts of Son Dao and Roy Matic are also appreciated; both went out of their way to push numerous papers, presentations and the over 300 pages of this dissertation through the clearance processes at HRL.

I am grateful to the DARPA SyNAPSE program manager Dr. Gil Pratt who reviewed this entire dissertation in a very short period of time. In addition, I am indebted to Robert Allen of DARPA, without whom this work never would have made it through the bureaucracy of government clearance.

Finally, I gratefully acknowledge the support for this work by the Defense Advanced Research Projects Agency (DARPA) SyNAPSE grant HRL0011-09-C-001. This work was performed under the DARPA SyNAPSE grant HR0011-09-C-001. This work is approved for public release and distribution is unlimited. The views, opinions, and/or findings contained in this dissertation are those of the author and should not be interpreted as representing the official views or policies, either expressed or implied, of the DARPA or the Department of Defense.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Imagine for a moment how you would set about building a modern airplane. The process may begin with some design diagrams, or sketches and perhaps lead to scaled models. You have mathematical principles you apply to ensure your wing design generates enough lift and that the static structures can withstand those forces. Fatigue analysis will give an idea of how long parts will last. You can even model the likelihood of your airplane being hijacked (Holden, 1986). Modern principles of engineering and software make it possible to build, analyze and fly your airplane entirely in model space. This is exactly how Boeing designed the 777 and revolutionized aerospace design (Norris, 1996). But, despite everything we know about neuroscience and physiology, this kind of design feat is unobtainable with biological systems.

The roundworm, *c. elegans*, has a nervous system of roughly $300$ neurons and $5,000$ synapses. However, we still cannot put $300$ model neurons together and simulate the behavior of a roundworm. It is not just that the complexity of the nervous system is too much, it is that there is still too much that is unknown at the single cell and network levels. This is not meant to paint a grim picture of computational neuroscience, but rather frame its role

in the context of neuroscience research as a whole.

Computational neuroscience encompasses a set of tools that are complementary to electrophysiological, biophysical and behavioral research. This multifaceted approach is likely the only way the complexities of the nervous system will be elucidated. Modern research techniques are only capable of revealing small aspects of neural function. Unfortunately, these small parts are not enough to tell the whole story. The goal of computational neuroscience is to help fill in those gaps and point empirical studies towards new directions. In basic sciences and medicine its role can be distilled into:

- A predictive tool, to guide new experiments.
- To demonstrate the plausibility of a theory.
- To quantitatively compliment experimentation.

It is important to point out that all models are wrong. As good as the model of the Boeing 777 was, it was still just an abstract representation of the real thing. In mathematical simulation the goal is to create a model that encompasses enough of the dynamics of the real system that is required to support a hypothesis. In fact, too much unnecessary detail may obfuscate the important details of a system.

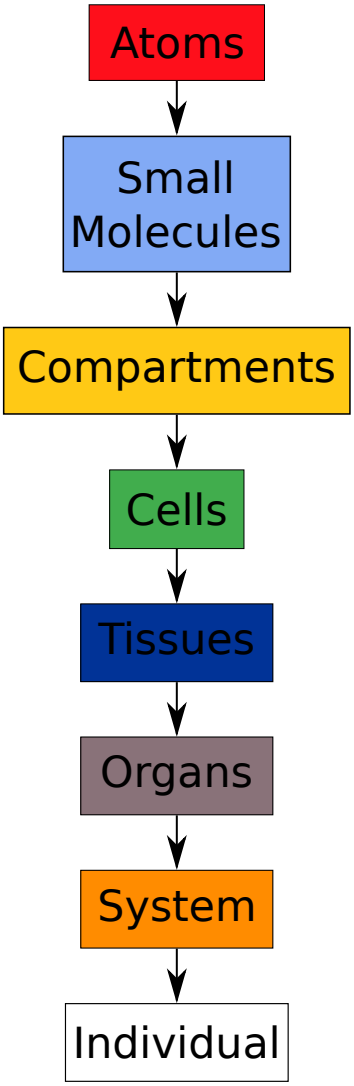



**Figure 1.1:** Levels of modeling abstraction

We can study biological phenomena at different levels of abstraction; Figure 1.1, illustrates these. Choosing how detailed a model is depends on a number of factors that include the hypothesis being tested, computing resources, and current understanding of the biological process. The models presented in this work fall in the single cell and organ levels. In addition, all of these are dynamic processes. Dynamic phenomena are anything that changes over time and almost everything in living organisms is dynamic. For example:

- Cell growth
- Cell division
- Intercellular communication
- Movement
- Electrical activity

These systems, that are never in static thermal equilibrium, maintain a constant movement of ions and chemical species across cell membranes (Fall et al., 2002). These can be described with differential equations but all contain nonlinearities. Unfortunately, those nonlinearities make it difficult or impossible to obtain a solution analytically. In these instances, numerical methods can provide a mechanism for obtaining accurate approximations. This dissertation looks at the differential equations that describe neurons, how those can be simulated and how the resulting models can be used for neuroscience research.

The approach to computational neuroscience presented here is decidedly bottom-up. We begin with a single neuron model and its dynamics. This single neuron is then connected to other neurons and the interactions between them are modeled and simulated. The dynamic changes in activity and structure are then analyzed and in some cases linked to behavior. In others, theories of how the functional anatomy contributes to activity is established. These models borrow from, and build on, other aspects of neuroscience.

Part I of this work explores some of the techniques and tools that are essential to researching large-scale computational models. We begin by exploring the strategies for developing distributed simulation environments. This is the perfect intersection of software and biomedical engineering; as the requirements for accurately simulating biological systems are in constant competition with the need for efficiency in their calculation. Chapter 3 explores hardware specific strategies for maximizing efficiency with accuracy. In addition, Chapter 4 takes this a step further by exploring the exchange of information between distributed compute elements. Here, we present an analysis of some current methods and

propose a novel implementation of spike exchange.

Developing the simulation environment is just the first step. Once the modeling capabilities are established, researchers need to interface with them both during and after a simulation. In Chapter 5, we present a design for embodying these neural models in stimulus rich environments. This provides a mechanism for generating input stimulus in both static and dynamic ways. Finally, a new tool-set is developed for analyzing the large amounts of spiking information generated during the simulation of large-scale models. This is the focus of Chapter 6.

Armed with a complete set of computational neuroscience tools we set out to explore models of the basal ganglia with two different approaches in Part II. Chapter 7 presents a brief background of the mammalian basal ganglia as well as some of the computational models of it. In Chapter 8 we then develop models that replicate some of the known higher-level function of the basal ganglia without strictly following the functional anatomy. This work is framed in the context of embodied modeling in neuromorphic hardware and the networks are tasked with learning to play two virtual games.

Chapter 9 takes a different approach by replicating as much of the physiology and dynamics of the basal ganglia as possible using a simple hybrid neuron. These replicate the results of several published models, and in some cases improve them, but require far less computational resources than the original. This reduction in complexity is exciting on a number of different levels. Not only does this validate the results of the original works, but the use of the hybrid neuron as well. In addition, these models are much more amenable to hardware implementations and the reduction in computational complexity opens the door for more thorough analysis of the models. Finally, some future directions and applications for this research are presented in Chapter 10.

Although there are some cases of overlap, the chapters in this dissertation are written to

be self-contained. These can realistically be viewed in any order and readers familiar with the subject can skip the background sections.

# Chapter 2

# Mathematical Neuroscience

Excitability in cells is an awkward concept for people outside of the physical sciences. Many of us go through our lives perfectly happy with the computer analogy, where our nervous system can be abstracted as computing elements connected together through cables that transmit electrical signals. For those of us who understand computer architecture, this is a comforting comparison. In fact, in science and engineering that juxtaposition is essential for analyzing complex systems. Mechanical structures can be reduced to springs for treatment with finite element analysis. Fluid dynamics can be modeled by electrical circuits. Even blood-flow in the lungs can be simulated using electrical components. So it is no surprise that the inadequacy of the computer analogy is startling to an engineer.

This chapter presents a very broad background that is required to not only understand the concepts presented in this dissertation but also purge the computer analogy from the readers mind. This is noticeably incomplete and can in no way replace an introductory neuroscience text. In addition, the focus is on point neurons, where a neuron is assumed to be a single point in space and only the active transmission of electrical signals is modeled. This simplification is required to model large-scale systems in feasible time-frames but

ignores the more complex interactions within the neuron.

## 2.1 Excitable cells

All cells have a voltage potential across their plasma membrane (the semi-permeable lipid bilayer surrounding the cell). This voltage potential is a product of differences in ionic concentrations between the fluid inside of the cell (cytoplasm) and the extracellular fluid outside of the cell. The unequal ion distribution results in a negative charge inside, compared to the outside of the cell. Typically that potential ranges from $-50$ to $-100$ millivolts.

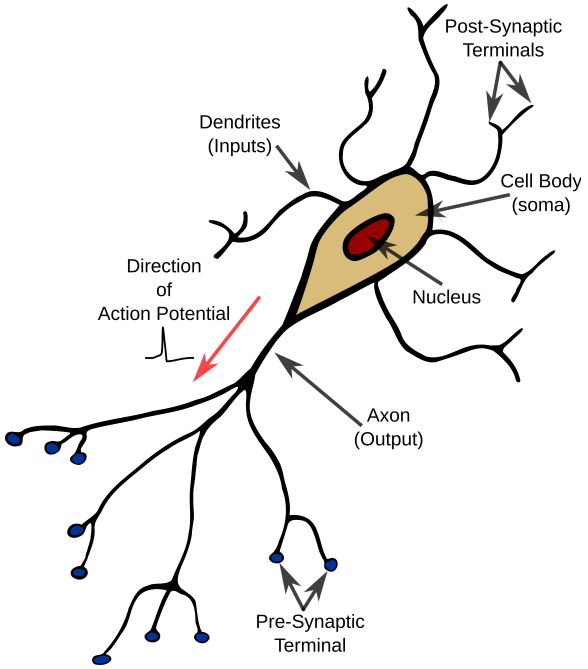



**Figure 2.1:** The prototypical neuron.

## 2.2 Ionic basis of excitability

Excitable cells are characterized by a dynamic change in membrane voltage that generates an action potential. This is facilitated by a voltage dependent change in membrane permeability and the passive diffusion of ions down their electrochemical gradient. Our description of cellular excitation begins with that electrochemical gradient.

Diffusion is the tendency of molecules to separate from each other in space. Individually, these molecules move randomly but the bulk molecular motion can have direction. When those molecules have an electrical charge associated with them there is an additional repelling force. The combination of these forces drive the diffusion of ions across a membrane.

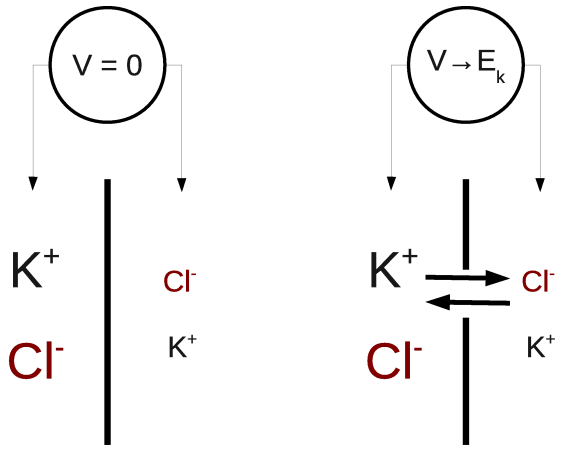



**Figure 2.2:** Diffusion through a selectively permeable membrane. A container is separated by an artificial membrane. Each side contains different concentrations of an electrically neutral $K^+$ chloride solution. Initially, left, the membrane is impermeable. The membrane is then made selectively permeable to $K^+$ only, right, and the excess of positive charge on the right side of the container creates a voltage potential that approaches the Nernst potential of $K^+$, $E_k$.

The electrical potential arises due to the selective permeability of the membrane. Consider Figure 2.2 left, where two aqueous solutions are separated by an artificial membrane. The ionic concentrations on each side are electrically neutral, so each positive $K^+$ ion is

matched by a negatively charged $Cl^-$ ion. The membrane is not permeable to either ion so there is no net flux or potential across it.

If the membrane is made permeable only to $K^+$, Figure 2.2 right, there will be a net flux of $K^+$ down its chemical gradient from the left to the right side of the container. As a positive $K^+$ ion moves across the membrane it is separated from its corresponding chloride anion. The result is a net positive charge on the right side of the container. Eventually, this excess positive charge will resist additional cations from entering the right side and an equilibrium point is reached. This equilibrium is where the diffusive force balances the electrical force and is called the Nernst potential.

The Nernst potential can be derived either through statistical mechanics, using the Boltzmann equation, or with statistical thermodynamics, using the Gibbs free energy (Keener and Sneyd, 2008). The resulting equation describes the equilibrium potential for a charged ion based on the concentrations inside and outside of a cell. It is defined by

$$\Delta E = \frac{RT}{zF} ln \left| \frac{\eta_{out}}{\eta_{in}} \right|. \tag{2.1}$$

Where $\eta_{in}$ and $\eta_{out}$ are the internal and external concentrations respectively, $R$ is the universal gas constant, $T$ is the temperature in Kelvin, $F$ is Faraday's constant and $z$ is the valence of the ion. This is also referred to as the reversal potential because it is the point where the bulk ion movement will reverse directions.

A key aspect of this equation, which has been overlooked up to this point, is the concentration difference between the inside and outside of the cell. This gradient is established and maintained through active diffusion mechanisms. Using energy, cells transport ions against their concentration gradient to create the electrical potentials. Table 2.1 presents the four major ion species in neurons as well as their corresponding concentration gradients and

Nernst potentials (Purves et al., 2007).

**Table 2.1:** Ionic concentrations for typical neurons

| | **Intracellular** $(\mathbf{mM})$ | **Extracellular** $(\mathbf{mM})$ | $\mathbf{E_x(mV)}$ |
|---|---|---|---|
| $\mathbf{K}^+$ | $140$ | $5$ | $-87$ |
| $\mathbf{Na}^+$ | $5-15$ | $145$ | $+60$ to $+88$ |
| $\mathbf{Cl}^-$ | $4-30$ | $110$ | $-88$ to $-60$ |
| $\mathbf{Ca^{2}}^+$ | $0.0001$ | $1-2$ | $+200$ |

As can be seen in Table 2.1 each ion contributes a different value to the membrane voltage. If each ion contributed equally the resting potential of a cell would be around $85$ mV. However, all cells maintain a resting potential that is more negative on the inside compared to the outside of the cell. The reason is that the Nernst potential assumes perfect conductance to an ionic species. Physically the membrane conductance to an ion dynamically changes and at rest is in a partially conducting state.

To calculate the resting potential of a cell that is permeable to multiple ions we employ the Goldman-Hodgkin-Katz (GHK) equation (Fall et al., 2002). This calculates the contribution of multiple ionic gradients and their respective conductances to the membrane potential. This is defined as

$$E_m = -\frac{RT}{F} ln \left( \frac{P_{Na}[Na^+]_i + P_K[K^+]_i + P_{Cl}[Cl^-]_e}{P_{Na}[Na^+]_e + P_K[K^+]_e + P_{Cl}[Cl^-]_i} \right). \tag{2.2}$$

Where $E_m$ is the membrane potential, $P_x$ is the relative permeability of the membrane to ion $x$, and $[X]$ is the concentration of ion $x$. The GHK equation is important for accurately calculating the membrane potential but it is difficult to use in practice. In addition, obtaining the measurements required for calculating the relative permeability is arduous and are themselves approximations. The ionic conductance is an acceptable substitute that is easier to obtain experimentally.

We can simplify the analysis of excitable cells considerably by assuming that the membrane electrochemical physics obey Ohm's law. From that, we can calculate the conductance using

$$G = \frac{I}{V}. \tag{2.3}$$

The current ($I$) and voltage ($V$) of the cell are now easily obtained using standard electrophysiological techniques. Similar to the GHK equation, the contribution of an ionic species to the membrane potential can be found using a weighted sum,

$$E_m = \frac{\sum_i g_i \cdot E_i}{\sum_i g_i}. \tag{2.4}$$

Where $i$ is the ion species, $E_i$ is the Nernst potential of that species and $g_i$ is the conductance. This is the parallel conductance model for the membrane (Fall et al., 2002). It can be simplified by rewriting Equation 2.4 as

$$V_m = \sum_i G_i^* \cdot E_i, \tag{2.5}$$

where

$$G_x^* = \frac{G_x}{\sum_i G_i}.$$

For an example cell with sodium ($Na^+$), potassium ($K^+$), and chloride ($Cl^-$) currents the membrane voltage would be found using

$$V_m = \frac{(g_{Na} \cdot E_{Na}) + (g_k \cdot E_k) + (g_{Cl} \cdot E_{Cl})}{g_{Na} + g_k + g_{Cl}}.$$

For the type of neuron models used in this work, our interest is in how the conductance of the membrane to different ionic species changes with time. These dynamic systems become more tractable with the assumption of ohmic properties presented above. Using that, the cell membrane can be described as an electrical circuit; with the components analogies:

1. **Phospholipid Bilayer:** Creates a capacitive effect where ionic charge accumulates across the nanometer thick Debye layer (Fall et al., 2002).
2. **Ionic permeability of the Membrane:** Acts like a resistor, impeding the ionic current.
3. **Electrochemical driving forces:** Establishes an ionic battery driving each species.

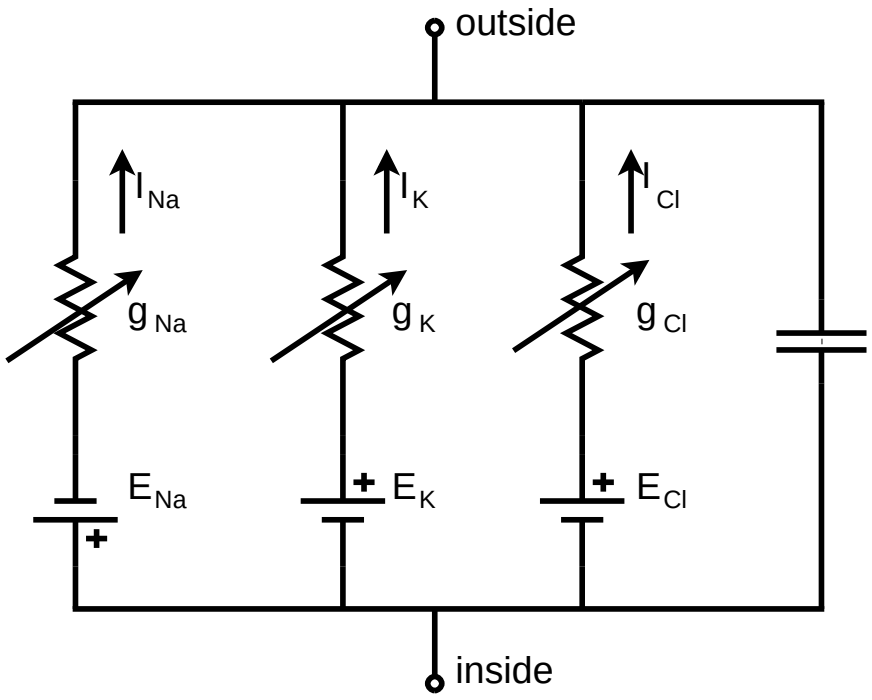



**Figure 2.3:** Cell membrane equivalent circuit diagram (adapted from Fall et al. (2002) and Keener and Sneyd (2008).

The diagram in Figure 2.3 illustrates the equivalent electrical circuit for a cell with three conducting species. However, the interesting aspect of excitable cells is not the steady-

state membrane potential calculations presented above but the dynamic changes required to reach steady-state. Continuing with the assumption of Ohmic properties and that the ionic battery remains constant, the current across the membrane for potassium now takes the form

$$I_K = g_K(V - E_K). \tag{2.6}$$

With the contribution of multiple ionic currents calculated using

$$I_{ion} = \sum_i I_i = \sum_i g_i(V - E_i). \tag{2.7}$$

The capacitive current across the membrane can be written as

$$I_{cap} = C_m \frac{dV}{dt}. \tag{2.8}$$

Applying Kirchoff's law of charge conservation to the circuit

$$I_{cap} + I_{ion} = 0.$$

Finally, substituting in we arrive at

$$C_m \frac{dV}{dt} = -\sum_i g_i(V - E_i).$$

Equipped with a representation of a cross section of cellular membrane we can now explore the dynamics of the electrical activity. Up to this point we have only stated that the membrane is selectively permeable. That selection is facilitated by pores, or ion channels, embedded within the membrane of the cell. These channels, that are often specific

to species or valence, allow ions to conduct through them. However, that conduction is modulated by a number of different factors. To develop the models of excitable cells for this work we focus a subclass of ion channels whose conductance is dependent on the membrane voltage but first general models of ion channels are developed.

## 2.3 Ion Channels

Ion channels are constructed of proteins that span the phospholipid bilayer of a cell. There are many different families of ion channels, and cells contain a lager number of channels from a particular family that are spread throughout the membrane. Individually ion channels behave stochastically, changing their conductance randomly through opening and closing. These random fluctuation are often dependent on an external phenomena. So called ligand gated channels will modify their conductance based on concentrations of particular molecules and voltage-gated channels are controlled by the membrane potential.

The activity of ion channels at the single and whole-cell levels can accurately be approximated using Markov chain models. A Markov process is memoryless, in that the transition from one state to another is not dependent on what the previous states were; only the current state is important. To illustrate this, consider the two-state ion channel in Figure 2.4.

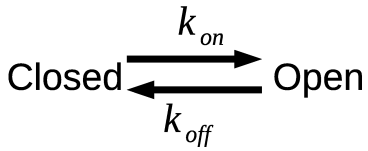



**Figure 2.4:** Two state markov channel model

If we consider the chance that the channel is in the closed state at a particular time $t$, then the rate $k_{on}$ is related to the probability that in the time interval $(\Delta t)$ the channel will

open (Fall et al., 2002). This relationship can be defined by

$$k_{on}\Delta t = Prob\{s = O, t + \Delta t | s = C, t\}.$$

Where $k_{on}\Delta t$ is dimensionless and $Prob\{s = O, t+\Delta t|s = C, t\}$ indicates the probability, given that the channel is closed at time $t$, of a closed to open transition occurring in the interval $[t, t + \Delta t]$ (Fall et al., 2002). Similar analysis can be completed for the transition from open to closed as well as the probability that the channel remains in its current state. The chance that a channel moves from one state to another can be summarized by the *transition probability matrix*

$$\begin{aligned} Q &= \begin{bmatrix} Prob\{C, t + \Delta t|C, t\} & Prob\{C, t + \Delta t|O, t\} \\ Prob\{O, t + \Delta t|C, t\} & Prob\{O, t + \Delta t|O, t\} \end{bmatrix} \\ &= \begin{bmatrix} 1 - k_{on}\Delta t & k_{off}\Delta t \\ k_{on}\Delta t & 1 - k_{off}\Delta t \end{bmatrix}. \end{aligned} \tag{2.9}$$

The elements of the *transition probability matrix* ($Q_{ij}$) represent the probability of a transition from state $j$ to state $i$. From the conservation of probability the columns of $Q$ must sum to $1$.

One way to simulate the stochastic activity of a single ion channel is to use Gillespie's Method. This utilizes the fact that the time to the next transition is an exponentially distributed random variable with a mean equal to the reciprocal of the rate coefficient (Fall et al., 2002). We can therefore simulate a two state channel by selecting open and closed dwell times consistent with that probability distribution.

The closed dwell time, $\tau_C$, of the two state channel has the probability distribution

$$Prob\{\tau < \tau_C \leq \tau + d\tau\} = k_{on} e^{-k_{on}\tau} d\tau.$$

Similarly, the open dwell time $(\tau_O)$ of the channel is an exponentially distributed random variable with the probability distribution function

$$Prob\{\tau < \tau_O \leq \tau + d\tau\} = k_{off} e^{-k_{off}\tau} d\tau.$$

With these we can simulate the channel by alternately choosing open and closed dwell times consistent with these distributions. A simple way to do this is to choose a uniformly distributed random variable $U$ on the interval $[0, 1]$ and use

$$\tau_C = -\frac{1}{k_{on}} lnU$$

and

$$\tau_O = -\frac{1}{k_{off}} lnU.$$

The results of this type of simulations are illustrated in Figures 2.5, 2.6 and 2.7. Notice as the rate constants are varied the open state probability, $P_O$, will change. This was calculated after $100$ simulated transitions.

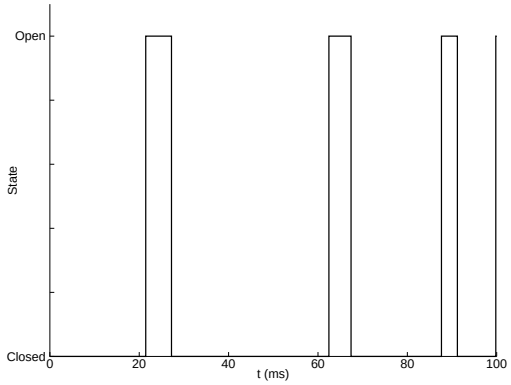



**Figure 2.5:** Single channel model using Gillespie's method. $k_{on} = 0.1ms^{-1}$, $k_{off} = 0.1ms^{-1}$, $P_O \approx 0.5$

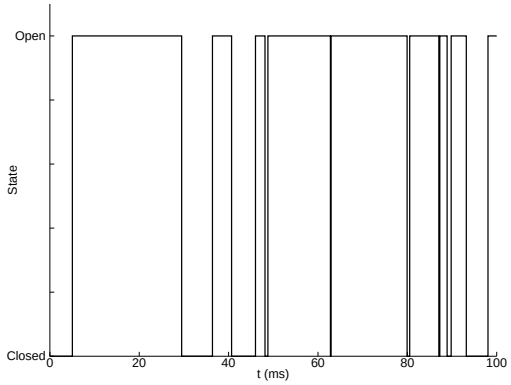



**Figure 2.6:** Single channel model using Gillespie's method. $k_{on} = 0.3ms^{-1}$, $k_{off} = 0.1ms^{-1}$, $P_O \approx 0.75$

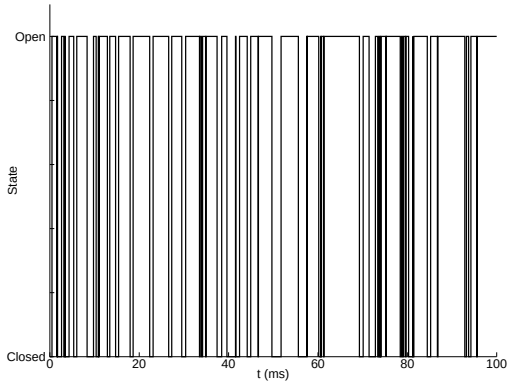



**Figure 2.7:** Single channel model using Gillespie's method. $k_{on} = 1.5ms^{-1}$, $k_{off} = 0.5ms^{-1}$, $P_O \approx 0.75$

When the stochastic gating of a number of single cells is averaged together, the whole cell activity emerges. There are several ways we can model this, including simulating a large number of stochastic channels. The most straightforward is to directly model the ordinary differential equations that result from taking the limit as $\Delta t \to 0$ of the probability relationship defined above (Fall et al., 2002). For the two state channel this would be

$$\frac{dP_C}{dt} = -k_{on}P_C + k_{off}P_O,$$

and

$$\frac{dP_O}{dt} = k_{on}P_C - k_{off}P_O.$$

From conservation of probability we can eliminate $P_C(t)$ using the relation

$$P_C(t) = 1 - P_O(t).$$

The governing differential equation is then

$$\frac{dP_O}{dt} = k_{on}(1 - P_O) - k_{off}P_O. \tag{2.10}$$

This two-state model is simple enough that a solution to the resulting equations can be found analytically. At steady-state $\frac{dP_O}{dt} = 0$ and the steady-state open probability is then

$$P_{Oss} = \frac{k_{on}}{k_{on} + k_{off}}.$$

Solving the simple differential equation results in the equation of open probability over

time

$$P_O(t) = P_{Oss} - (P_{Oss} - P_{OInitial}) * e^{-\frac{t}{\tau}},$$

where $\tau = \frac{1}{k_{on}+k_{off}}$. Basically, when the rate constants are stepped to different values the open probability will go to a new steady-state value with the time constant $\tau$. Figure 2.8 illustrates the settling of the model using the analytical solution.

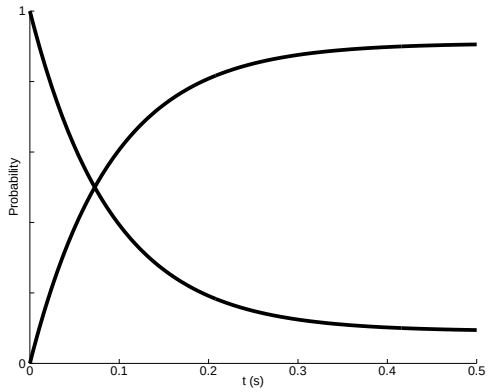



**Figure 2.8:** Analytical solution for a two-state ion channel model.

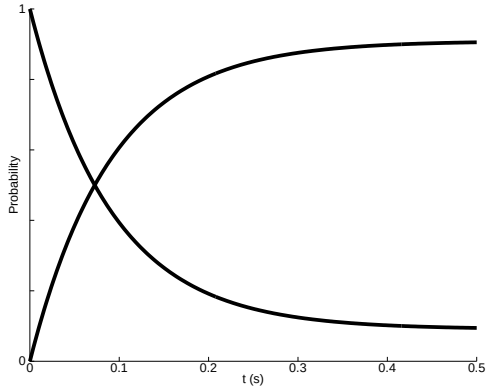



**Figure 2.9:** Numerical solution for a two-state ion channel model.

Although it is possible to find an analytical solution to the two state channel presented here, this is not often the case. As the complexity of the channels increase so does the

number of states and the interactions of the rate variable with other phenomena (i.e. ligand concentrations of voltages), making an analytical solution difficult or impossible to find. In these cases numerical methods must be employed. Figure 2.9 illustrates the solution to the same two-state channel model using a Runge-Kutta method.

## 2.4 Active electrical transmission

Many of the concepts presented above were actually established after the work of Alan Hodgkin and Andrew Huxley on the electrophysiology of the giant squid axon (Hille, 2001). Their work, empowered by the newly discovered voltage clamp techniques, developed the first kinetic theories of membrane permeability. These mathematical descriptions, accomplished without any knowledge of ion channels, are part of the foundation of neuroscience.

The first key insight provided by this work was that the membrane currents observed in the giant squid axon could be separated into different ionic species. This was important in determining the mechanisms involved in the changing membrane permeability. Two contributing species were identified, $K^+$ and $Na^+$, as well as a small background leak current. Through a series of ingenious experiments, Hodgkin and Huxley were able to demonstrate that not only do these currents mostly follow Ohm's law, but their permeability can be described by combinations of two-state kinetic processes. Here we focus on the mathematical model (HH model) that was developed based on these experiments.

The HH model equations describe the electrical activity of the giant squid axon in terms of ionic and capacitive currents (2.11).

$$I_M = I_C + I_i \tag{2.11}$$

Returning back to Figure 2.3, the ionic currents can be separated into

$$I_M = C_M \frac{dV}{dt} + \bar{g}_K n^4 (V - V_K) + \bar{g}_{Na} m^3 h (V - V_{Na}) + \bar{g}_{leak} (V - V_{leak}). \quad (2.12)$$

Where $I_M$ is the total membrane current (including externally applied currents). Notice that the conductances are now defined by the product of $\bar{g}_i$, the maximal conductance for ionic species $i$, and one or more gating variables. Starting with the non-inactivating slow $K^+$ current, $I_K$, we can see that the gating is dependent on the variable $n$. This is called the activation variable and essentially defines the percentage of channels that are open and contributing to the conductance of $K^+$. This probability can be defined by the two-state kinetic process,

$$\frac{dn}{dt} = \alpha_n(V)(1 - n) - \beta_n(V)n. \quad (2.13)$$

Recall that this is equation 2.10 developed for the two-state whole cell ion channel model. The transition from the closed non-conducting state is driven by $\alpha_n(V)$ and the transition back is now dependent on $\beta_n(V)$, $k_{on}$ and $k_{off}$ respectively. The key difference is the voltage dependence on these variables. Hodgkin and Huxley were able to fit that dependence to their experimental results using the equations

$$\alpha_n = \frac{0.01\,(V + 10)}{e^{\left(\frac{V-10}{10}\right)} - 1}$$

and

$$\beta_n = 0.125 e^{(V/80)}.$$

This combination of Ohmic devices and two-state kinetic process describes the contribution of $K^+$ to the membrane current at different voltages. Returning back to the $I_K$ term of Equation 2.12, the exponent attached to $n$ is still left unexplained. The currents observed by Hodgkin and Huxley had a time course that led them to predict four different independent particles that contribute to the activation of a channel. Given this independence, the probability that all of these channels was in the open state is $n^4$ (Hille, 2001).

The $K^+$ channel described by the HH model has only an activation gating whose dynamics determine the channels conductance. However, it was discovered that the $Na^+$ channels are dependent on two independent, but competing gates; an activation gate similar to $K^+$ and an inactivation gate. Inactivation in $Na^+$ channels is the process of closing the channel after it has been depolarized (driven to a more positive potential). This is independent of the channel's activation processes and conductance through the channel is only returned after a prolonged polarization (membrane potential less than or equal to the resting potential). In the HH model these two processes are again represented by two-state kinetic models defined by the equations

$$\frac{dm}{dt} = \alpha_m(V)(1-m) - \beta_m(V)m \tag{2.14}$$

and

$$\frac{dh}{dt} = \alpha_h(V)(1-h) - \beta_h(V)h. \tag{2.15}$$

The rate constants for these variables are defined by

$$\alpha_m = \frac{0.1\,(V+25)}{e^{\left(\frac{V+25}{10}\right)} - 1},$$

$$\beta_m = 4e^{(V/18)},$$

$$\alpha_h = 0.007e^{(V/20)},$$

and

$$\beta_h = \frac{1}{e^{\left(\frac{V+30}{10}\right)} + 1}.$$

It is important to point out that the above equations were developed using sign conventions and voltage offsets that are different from the conventions used today. For convenience the following table summarizes these differences:

| | **H-H** | **Present** |
|---|---|---|
| Membrane Potential | $V$ | $E$ |
| Resting Potential | $V = 0mV$ | $E = E_{r.p.} \cong -60mV$ |
| $Na^+$ Equilibrium Potential | $V_{Na} = -115mV$ | $E_{Na} = 55mV$ |
| $K^+$ Equilibrium Potential | $V_K = 12mV$ | $E_K = -72mV$ |
| Current | $I$ | $I$ |
| Inward Current | $> 0$ | $< 0$ |
| Clamp Currents | Inward Current $\uparrow$ | Inward Current $\downarrow$ |

In addition, the more conventional form of the (in)activation equations is

$$\frac{dn}{dt} = \frac{(n_\infty - n)}{\tau_n(V)},$$

$$\frac{dn}{dt} = \frac{(m_\infty - m)}{\tau_m(V)},$$

and

$$\frac{dh}{dt} = \frac{(h_\infty - h)}{\tau_h(V)}.$$

Where the steady-state gating values are defined by

$$n_\infty = \frac{\alpha_n}{\alpha_n + \beta_n},$$

$$m_\infty = \frac{\alpha_m}{\alpha_m + \beta_m},$$

and

$$h_\infty = \frac{\alpha_h}{\alpha_h + \beta_h},$$

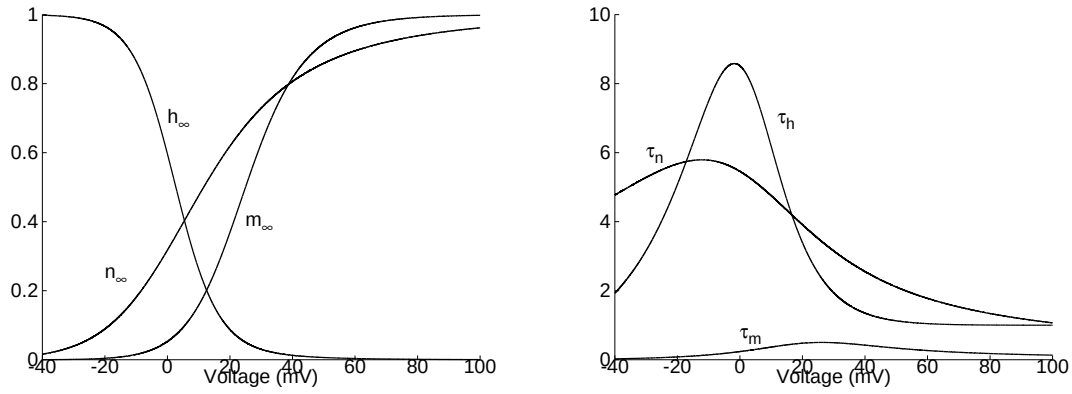
The voltage dependent time constants are defined by

$$\tau_n = \frac{1}{\alpha_n + \beta_n},$$

$$\tau_m = \frac{1}{\alpha_m + \beta_m},$$

and

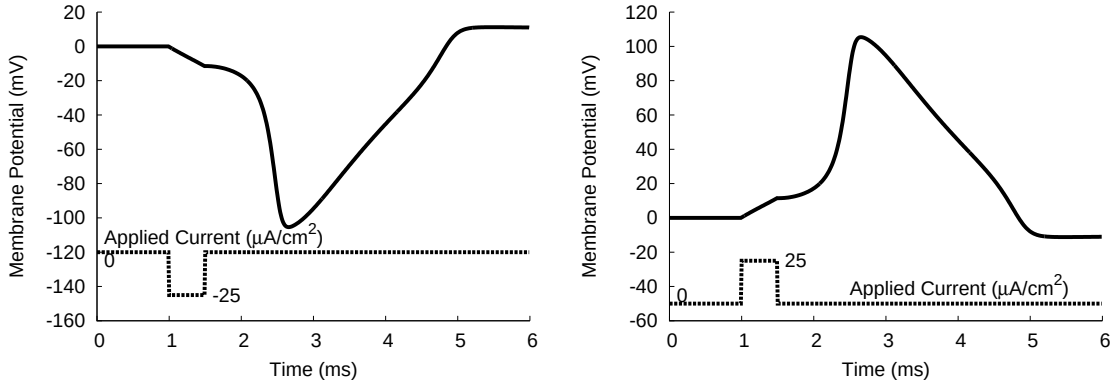
$$\tau_h = \frac{1}{\alpha_h + \beta_h}.$$




(a) (b)

**Figure 2.10:** Hodgkin-Huxley Variables. (a) (in)activation variables., (b) Time constants.

Figure 2.10 plots these at different voltage potentials. It is important to point out that these equations are simple curve fits to the data observed by Hodgkin and Huxley. They make predictions about the kinetics involved in the gating of these channels but do not provide a mechanistic description of how that gating is accomplished. There has been a tremendous amount of subsequent research that has demonstrated that these interactions are more complicated than communicated here. However, the HH equations are the foundation for the computational models developed in this dissertation.




(a) (b)

**Figure 2.11:** Hodgkin-Huxley action potentials. (a) Original Hodgkin-Huxley Action Potential, (b) Inverted Action Potential.

The model developed by Hodgkin and Huxley accurately describes the dynamic change

in membrane voltage referred to as the action potential (AP). This phenomena, that is unique to excitable cells, is essential for sensing, processing and communication in the nervous systems as well as glandular and muscle function. An AP is generated when the membrane voltage of a cell is driven past a threshold. When that threshold is crossed an avalanche of current drives the membrane voltage to spike and then repolarize. Figure 2.11 illustrates the AP of a point along the giant squid axon described by the HH Model.

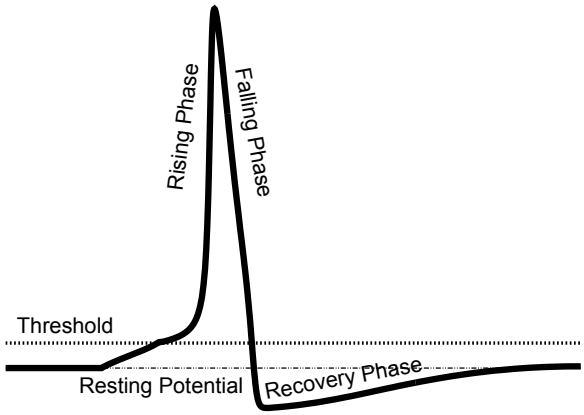



**Figure 2.12:** Stereotyped action potential.

The AP begins with the activation of $Na^+$ channels, resulting in an inrush of $Na^+$ ions and the rising phase of the membrane potential labeled in Figure 2.12. The increased membrane potential enables the inactivation gate, blocking the conductance of $Na^+$. Simultaneously, the $K^+$ channels become more activated as the membrane voltage increases. The net efflux of $K^+$ ions drives the membrane voltage back down towards its resting potential. These are illustrated by the falling phase on Figure 2.12. The $Na^+$ channels are then deinactivated (the process of unblocking by the inactivation gate) and they become available to contribute to further APs. In addition, the $K^+$ channels slowly deactivate, relaxing their contribution to driving the membrane voltage negative. This is illustrated in slow rise in the membrane voltage in the recovery phase of Figure 2.12.

The HH model describes an AP at a specific point along the giant squid axon. However, once an AP is generated it propagates away from that point throughout the cell. The propagation of an AP is facilitated by both the active and passive transmission of the electrical signal. Although it is not described here, the passive transmission of electricity in biological cells is defined by the cable equations (Koch and Segev, 1998; Bower and Beeman, 1998; Hille, 2001). These describe how the changes in membrane voltage at a location travel a relatively small distance away from the source. This is due to the charge traveling perpendicular to the membrane, into the cytosol and extracellular fluids, as well as parallel to it. Cells rely on the proximity of neighboring channels being small enough to sense APs or external insulation (a myelin sheath) to reduce the loss of charge away from the membrane.

## 2.5 Synaptic transmission

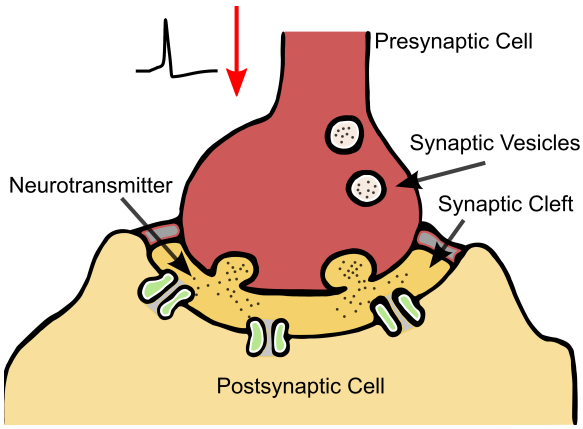



**Figure 2.13:** Neurotransmitter release at the synapse.

When an AP is fired, it travels down an axon and terminates at the synaptic cleft. This is where the innervating neuron and its target neuron are connected through a fluid interface.

At the majority of neural synapses there is no physical connection capable of transmitting an AP between the cells (this is not the case with connexons and gap junctions which are not covered here). Instead, a chemical signal facilitated by neurotransmitters is the mechanism of communication. The arrival of the AP, and the resulting voltage change, open $Ca^{2+}$ channels at the presynaptic terminal. The inrush of $Ca^{2+}$ ions initiates a communication cascade that culminates in the exocytosis of neurotransmitter molecules. The neurotransmitters travel across the fluid interface of the synaptic cleft down their concentration gradient. At the post-synaptic neuron they then signal ion channels, resulting in a dynamic change in membrane potential at the post-synaptic neuron. This process is illustrated in Figure 2.13. It is important to note that this change can be excitatory (positive voltage difference) or inhibitory (negative voltage difference).

### 2.5.1 Neurotransmitter Diffusion

The release and diffusion of neurotransmitter happens relatively fast; with the initial change occurring in as little as $0.3$ ms and peaking in $0.5$ to $1.0$ ms (Scott, 2002). Given the huge number of synapses in large-scale models, as well as the large integration time-steps of $0.5$ to $1.0$ ms, the onset of neurotransmitter release is generally approximated as instantaneous.

There is an immediate buffering of the neurotransmitter that occurs after release. In addition, re-uptake mechanisms on the presynaptic cell recycle it. In large-scale modeling this is either ignored and the neurotransmitter release is treated as a step change in post-synaptic membrane current, or the buffering and re-uptake is modeled by a decay function.

Rather than modeling the concentration of the neurotransmitter species, its influence on the conductance of the ion channels in the post-synaptic cell can be simulated. This takes

the form

$$\tau \frac{dg}{dt} = -g + \sum W_j \delta \left( t - t_j \right). \tag{2.16}$$

Where $g$ is the conductance, $\tau$ is the time constant that describes the time course of the decay in synaptic conductance, $W_j$ is the initial change in conductance when a spike occurs, and the delta function, $\delta \left( t - t_j \right)$, imparts that influence for a spike at time $t_j$.

This is generally a good approximation to the release and buffering of neurotransmitter. An important aspect that this equation does not address is the concept of a ceiling in conductance. There is no upper-bound in the conductance at the synapse which is not physiologically realistic. In most models however, this is not an issue as the activity of the synapse is not high enough to create an unbalance.

More detailed modeling efforts attempt a closer match to the empirically measured release curves. However, the mathematical benefit of that in large-scale models is still unknown and the computational burden it presents has been prohibitive. As the detail of these models increases those additions may prove important.

### 2.5.2 Synaptic plasticity

The ability for synapses to change their post-synaptic influence in both short and long time frames is essential for learning. Activity dependent changes, both positive and negative, are observed in most cortical connections of the brain and many theories of learning are built on these changes.

There are two ways that plasticity can modulate the effect of an AP, either the presynaptic release is modified or the sensitivity on the postsynaptic side is. Generally the release of neurotransmitter is discrete, with each vesicle containing a similar quantity. Therefore

fine changes in the concentration released are difficult to control. There are however, many instances where the plasticity of the synapse is dependent on the presynaptic neuron, but, these are exclusively a short-term phenomena. Plasticity is modulated more prominently at the postsynaptic site, where changes are dependent on activity in both a frequency and timing dependent manner. We discuss a few relevant examples of this below.

**Short-term plasticity**

Changes in synaptic efficacy on short time scales, milliseconds to seconds, is usually activity dependent. These mechanisms are affected by frequency changes in the presynaptic cell and generally related to vesicle availability (Purves et al., 2007). Augmentation is an increase in synaptic efficacy and is related to a greater vesicle release response to APs. Depression is a decrease in synaptic efficacy that is related to the reduced vesicle availability.

One way the short-term availability of a synapse can be modeled is in a frequency-dependent manner. The mathematical model of Markram et al. (1998) captures the phenomena of both frequency dependent potentiation and depression in a single synaptic model. Using a mix of variables from Markram et al. (1998) and Sussillo et al. (2007) the model can be described in terms of the fraction of total synaptic efficacy and the loss of availability to an AP. The conductance of the synapse can be defined by

$$g_n = A \cdot R \cdot u. \tag{2.17}$$

Where $A$ is the maximum synaptic efficacy, $R$ represents the fraction of synaptic availability, $R \in [0, 1]$, and $u$ is the utilization of synaptic efficacy. Facilitation in the model is included as an increase in $u$ in response to an AP. This then decays with time-constant $F$

to a resting value $U$. This can be described by

$$u_{k+1} = u_k exp\left(\frac{-\Delta t}{F}\right) + U\left(1 - u_k exp\left(\frac{-\Delta t}{F}\right)\right). \tag{2.18}$$

Where $\Delta t$ is the time difference between spike $n$ and spike $n+1$. Depression of the synapse can be modeled in a similar way, except now the available synaptic efficacy, $R$, is reduced as an AP arrives and is recovered with a time constant $D$. This is included in the expression for the available synaptic efficacy

$$R_{k+1} = R_n\left(1 - u_{k+1}\right) exp\left(\frac{-\Delta t}{D}\right) + 1 - exp\left(\frac{-\Delta t}{D}\right). \tag{2.19}$$

**Long-term plasticity**

Long-term changes in synapses are measured in minutes or hours. These happen quickly and persist through biophysical and genetic mechanisms. Descriptions of long-term plasticity are generally grounded in studies of excitatory synapses containing NMDA glutamate receptors. At rest NMDA receptors are clogged by a $Mg^{2+}$ ion, therefore they do not contribute to the membrane potential at the synapse. The $Mg^{2+}$ ion can be purged through prolonged or repeated depolarization of the synapse. When this happens the channel can then positively contribute to the membrane voltage through the conduction of $Ca^{2+}$ ions. This creates an immediate increase in excitability of the synapse. In addition, the influx of $Ca^{2+}$ begins a CaMKII and PKA initiated signaling cascade that leads to increased up-regulation of AMPA glutamate receptors(Purves et al., 2007). Long term depression of synapses is also similarly affected by $Ca^{2+}$, with high concentrations and low synaptic activity causing a down-regulation of AMPA channels (Purves et al., 2007).

One of the most popular long-term plasticity rules was discovered by Markram et al.

(1997) and is referred to as spike timing dependent plasticity (STDP). This a Hebbian learning rule where the arrival of presynaptic APs is compared to the activity of the postsynaptic neuron. If the presynaptic spike arrives before the postsynaptic neuron fires, the connection is potentiated; the presynaptic neuron is contributing positively to the postsynaptic neuron's firing. If the presynaptic spike arrives after the postsynaptic neuron fires the synapse is depressed; the presynaptic neuron is not contributing.

A version of this rule is for excitatory synapses observed in the cortex (inhibitory synapses have been shown to follow a different rule) and is presented here in the form used by Song et al. (2001). The conductance of a channel is updated using the expression

$$g \rightarrow g + g_{max} F(\Delta t). \tag{2.20}$$

Where $\Delta t = t_{pre} - t_{post}$, is the timing difference between the presynaptic and postsynaptic APs. The fractional update is

$$F(\Delta t) = \begin{cases} A_+ e^{\left(\frac{\Delta t}{\tau_+}\right)} & \Delta t > 0 \\ A_- e^{\left(\frac{\Delta t}{\tau_-}\right)} & \Delta t < 0 \end{cases} \tag{2.21}$$

if $(g < 0)$ then $g \rightarrow 0$

if $(g > g_{max})$ then $g \rightarrow g_{max}$

This is a reduced learning rule and only captures some of the phenomena observed experimentally. However, it is one of the most popular learning rules in large-scale modeling. As mentioned above, inhibitory synapses follow a completely different STDP rule; as do some subcortical structures. Exploration of these is left to the reader.

## 2.6 Simple point neuron models

The HH model presented in Section 2.4 is useful for describing the electrical activity of neurons but is computationally expensive to simulate. In addition, there is much more to the dynamics of neurons than just the generation of action potentials.

### 2.6.1 Reduced Hodgkin-Huxley neuron

We begin by showing how the dynamics of the HH model can be simplified through assumptions based on the physiology as well as the dynamics we have talked about previously. Although exploring the HH equations using dynamical system theory is quite intractable due to the four-dimensional nature of the system, we can characterize them by their fast and slow dynamics.

The first simplification is based on the assumption that $Na^+$ activation occurs much faster than its inactivation as well as the $k^+$ activation. Based on this it can be assumed that the $Na^+$ activation occurs instantaneously, or that $m = m_\infty(V)$.

The next simplification is based on the dependence between $h$ and $n$. Assuming the simple linear relationship

$$h(t) = 1 - n(t).$$

The HH equations can then be reduced to two coupled differential equations

$$I_M = C_M \frac{dV}{dt} + \bar{g}_K n^4 (V - E_K) + \bar{g}_{Na} m_\infty^3 (1 - n)(V - E_{Na}) + \bar{g}_l (V - E_l) \quad (2.22)$$

and

$$\frac{dn}{dt} = \alpha_n(V)(1-n) - \beta_n(V)n. \tag{2.23}$$

When compared to the full model, Figure 2.14, it can be observed that it is not a great approximation but for some applications it may be acceptable.

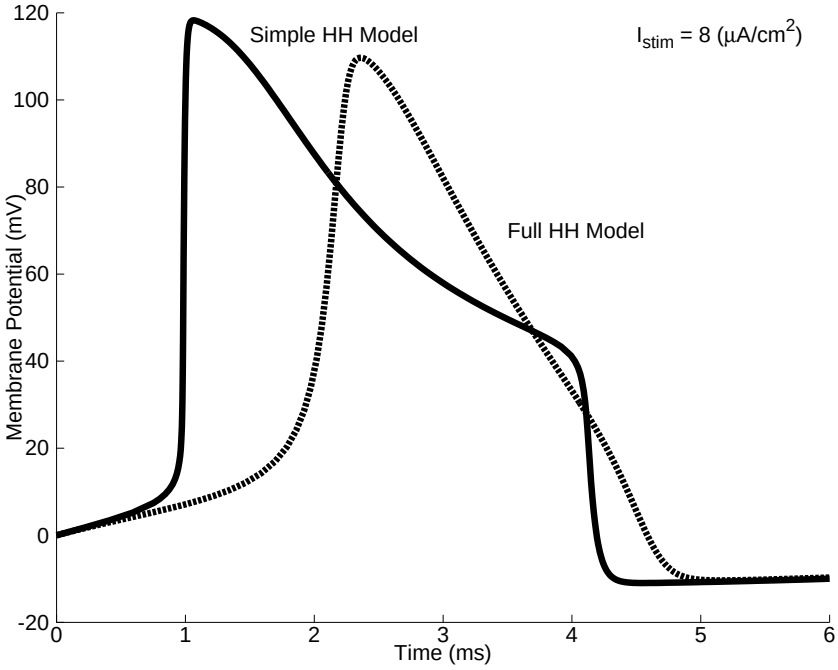



**Figure 2.14:** Comparison of simple model with the full model.

Returning to the full Hodgkin-Huxley model and plotting the gating variables as action potentials are triggered, the relationship between $n$ and $h$ can be observed more closely Figure 2.15. This is exactly what was done in some of the early computer simulations (Izhikevich, 2007b). These demonstrated that

$$n + h \approx 0.84$$

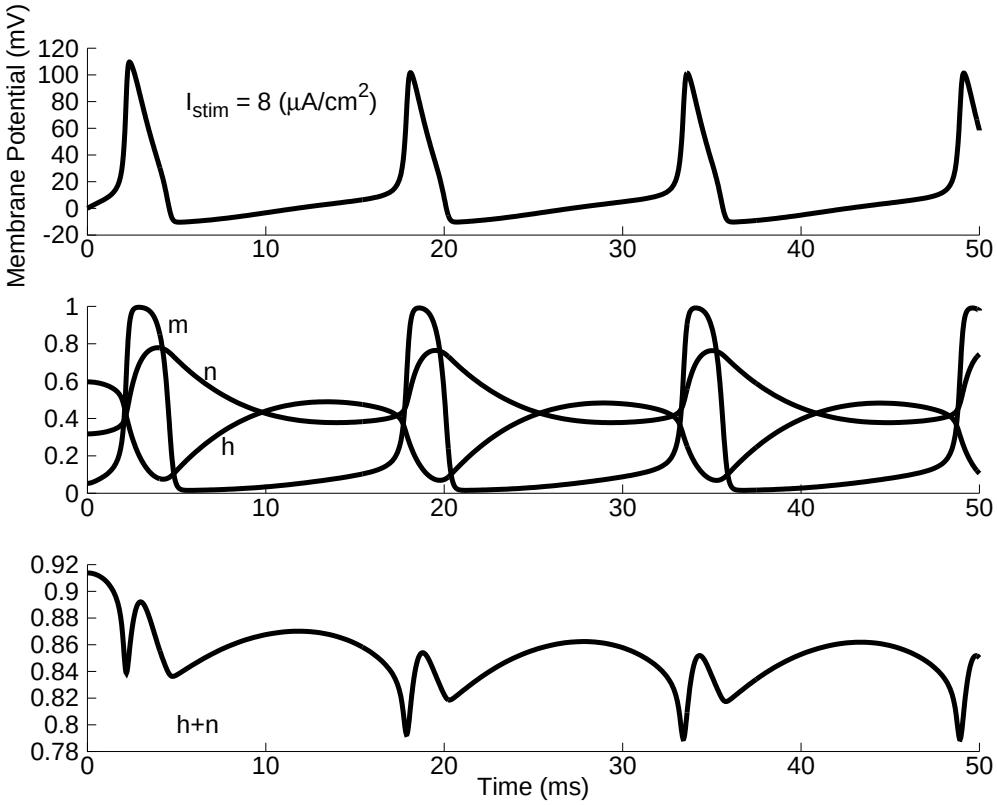



**Figure 2.15:** Full Hodgkin-Huxley Model. Notice how $n + h$ surrounds $0.84$

We can actually compare these directly as in Izhikevich (2007b), which reveals that

$$h = 0.89 - 1.1n.$$

Incorporating this relationship into the simplified model results in a better approximation to the full HH model, Figure 2.16. The model is still slightly shifted in time but comparing the aligned action potentials it can be seen how they compare, 2.17.

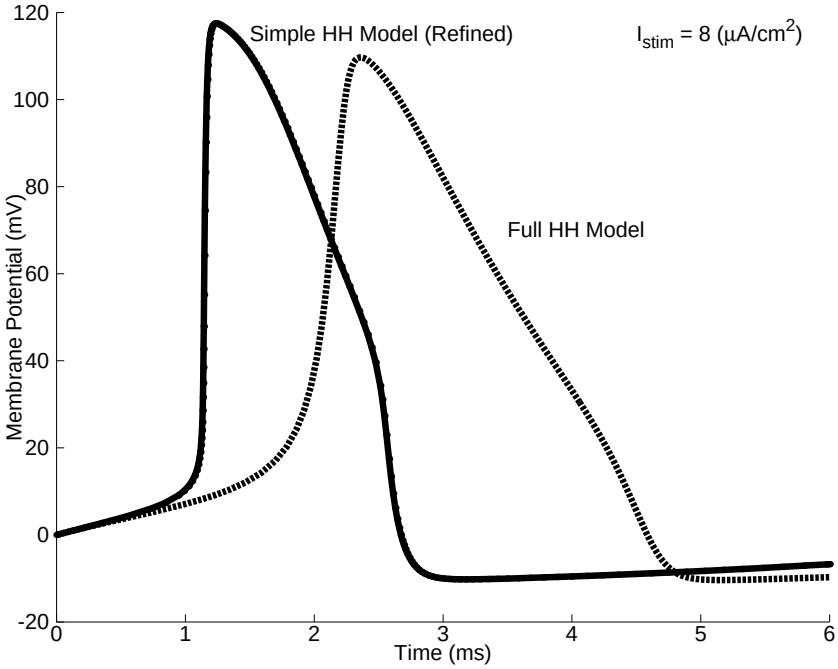



**Figure 2.16:** Comparison of the improved simple model with the full model.

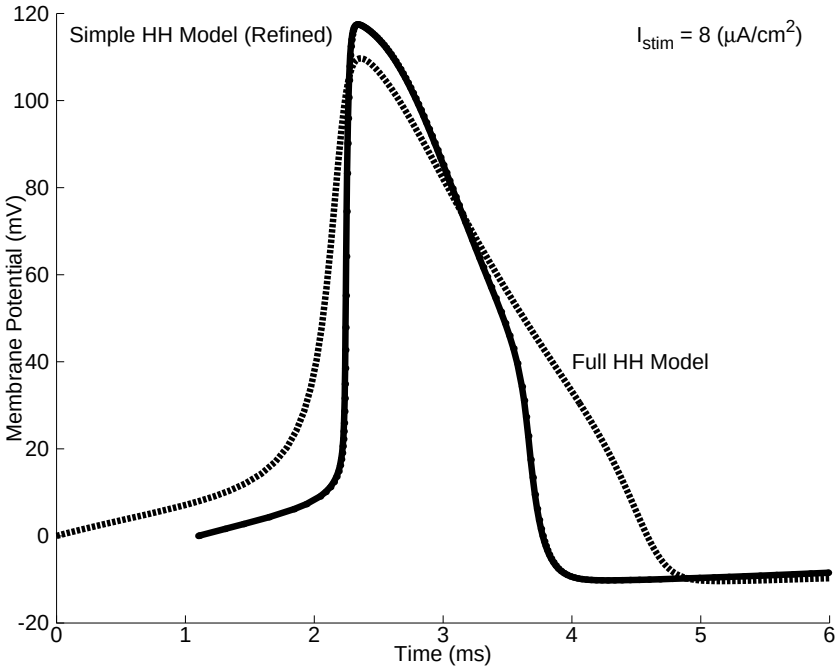



**Figure 2.17:** Comparison of improved simple model with the full model after shifting.

Although the method above improves the computational complexity of the HH equations, we can still take this further. The two channels of interest in the HH model, the fast sodium and slow rectifying $K^+$ channels, contribute mainly to the action potential generation. As described above, this is an all or nothing event that occurs once the threshold has been reached. Assuming that these channels have minimal contribution to the subthreshold dynamics, we can remove the channels and instead reset the membrane voltage once the threshold has been reached. The characteristic action potential can, and usual is, then added to the voltage potential using a spike template. This addition is completely arbitrary and really only useful for visualization.

### 2.6.2 Leaky Integrate-and-Fire Neuron

Probably the most basic spiking neuronal description is the integrate-and-fire model. With extensive research dating back as far as 1907, it is a simple model that requires limited computational resources (Koch and Segev, 1998). In the leaky integrate-and-fire model (LIF) the neuron sums the input currents to calculate the membrane potential. When the voltage potential reaches a set threshold an AP is fired. The model is created by removing the $Na^+$ and $K^+$ channels. The AP is simulated by reseting the membrane voltage and placing the neuron in a forced refractory period; where the membrane voltage is fixed for a set period of time. The equation takes the form

$$C_m \frac{dV}{dt} = I - g_{leak}(V - V_{leak}). \tag{2.24}$$

This is sometimes expressed as

$$C_m \frac{dV}{dt} = I - \frac{V_m}{R_m}. \tag{2.25}$$

Where

$$V_m(t) = IR_m \left(1 - e^{\left(\frac{-t}{\tau_m}\right)}\right),$$

and $\tau_m = R_m C_m$ is the time constant. This can also be simplified as

$$\dot{v} = b - v, \tag{2.26}$$

and if $v = 1$, then $v = 0$.

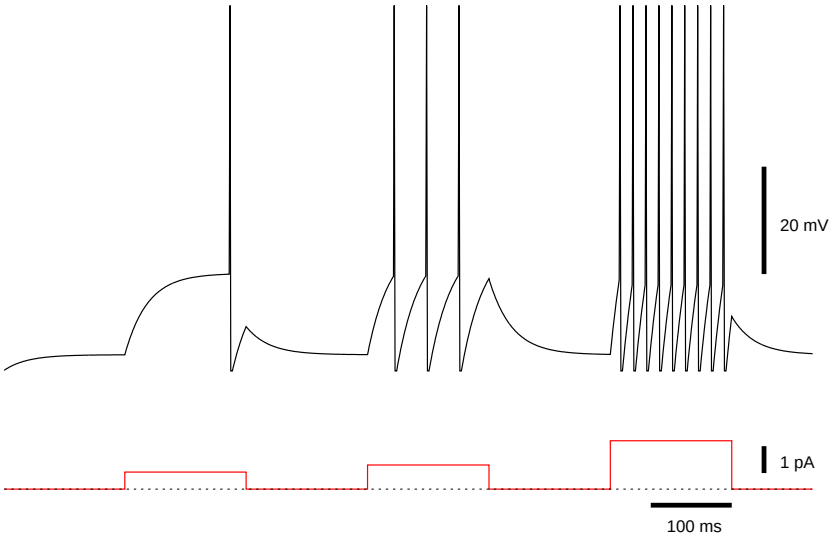



**Figure 2.18:** Leaky-Integrate-and-Fire neuron response to excitatory input.

Figure 2.18 demonstrates the response of a typical LIF neuron to increasing voltage stimulus. Note that the spike shapes are created by artificially setting the voltage to $30$ mV when the threshold is reached.

### 2.6.3 Subthreshold models

The LIF model is capable of capturing only a small subset of neuronal dynamics. It works well as an integrator but the more complex responses, characteristic to many areas of the nervous system, are lost with this model. One strategy for replicating those dynamics is to add additional currents representing ion channels that are involved in the subthreshold activity of the cells. Here we present three such ionic currents as an example of how the dynamics of the LIF model can be altered.

We begin with two currents that contribute to the membrane voltage by controlling spike-frequency adaptation. These are small ionic currents that have a long period of activity when the membrane voltage is between rest and threshold.

The noninactivating muscarinic $K^+$ current, $I_M$, is defined by

$$I_M = \bar{g}_M m^P (E_k - V). \tag{2.27}$$

Where $P$ is the power that the activation variable $m$ is raised to. This is essentially decreasing the slope of the activation variable. The change of that activation variable is defined as

$$\frac{dm}{dt} = \frac{m_\infty - m}{\tau_m}. \tag{2.28}$$

Where

$$\tau_m = \frac{\epsilon}{e^{\left(\frac{V - V_{1/2}}{\omega}\right)} + e^{-\left(\frac{V - V_{1/2}}{\eta}\right)}}$$

and

$$m_\infty = \frac{1}{1 + e^{-\left(\frac{V - V_{1/2}}{\xi}\right)}}.$$

Here, $\epsilon$ is a scale factor, $\omega$, $\eta$ and $\xi$ are slope factors affecting the rate of change of the activation variable m. $V_{1/2}$ satisfies the equation $m_\infty(V_{1/2}) = 0.5$.

Adding this current to the LIF model creates the classic non-accommodating gabaergic interneuron, Figure 2.19. These neurons fire at steady-state at the onset of a constant current injection.

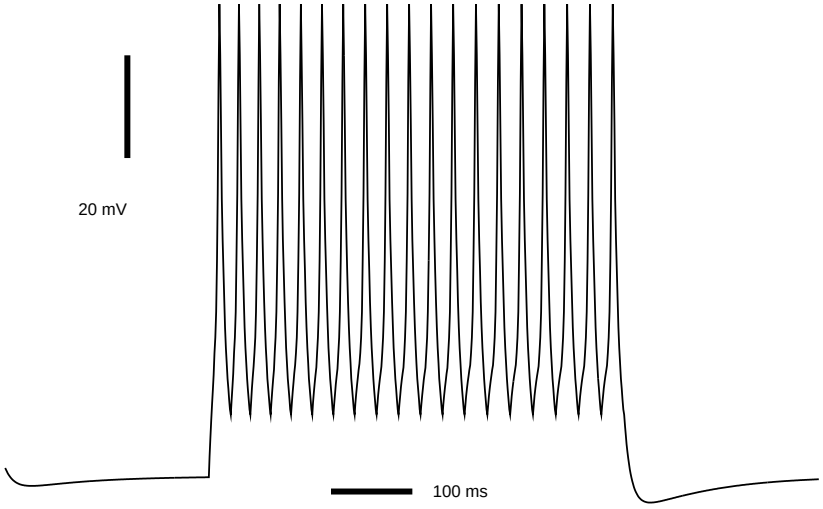



**Figure 2.19:** classic non-accommodating gabaergic interneuron modeled using the noninactivating muscarinic $K^+$ current.

The other small spike-adaptation contributing current is an afterhyperpolarization $K^+$ channel. These are voltage independent and regulated by internal calcium. The current is defined by

$$I_{AHP} = \bar{g}_{AHP} m^P (E_k - V). \tag{2.29}$$

Where $P$ is the power that the activation variable $m$ is raised to. The change of that activation variable is defined as

$$\frac{dm}{dt} = \frac{m_\infty - m}{\tau_m}. \tag{2.30}$$

Where

$$\tau_m = \frac{\epsilon}{f(Ca) + b},$$

$$m_\infty = \frac{f(Ca)}{f(Ca) + b},$$

$\epsilon$ is the scale factor, $b$ is the backwards rate constant and $f(Ca)$ is the forward rate constant defined by

$$f(Ca) = \kappa [Ca]_i^{\alpha}. \tag{2.31}$$

Physiologically the calcium concentration of a cell increases when an action potential fires. After the action potential has ended the internal concentration of calcium will diffuse throughout the cell where it is taken up by numerous physiological buffers. This diffusion/buffering phenomena can be modeled by a simple decay equation defined by

$$[Ca]_i(t+1) = [Ca]_i(t) \left(1 - \frac{dt}{\tau_{Ca}}\right). \tag{2.32}$$

Where $dt$ is the integration time step and $\tau_{Ca}$ is the time constant for the $Ca^{2+}$ decay.

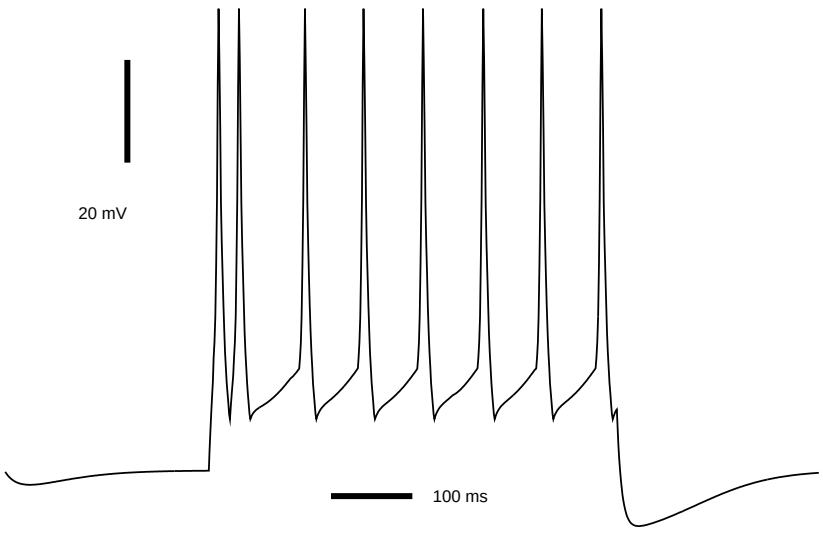



**Figure 2.20:** Bursting non-accommodating gabaergic interneuron response using small spike-adaptation contributing current is an afterhyperpolarization $K^+$ channel.

The addition of this channel simulates a bursting non-accommodating gabaergic interneuron, Figure 2.20. These are characterized by a delayed rise in membrane voltage after stimulus onset, followed by a period of steady-state firing.

The third channel type is the transient outward $K^+$ current or $K_a$. This channel requires hyperpolarization for its activation; meaning that the channel will open during inhibitory synaptic input. This is defined by

$$I_K = \bar{g}_M m^P h^C \left(E_k - V\right). \tag{2.33}$$

Where as before $P$ is the power that the activation variable $m$ is raised to and $C$ is the power that the inactivation variable $h$ is raised to. The change of activation and inactivation variables is defined by

$$\frac{dm}{dt} = \frac{m_\infty - m}{\tau_m} \tag{2.34}$$

and

$$\frac{dh}{dt} = \frac{h_\infty - m}{\tau_h}. \tag{2.35}$$

Here

$$m_\infty = \frac{1}{1 + e^{-\left(\frac{V - V_{1/2m}}{\xi}\right)}},$$

where $V_{1/2m}$ satisfies the equation $m_\infty(V_{1/2m}) = 0.5$ and $\xi$ is the slope factor affecting the

rate of change of the activation variable $m$, and

$$h_\infty = \frac{1}{1 + e^{-\left(\frac{V - V_{1/2h}}{\eta}\right)}},$$

where $V_{1/2h}$ satisfies the equation $h_\infty(V_{1/2h}) = 0.5$ and $\eta$ is slope factor affecting the rate of change of the inactivation variable $h$.

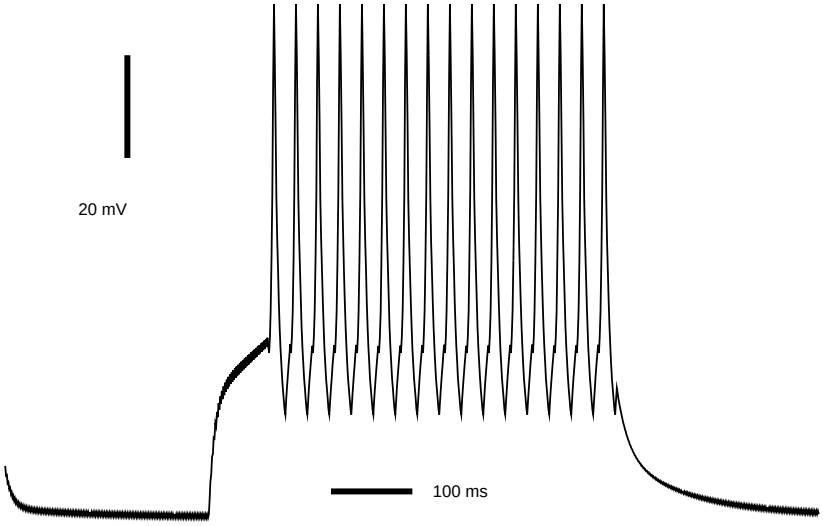



**Figure 2.21:** Delayed non-accommodating gabaergic interneuron modeled with a transient outward $K^+$ current.

Figure 2.21 is the response of a delayed non-accommodating gabaergic neuron that results from a transient outward $K^+$ current. The response is a delayed rise in membrane voltage after stimulus onset, followed by a period of constant firing.

As can be seen by the inclusion of these example currents, the LIF model can be employed for replicating more complex firing patterns. Combinations of subthreshold channels can be included to replicate almost any neuronal response.

An additional technique for extending the LIF neuron is to include an artificial current

source that replicates the desired current dynamics. This is used for the Integrate-and-fire-or-burst model that will be introduced in Chapter 9 as well as, among others, the resonate-and-fire neuron.

The resonate-and-fire model incorporates a function that replicates a low-threshold $K^+$ current. It is important to note that this term can also be any other current that is partially activated at rest. The current is defined by the variable $W$ and can be added to the LIF model by

$$C\frac{dV}{dt} = I - g_{leak}(V - E_{leak}) - W. \tag{2.36}$$

The dynamics of $W$ are defined by

$$\frac{dW}{dt} = \frac{V - V_{1/2}}{k - W}. \tag{2.37}$$

By changing the $V_{1/2}$ and the $k$ terms the resonate-and-fire model can simulated more complex excitability, damped oscillations and rebound spiking. Creating an artificial current term provides more flexibility to the model but this can be taken a step further by instead of directly incorporating the empirical ion channel models, the neuron is treated as a dynamical system. The goal is create a spike generating model that replicates the relevant dynamics but may lack biophysical meaning in its parameters.

### 2.6.4 Hybrid neuron models

Hybrid neuron models, as defined here, have a continuous spike-generation function and an after-spike reset (Izhikevich, 2010). Like the HH model, the spike-generation function does generate an AP, however it does not reset on its own. When the model voltage reaches

a spike-cutoff it is instead set to a reset value. The first example of this is the quadratic-integrate-and-fire neuron. This is created by replacing $-v$ in Equation 2.26 with $v^2$

$$\frac{dv}{dt} = b + v^2, \qquad \text{if } v = v_{peak}, \text{ then } v \leftarrow v_{reset}. \tag{2.38}$$

Izhikevich (2003) extended this by including a recovery variable $U$ which can put the model into different modes (i.e. resonant or amplifying). This can be defined by the system of equations

$$\frac{dV}{dt} = I + v^2 - u \qquad \text{if } v \geq 1, \text{ then} \tag{2.39}$$

$$\frac{dU}{dt} = a(bv - u) \qquad v \leftarrow c, u \leftarrow u + d. \tag{2.40}$$

Where $V$ is the membrane potential and $U$ is the recovery variable. The sign of $b$ determines if $U$ is an amplifying or a resonant variable. In addition, $a$ is the recovery time constant, $v_{peak}$ is the spike cutoff value and the reset voltage is defined by $c$. Finally, $d$ drives the after-spike behavior.

Equations 2.39 and 2.40 (5-parameter model) can replicate many different cortical cell types (Izhikevich, 2003) but for other cortical and subcortical neurons it is sometimes more powerful to represent the model as

$$C\frac{dV}{dt} = k(v - v_r)(v - v_t) - u + I; \qquad \text{if } v \geq 1, \text{ then} \tag{2.41}$$

and

$$\frac{dU}{dt} = a\{b(v - v_r) - u\} \qquad v \leftarrow c, u \leftarrow u + d \tag{2.42}$$

Where $C$ is the capacitance, $k$ is related the neuron's rheobase, $V_r$ is the resting membrane potential, and $v_t$ is the instantaneous threshold potential.

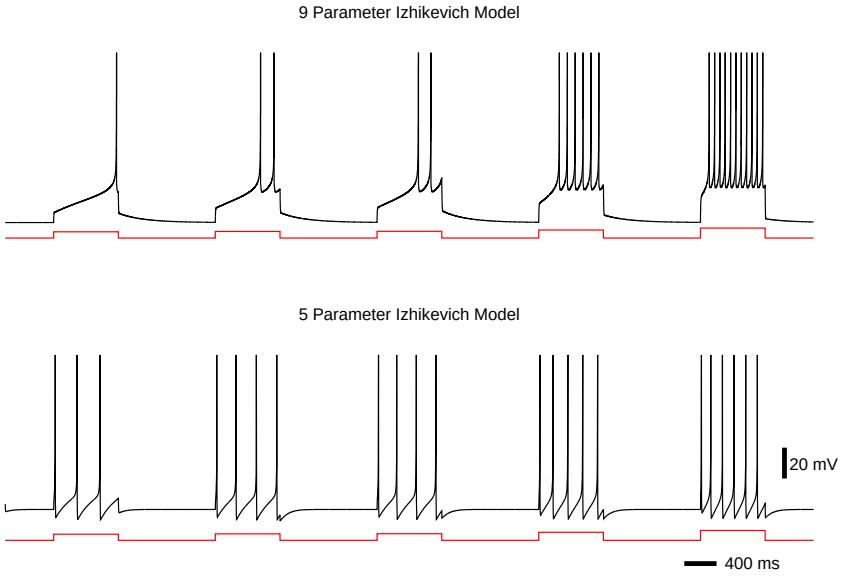



**Figure 2.22:** Izhikevich neuron. 5-parameter model (top). 9-parameter model (bottom).

It is slightly more expensive to simulate when expressing the model using Equations 2.41 and 2.42 (9-parameter model). However, it provides a level of control that the 5-parameter model does not. Figure 2.22 shows the simulation of a striatal neuron. The firing characteristics of striatal neurons demonstrate distinct up-down states. In the down state the neurons have significantly reduced excitability. If enough excitatory input is received to create an AP, the neuron remains in state of raised depolarization, the up state. In this increased state of excitability the neuron is more likely to respond to excitatory input with an AP. The 9-parameter model is capable of replicating those states, Figure 2.22 top. The 5-parameter model however, does not demonstrate this bistability.

## 2.7 Summary

These simple point neuron models represent a simplification of reality. However, even in this reduced form, the processing power cannot easily be described discretely. In some cases a mixed logic description fits; where the neurons process information in analog (membrane voltage) but communicate digitally (APs). Unlike a digital computer, these systems do not read in a set of instructions that describe how to handle the information flow. Instead, that processing is built into a constantly changing functional anatomy. Even this analogy falls short when exploring how that information is represented in a network.

There is currently no consensus on how the brain represents, processes and stores information. Research has presented examples of many different types of encoding schemes (Eliasmith and Anderson, 2003). For example, rate based representations assume the information is encoded in the rate of the neurons in a population. Or population based encoding; where the activity of an entire population contributes to encoding information. There is also spike-time encoding; where the time between successive spikes is used. All of these, and a few others, have been demonstrated empirically but there is no single one that can explain information flow in the brain. It is currently hypothesized that the brain incorporates all or a subset of these in processing and representing information (Eliasmith and Anderson, 2003). Describing this phenomena as a comparatively simple computer does not to do it justice.

# Part I

# Engineering Neural Systems

# Chapter 3

# Anatomy of a High-Performance Neural Simulator

There are many difficulties to overcome in the modeling of large-scale neural systems. These inherent difficulties can be further compounded by the need for high-performance and near real-time simulations. Although the simulation of spiking neural systems can be classified as embarrassingly parallel, the models generally do not scale linearly with the number of compute elements. This is due to the computational complexity of numerically integrating the governing equations, as well as efficiently communicating spiking information. This chapter presents some concepts used in designing modern neural simulators. Here the focus is on network layout, distribution and numerical integration. Chapter 4 expands on this by exploring spike exchange methods on general-purpose high-performance computing clusters.

## 3.1 Introduction

For many researchers the choice of neural simulation environment can be extremely difficult. There is a continuous trade-off between biological realism and computational complexity. If access to high-performance computing resources is unavailable, large-scale modeling may be out of reach. In addition, the time investment required for installing and learning a new simulator is a hindrance. With the relatively low-cost of GPGPU computing more researchers now have the ability to explore large-scale models. However, the difficulty in adopting a new simulation environment remains.

### 3.1.1 CPU based neural simulators

There are a number of general CPU based simulators that support large-scale neural models. NEURON, (Hines and Carnevale, 2007; Hines and Carnavale, 1997), and GENESIS, (Bower et al., 2002; Bower and Beeman, 1998), are two of the most popular. Both offer CPU versions for single and distributed computer environments. This list also includes NEST, (Diesmann and Gewaltig, 2002), NCS, (Wilson et al., 2001), and CSIM, (Brette et al., 2007). Below is a review of some of the more prominent CPU simulation environments.

NEURON was started by John W. Moore and Micheal Hines at Duke University (Hines and Carnevale, 2007; Hines and Carnavale, 1997). Originally, The NEURON simulator was used for modeling single neurons in high levels of detail. It has since been applied to the study of larger networks of neurons. It is most useful for simulations where the neurons of interest are spatially diverse, have complex membrane currents and channel dynamics, and both intracellular and extracellular ionic concentrations are important (Hines and Carnavale, 1997).

Different mechanisms for constructing cells is provided by the concept of sections. This provides a level of abstraction that not only removes the underlying differential equations from the user but also separates the cell physiology from the numerical solution. Several solution methods for the differential equations are also provided. The choice of algorithm is left to the user and depends on the level of accuracy needed and the overall complexity of the model. NEURON has had the benefit of years of development that has resulted in a robust tool set that has helped to attract a wide user-base. A parallel implementation is offered with the current version and can distribute the simulation by neuron or even by section of neuron.

The General Neural Simulation System (GENESIS) project began at California Institute of Technology by James M. Bower. It was originally utilized for the simulation of large neural networks (Bower et al., 2002). It has since been used in studies of varying levels of abstraction and elaboration. The GENESIS design employs an object oriented approach at the simulator level. Modules within the simulator are black-boxed; allowing connecting modules the luxury of only needing the associated interface. This provides modelers the ability to change and reuse discrete components of the simulator without having to change unassociated code. GENESIS also offers a parallel simulation environment allowing researchers to model over networked workstations, a parallel cluster or supercomputer (Brette et al., 2007; Bower et al., 2002; Bower and Beeman, 1998).

The NeoCortical Simulator (NCS) was developed at The University of Nevada, Reno by the Brain Computation Lab under the direction of Dr. Phillip Goodman. From its inception a heavy emphasis was placed on parallelization and performance. In addition, mechanisms for accessing spiking information and injecting stimulus was also extremely important. Despite the focus on performance, NCS provides a number of important biological models. For a review of what NCS refer to Wilson et al. (2001) and to see how NCS compares to

other neural simulators seeBrette et al. (2007).

NEural Simulation Technology (NEST) (Diesmann and Gewaltig, 2002) is the result of the NEST initiative, a collaborative project intended to help extend neural simulator development. NEST is intended for simulations of large neural networks consisting of point neurons and neurons with minimal compartments. It is employed in studies interested in the dynamics of neural structures Brette et al. (2007); Diesmann et al. (1999). Parallelism is achieved by a combination of multithreading and message passing.

Circuit SIMulator (CSIM) (Brette et al., 2007) is a package for modeling networks of point neurons. It was designed for studies at the network level, with the intention of revealing high level network dynamics that are unavailable at the single cell level. The software itself is a combination of a C++ solution engine and Matlab or Python interface. Currently the implementation includes models for linear leaky integrate-and-fire neurons, non-linear leaky integrate-and-fire neurons and compartmental based neurons.

XPPAUT (Brette et al., 2007) is a software package for solving differential, difference, delay, functional, and stochastic equations as well as boundary value problems. While it was developed as a general numerical tool, its ability to analyze a numerical system's dependence on specific parameters has made it extremely useful to neuroscientists.

SPLIT is an experimental package for modeling large-scale multicompartmental neural models. Rather than rely on a formal interpreted modeling language, SPLIT is a kernel that developers incorporate into custom C++ programs. It has support for parallel computations that are completely hidden from the user.

Large-scale Edge Node Simulator (LENS) (Peck et al., 2003) is a simulation environment developed by the Biometaphorical Computing Group at IBM T.J. Watson Research Center. LENS is considered a "problem solving environment" for large neural networks. It offers researchers unique levels of abstraction that range from compartments to global

brain processes.

Each of these provides a parallel CPU implementation that is well-suited for many distributed environments. However, they do not yet offer a GPU compatible version.

### 3.1.2 GPU based neural simulators

Given the ubiquity of graphical processing units (GPUs) in almost every computing device today, it is no surprise that they are being exploited for general purpose computing. GPUs have a large number of single-instruction multiple-data (SIMD) processors capable of efficiently processing huge amounts of data in parallel. In addition, the cost associated in creating clusters of GPU's is considerably lower than CPU based super-computers capable of similar performance (Fan et al., 2004).

In the computational neuroscience community there have been a large number of projects focused on single or dual-GPUs localized to a single compute node, however there are no GPU-cluster based neural simulation environments openly available at the time of writing.

The lack of GPU support in general neural simulation has resulted in a number of projects focused on creating general environments specific to GPU implementations. This began with Nageswaran et al. (2009) and their release of a simulator supporting Izhikevich neuron models, and a C++ user interface for creating networks. This work was recently updated by Richert et al. (2011), however, both target a single GPU.

Thibeault et al. (2011) presented a proof-of-concept simulator that targeted multiple GPU's within a single computer, this is described in Section 3.4. A method for distributing the simulations on multiple nodes was presented as well as a novel spike message passing scheme that represented the neuron states using individual bits. Izhikevich neurons were supported along with conductance based synapses and STDP based plasticity.

Similarly, a number of projects have resulted in model-specific GPU implementations.

Scorcioni (2010) presented a single GPU simulator capable of modeling 100,000 Izhikevich neurons with 100 randomly connected STDP synapses in real-time.

Tiesel and Maida (2009), created a single planar network of Integrate-and-fire neurons using the OpenGL graphics API. Along the same lines, the work of Han and Taha (2010), presented a two-layer input-output network specific to image recognition.

Igarashi et al. (2011) developed a heterogeneous model of action-selection in the basal ganglia. Two different neuron types are simulated in the model, Izhikevich neurons for the striatum and Leaky integrate-and-fire with Hodgkin-Huxley channels for the other areas. The simulation was executed on a single CPU-GPU combination in real-time.

Richmond et al. (2011) presented a model with 2 layers of integrate-and-fire neurons with recurrent connections. The resulting code demonstrated a speed-up as high as 42x over the comparable Python implementation. In this case the parallelism of the GPU was exploited for parametric optimization.

Fidjeland and Shanahan (2010) published results for a single GPU simulation similar to the work of Nageswaran et al. (2009). The system demonstrated higher throughput, defined as spike arrivals per second, but a lower number of total neurons for real-time simulation.

Yudanov et al. (2010) demonstrated a single GPU simulations of Izhikevich neurons integrated using an adaptive Parker-Sochacki method. Emphasis was placed on sub-millisecond event tracking and accuracy between CPU and GPU implementations. Speedup of 9x between a comparable CPU implementation was achieved.

Nere et al. (2011) created an extension to a learning model of the mammalian neocortex. The simulation abstracted the neural activity up to the level of neocortical minicolumns using a rate-based model. Synaptic plasticity is only applied to active columns and follows a Hebbian learning rule where the weight matrix between columns is increased if the input is active and decreased if it is not. Simulations were distributed between a single CPU and

either a single GPU card or dual GPU cards. The resulting implementation demonstrated a 60x speedup over the single-threaded implementation.

de Camargo et al. (2011) created a GPU simulation of multi-compartment conductance-based neurons. The test network was comprised of excitatory pyramidal cells and inhibitory cells. Each neuron contained two channel conductances modeled using Hodkgin-Huxley dynamics. Different number of connections, weights and neural activity were explored resulting in a speed up of 40x in some cases over the serial CPU implementation.

## 3.2 Generic simulator design

The basic steps in a neural simulation are network design, network construction, integration, spike exchange and finally reporting. To illustrate the storage containers and basic concept of spike exchange a simple neural simulator, Neurolite, is presented. Neurolite is a general neural simulator developed by the author to overcome some of the limitations present in NCS (i.e. limited channel implementations, complexity of network creation and difficulty accessing individual neurons). The simulator supports single compartment models that include only the active transmission of electrical signals. Currently model creation is only in C++.

## Data Structures

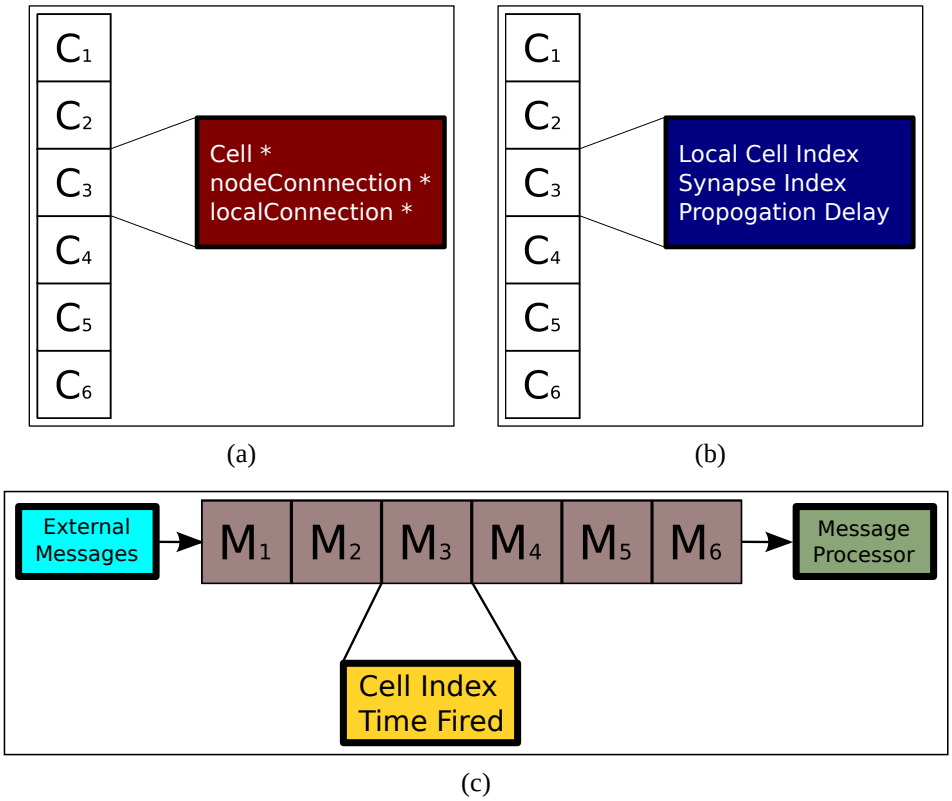



**Figure 3.1:** Basic cell data structures. (a) Cell Information Structure. (b) Fanout Information Structure. (c) External Message FIFO

The action potential generated by a cell when it fires is represented in the Neurolite simulation as a message. That message will arrive at the receiving cell, after traveling along the theoretical axonal connection, a certain amount of time after it was sent. In other words there is a finite amount of time the action potential will need to travel along the axon. This is known as the propagation delay. The data structures in Neurolite are designed to store these messages in a way that utilizes the shared memory while accurately representing that propagation delay, Figure 3.1. The intention is to create a system that passes the cell messages in the most efficient way possible. A high level description of the design of the message passing system and data structures is included below.

This begins by visualizing a group of containers connected in a row, Figure 3.2, where each container represents a moment in time. The number of containers in this group is determined by the action potential that has the longest travel time, as well as the difference in time between the containers.

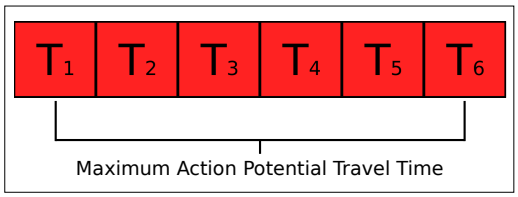



**Figure 3.2:** Group of cell containers

Each container in this group will in turn hold another group of containers, Figure 3.3. These hold messages or spiking information about a cell for that particular moment. There are identical groups stored in each container of the travel time group. It should be noted here that once the end of the data structure is reached the software will simply loop back to the beginning. This creates what is known as a circular buffer. These concepts should become clear below.

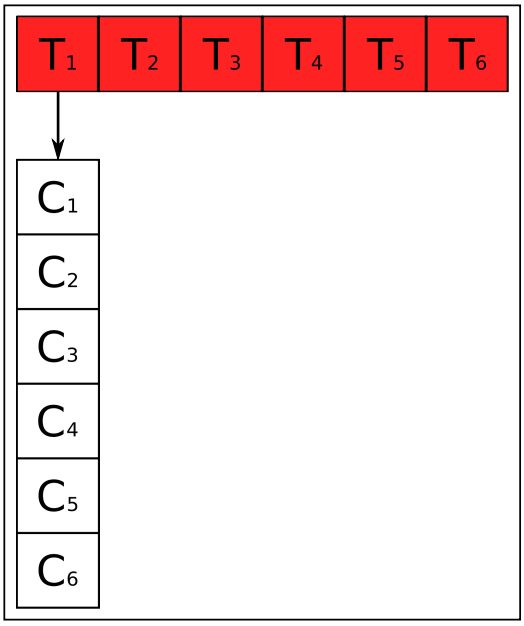
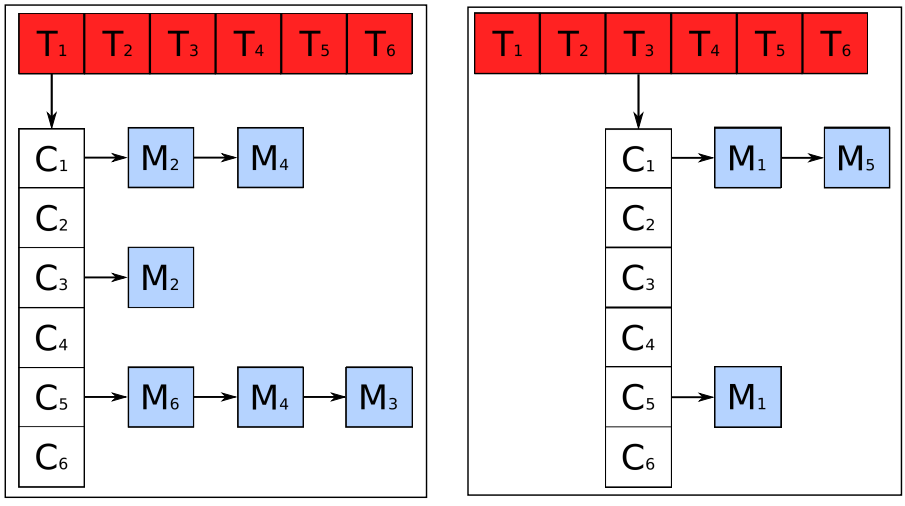



**Figure 3.3:** Cell containers for time T1

Figure 3.4a shows an example of messages contained in the cell group for time T1. Notice that action potentials can arrive from multiple cells at the same time. Similarly, Figure 3.4b, shows an example of messages stored at time T3.




(a) (b)

**Figure 3.4:** (a) Spiking messages for six cell group at T1. (b) Spiking messages for six cell group at T3.

It was mentioned before that the size of the time group was limited by the maximum propagation delay of the simulation. This is because when a cell fires the simulation will determine which cells will receive that message. It will then place messages in the appropriate spots of the structure based on the propagation delay and the receiving cell. The resulting structure ends up looking like Figure 3.4a and Figure 3.4b. With this scheme messages are placed forward in time and the maximum distance ahead is determined by the largest propagation delay. This is why the time group can be constructed using a circular buffer.

## 3.3 Shared memory design

The generic design of Neurolite lends itself to a number of hardware implementations. Presented here is an example applying these concepts to a shared memory architecture (NeuroliteSM). The design was completed with Dr. Frederick C. Harris, Jr., James Frye and the author, as part of a NCS redesign at The University of Nevada, Reno. The target architecture was the Sun microsystems Sunfire X4600.

The X4600 consists of 8 processing boards linked together with a high speed physical bus. Each board has a single processor socket as well as 8 slots for memory. The memory on each board is accessible by all of the other processors but it is physically local to one of the eight processing boards. The memory that is located on the same board as a particular core (processing unit) is considered local. The memory that is off-board is considered remote and it requires more coordination between processors to access. This can affect performance of the system and must be considered in any large shared memory design.

With a basic idea of the message data structures defined above there are two questions that need to be explored further. First, how are these structures constructed? Second,

how can the construction and processing of the messages be completed faster and more efficiently on a shared memory architecture? Also of interest are the problems that occur when more than one core attempt to access a shared region of memory. If one core is writing to a location while another is trying to read it, the core reading will not see an accurate representation of that memory location. This type of contention is handled using a locking mechanism. The software will in some manner lock out a section of shared memory before reading or writing to it. This can slow down the simulations in two ways. The first is the timing overhead required to set and clear those locks. The second is the time that can be spent waiting for a locked section of memory that is required to continue execution.

The NeuroliteSM design circumvents those issues by creating redundant message structures; one for each core. Figure 3.6 illustrates this idea for a three core system.

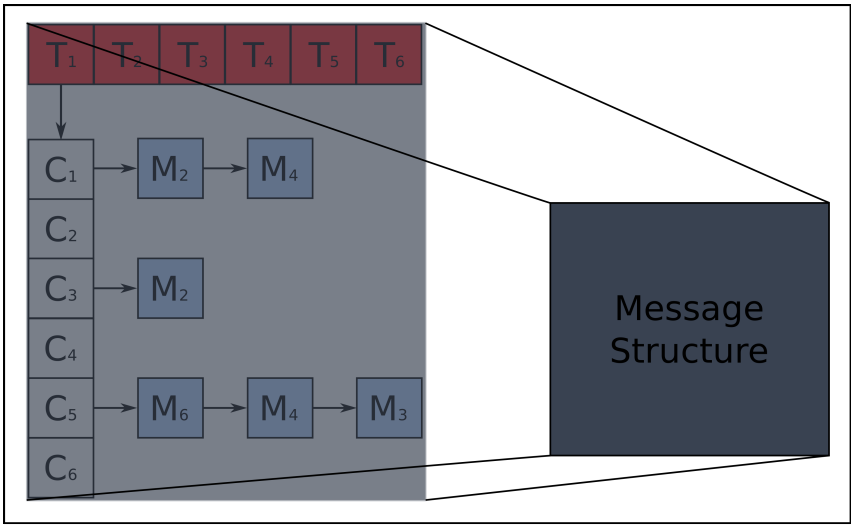



**Figure 3.5:** New message structure representation

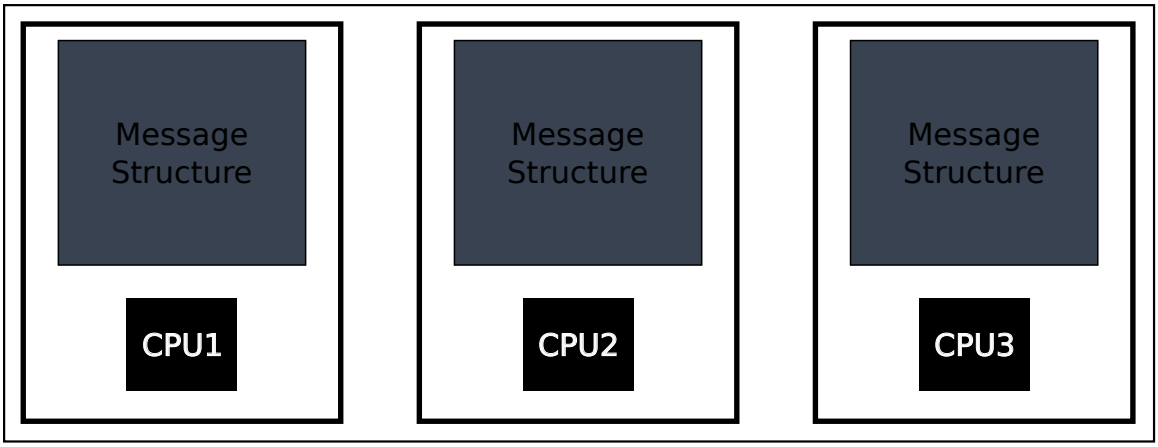



**Figure 3.6:** Redundant data structures.

During the processing stage of the simulation, a cell or group of cells is assigned to a core, shown in Figure 3.7. This core will check the containers for that cell on each of the redundant message structures. Keep in mind that although each structure is stored locally, they are globally accessible to the other cores, Figure 3.8.

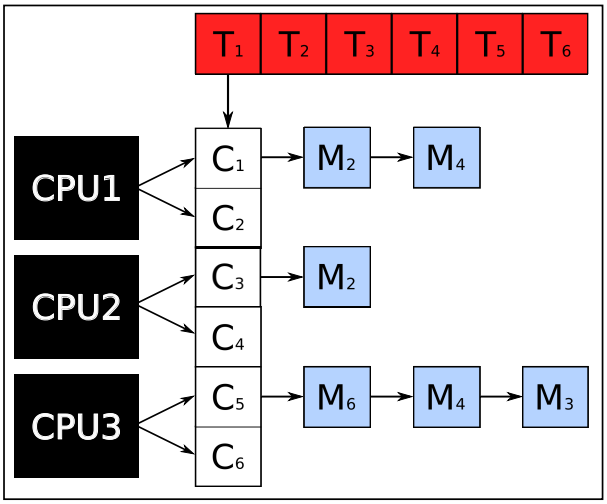



**Figure 3.7:** Work distribution.

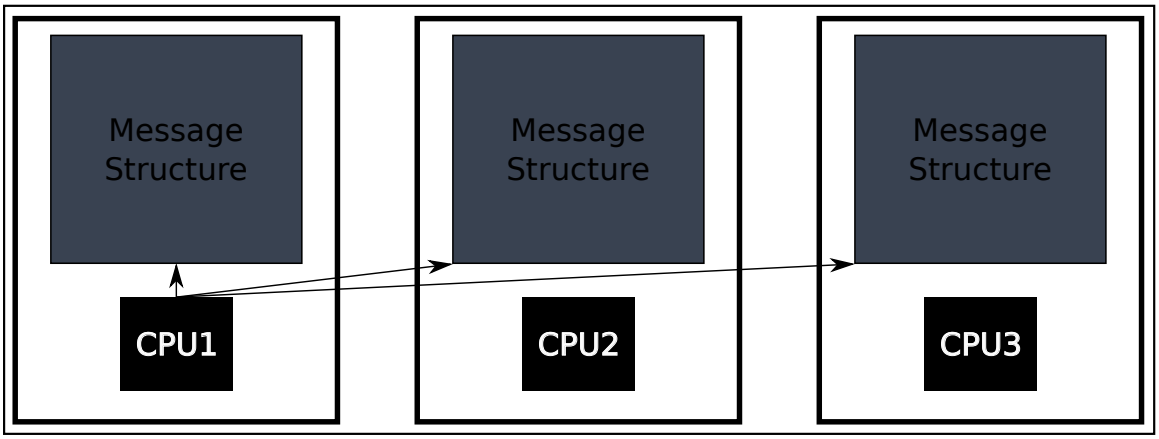



**Figure 3.8:** Each structure is globally accessible.

The issue of mutual exclusion is avoided by assigning each core a specific cell to work on. When a cell fires, the core will store all messages to its local structure; once again avoiding the need to use locks. The passing and processing of messages between cores on the same node becomes a simple local memory operation. A similar concept can used for internode communication but in those instances the strengths of the Infiniband connection would be emphasized.

This design trades memory usage for speed; because the redundant structures require large amounts of memory. With those structures the message passing can be done in a very clean and efficient manner; the passing of a message becomes an address change. On distributed compute clusters the message passing functionality obviously changes but the overall data structures remain similar. This provides a mechanism for code reuse and offers a high-level of extensibility.

## 3.4 Single node GPU simulation

Recently, the utilization of graphics processing units (GPUs) for scientific computing has increased dramatically. Originally intended as a means of offloading graphics and visualization tasks away from the central processing units (CPUs), the single instruction, multiple data architecture of GPUs lends itself to many scientific computing problems. As one of the leaders in graphics chip design, NVIDIA has invested considerable resources in providing the scientific community with both hardware and software solutions aimed at leveraging their products for just such applications. The Compute Unified Device Architecture (CUDA) created by NVIDIA provides developers with a relatively simple instruction set as well as comprehensive tools for working in a GPU environment.

The proof-of-concept simulation code described here is presented as an illustration of both the scalability of the design and the performance potential of GPUs. As an unoptimized prototype, it is in many ways a worst-case scenario. Only GPUs within a single compute node are supported. However, even in this immature state, the design lends itself to the addition of message passing between compute nodes. This prototype supports a simple input file format that at present is generated by a separate program. Once the input file has been read in, the program executes the steps outlined in Figure 3.9.

This work was originally published in Thibeault et al. (2011) and was designed by Roger Hoang and the author. The development of the simulator was carried out by Roger Hoang, while the model construction code and the benchmarks were completed by the author.

### 3.4.1 Design

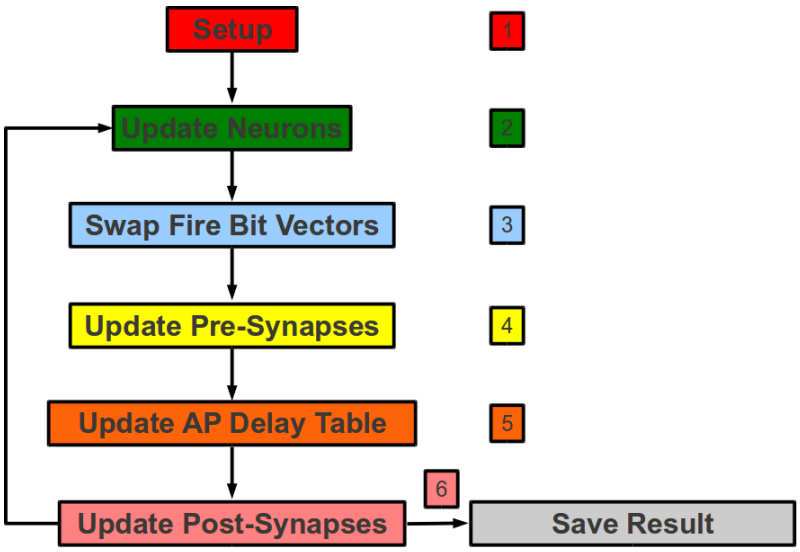



**Figure 3.9:** Simulator Execution Flow

The simulation setup begins with a redistribution of the input model. The neurons are sorted based on the number of synaptic connections. These are then distributed to the respective GPUs in a round-robin fashion; providing a first-pass load balancing of the model. Once the neurons have been distributed each GPU forms a local indexing and representation of its neurons. The new indexing scheme is shared amongst GPU threads and is used to develop the local neuron structure array and the Cell Firing Bit Vector, as shown in Figure 3.10. In this implementation the Cell Firing Bit Vector is a representation of the entire neural model at the current simulation time tick. In future version this will be restricted only to Neurons that are of interest to a particular GPU thread.

The Local Synapse array is constructed in a similar manner with the synapses being grouped by their presynaptic neural connection. This layout provides a contiguous region of memory that can be accessed with minimal overhead within the GPU architecture.

Finally, the Action Potential Delay Table is constructed. This is a bit vector that pro-

vides a mechanism for simulating the propagation delay of action potentials. As seen in Figure 3.10, the X axis represents the local synapse's axonal connection. The Y axis is a circular buffer that is the size of the maximum propagation delay.

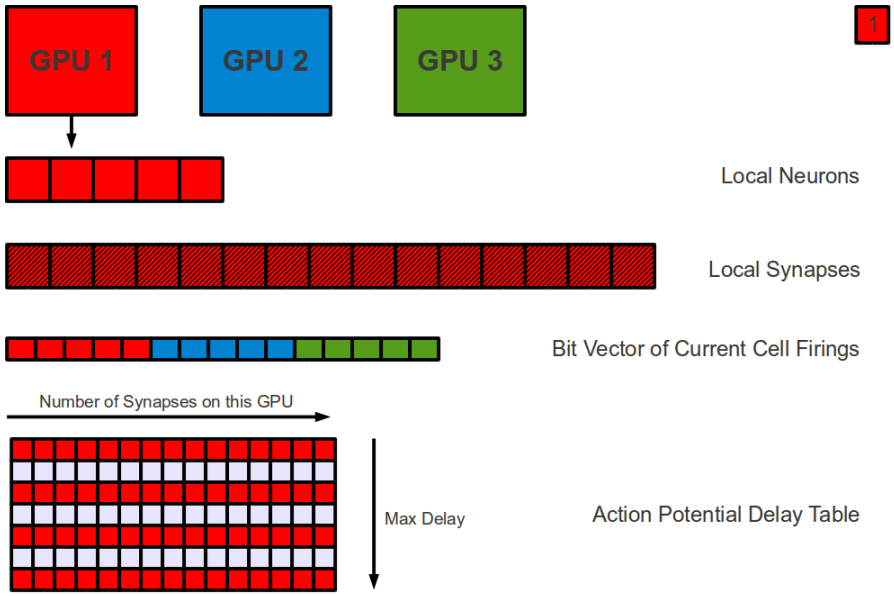



**Figure 3.10:** System Setup

After setup, the simulation begins by updating (numerically integrating) the neurons. The appropriate region of the Action Potential Delay Table is read and the number of "1" bits are noted. In this context, a "1" bit represents an action potential that has arrived at that particular synapse. The neuron code then samples the electrical current contributed by that synapse. After the entire Delay Table for the current time tick has been read, the voltage of the cells are computed numerically using a forward Euler method. If the cell reaches threshold and fires an action potential, its corresponding bit in the Current Cell Firings Bit Vector is set high.

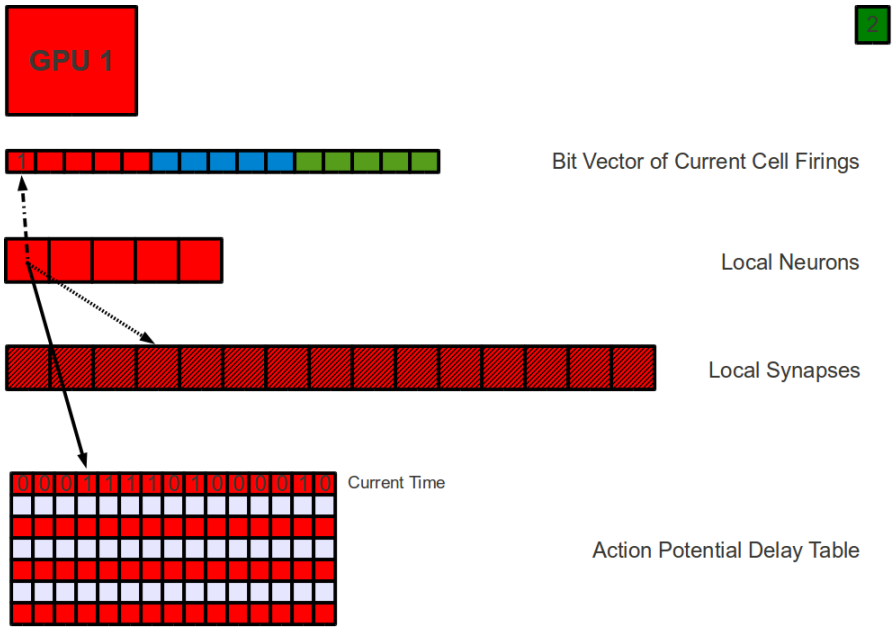



**Figure 3.11:** Update Neurons

Once the neurons have been updated and the Cell Firings Bit Vector has been filled in, the GPU threads will pass a copy of the vector to the other GPU threads. This is illustrated in Figure 3.12. The layout of this vector for each of the respective GPUs should be noted. This was described in the setup above and is a result of the redistribution of neurons.

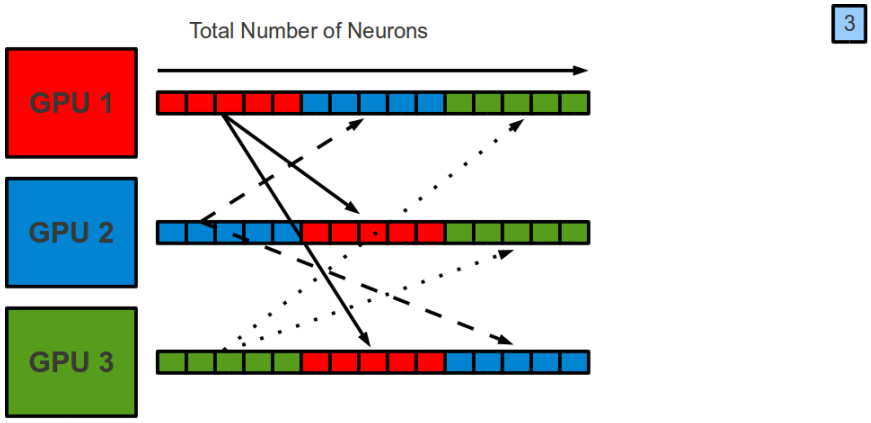



**Figure 3.12:** Swapping of the Current Cell Firings Bit Vector

The synaptic updates begin by reading the respective Current Cell Firings Bit Vector,

shown in Figure 3.13. Negative synaptic learning can be calculated and the Action Potential Delay Table can be updated at this time.

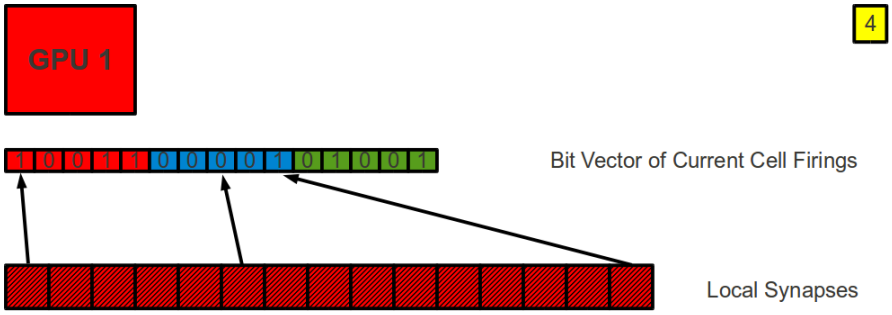



**Figure 3.13:** Update Pre-Synapses

Shown in Figure 3.14, the Action Potential Delay Table is updated for the Neurons that have fired. The appropriate bit, based on the delay specified by the model, is set high.

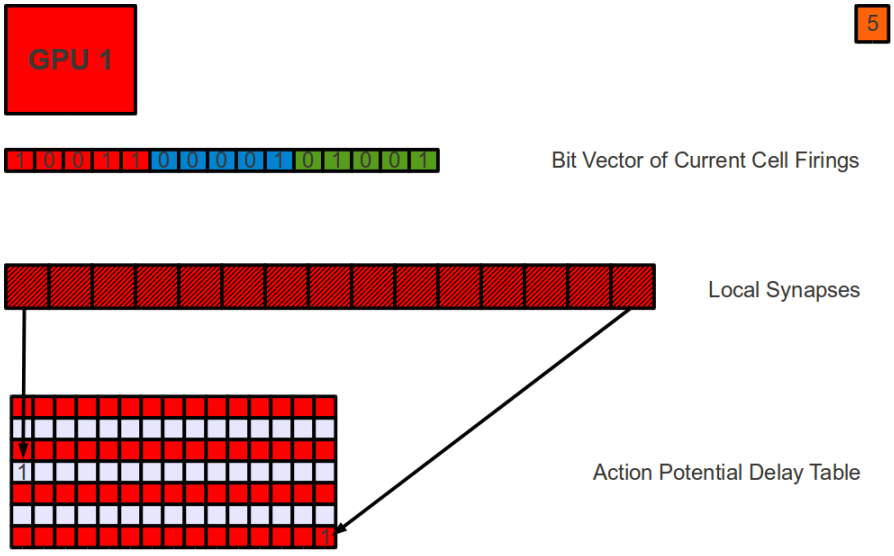



**Figure 3.14:** Update Action Potential Table

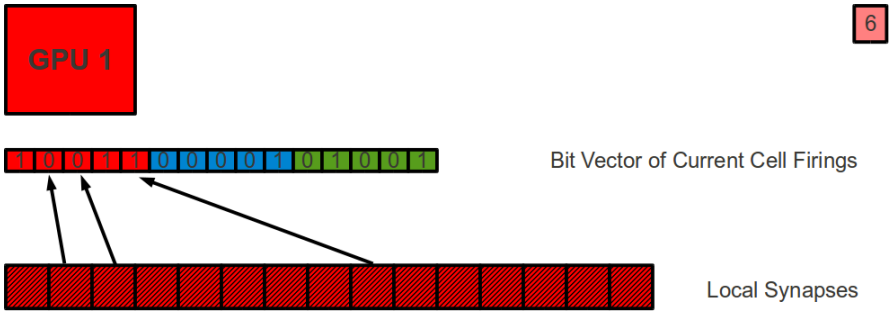



**Figure 3.15:** Update Post Synapses

Finally, the synapses sample their post synaptic neuron and use the result to calculate positive learning if needed.

At this point, a producer-consumer thread model will grab the current cell firings vector and begin writing it to the output file. Concurrently, if needed, the Neuron Update step starts the process all over again.

### 3.4.2 Example performance

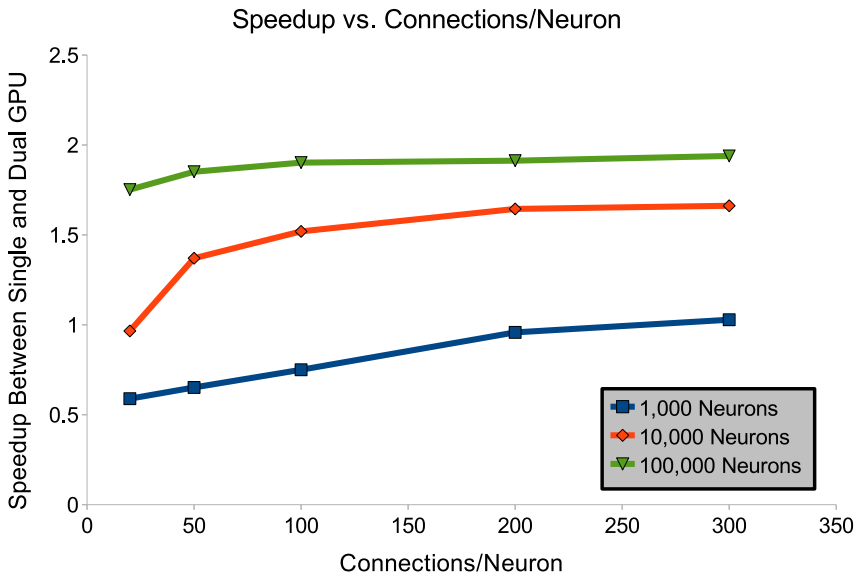



**Figure 3.16:** Speedup vs. Connections/Neuron

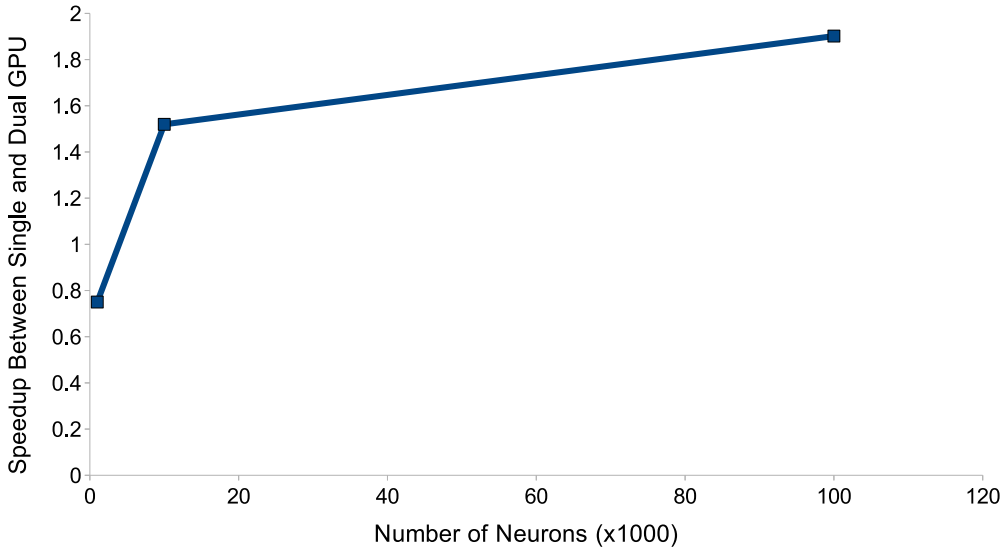



**Figure 3.17:** Speedup vs. Number of Neurons (100 Connections per Neuron)

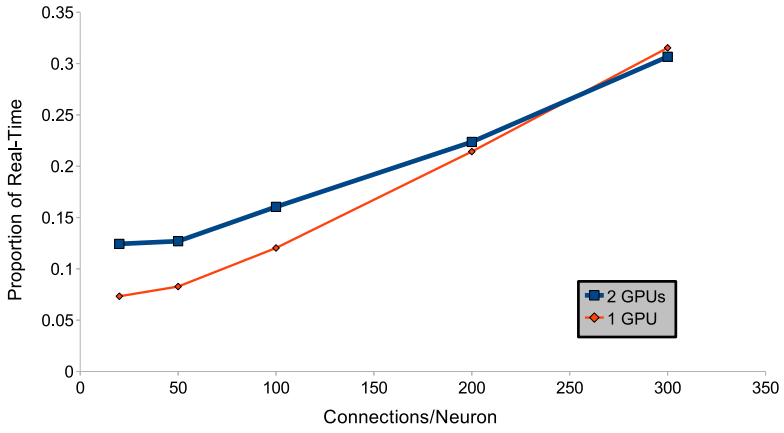
Presented in Figure 3.16 are the speedup results between one and two GPU simulations. For networks of only 1000 neurons there is no advantage to moving the simulation off of a single card. As the network size increases there is a non-linear increase in speedup that can be seen in Figure 3.17. As the Number of neurons and connections increases the advantage of two cards approaches the ideal speedup of two.

Also of note is the amount of data transferred between GPUs. Table 3.1 illustrates the small amount of information required at each time step. Based on the small bandwidth requirements of the design and the linear dependence on the number of neurons, the addition of hardware should provide a near linear increase in speedup. This is of course dependent on the model sizes as illustrated by these benchmarks.

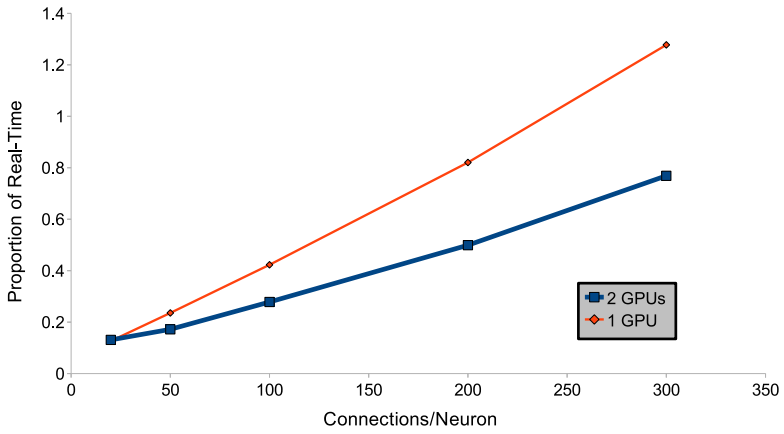
**Table 3.1:** Total Data Transfer between GPUs at each time step.

| Neurons | Data/Time step (Bytes) |
|---|---|
| 1,000 | 125 |
| 10,000 | 1,250 |
| 100,000 | 12,500 |
| 1,000,000 | 125,000 |

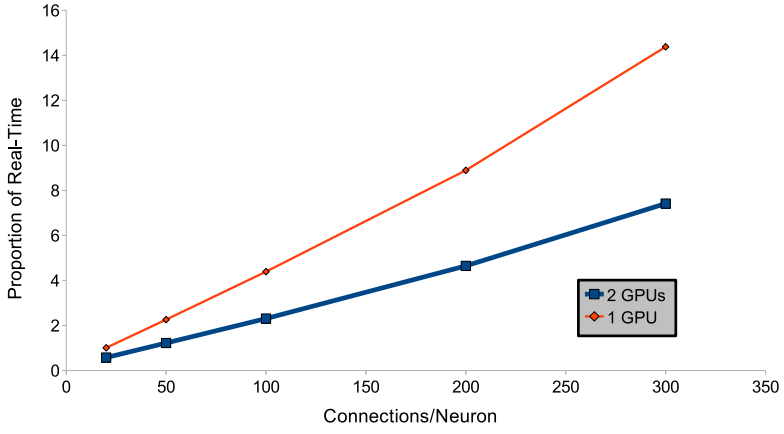
Figure 3.18 illustrates the real-time capabilities of the prototype simulator. Once again as the number of connections per neuron is increased the advantage of multiple GPUs is enhanced. From this a model with 100,000 Neurons and 50 connections per neuron can run at about 1.2 times real time. We are confident with some basic optimizations that these numbers will be drastically improved.




(a)




(b)




(c)

**Figure 3.18:** Timing comparison for number of neurons vs. number of connections per neuron. (a) 1,000 Neurons., (b) 10,000 Neurons., (c) 100,000 Neurons.

## 3.5 Cluster based GPU simulation

Moving neural simulations to high-performance cluster environments offers the potential for substantial speedup compared to CPU based clusters. Making that move however, is not straightforward and there are many different design choices that can affect the performance. We present here a review of the unique design patterns used for a general large-scale neural simulation framework. Named HRLSim, it was designed for both parallel CPU architectures and parallel General-Purpose Graphics Processing Unit (GPGPU) super-computers. In addition, example benchmarks are presented to illustrate the potential of the framework.

HRLSim development was started by Aleksey Nogin, Youngkwan Cho and Michael J. O'Brian. This work was then overhauled and extended by Kirill Minkovich, who was later joined by the author in that effort. The motivation for creating another neural simulator was driven by the need to support the neuromorphic hardware of the DARPA SyNAPSE project (DARPA, 2012). The goal of which, is to implement in a square cm of CMOS, $10^6$ neurons with $10^{10}$ synapses and an average of $10^4$ synapses per neuron. Recently, as part of this effort, a compiler for the automatic translation of a given neural architecture into custom neuromorphic hardware was published by Minkovich et al. (2012). HRLSim was developed to support the neuroscience research of the SyNAPSE project and create the input into that neuromorphic compiler.

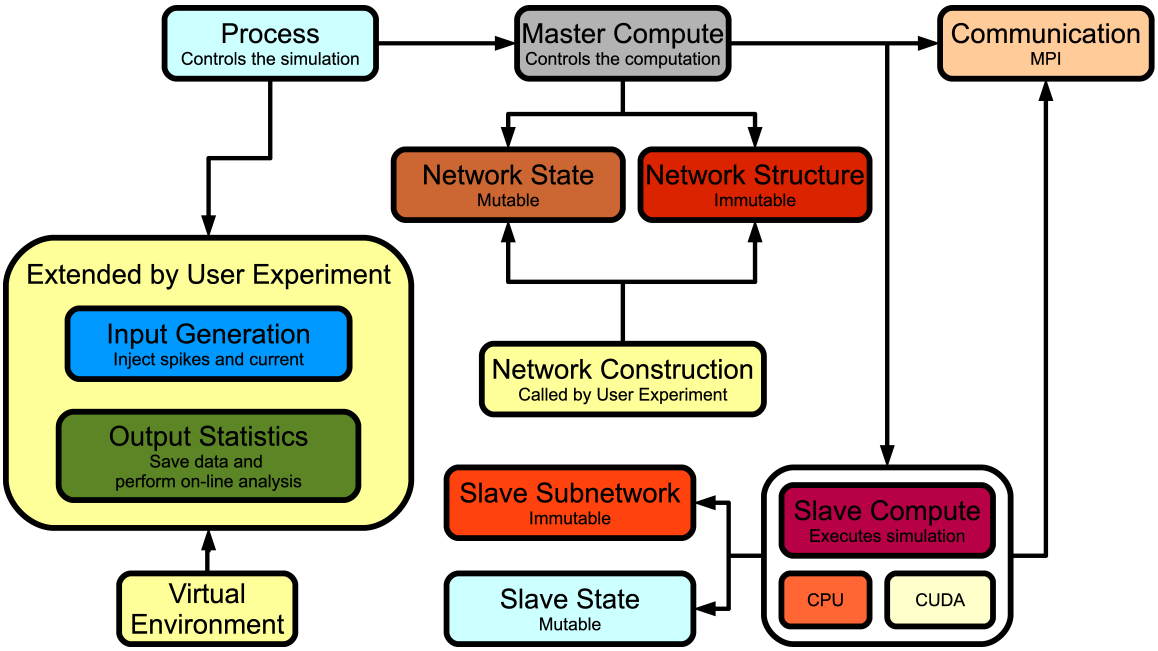



**Figure 3.19:** Interactions of the HRLSim neural simulator modules.

Models in HRLSim are created in C++ and then compiled into an executable. By defining experiments this way the performance optimizations of the compiler can be exploited. Unused code is automatically removed and other optimizations such as loop-unrolling and value precalculation can be performed.

Figure 3.19 demonstrates the relationship between different modules of the simulator. A simulation begins with the main process instantiating a Statistics object, an Input object and a Master Compute object. The master compute object instantiates the Network and User Experiment which in turn construct the Network and Network State objects. The Network is then split into subnetworks and passed to Slave Compute modules using Communication objects.

The flow of the simulation is controlled by the master process. Inputs and outputs are facilitated by calls to virtual functions in the user experiment base class. The heavy lifting of the simulation is done by the slave nodes. The master node is then responsible for

performing user supplied task and recording the network state, spiking activity and synaptic weights.

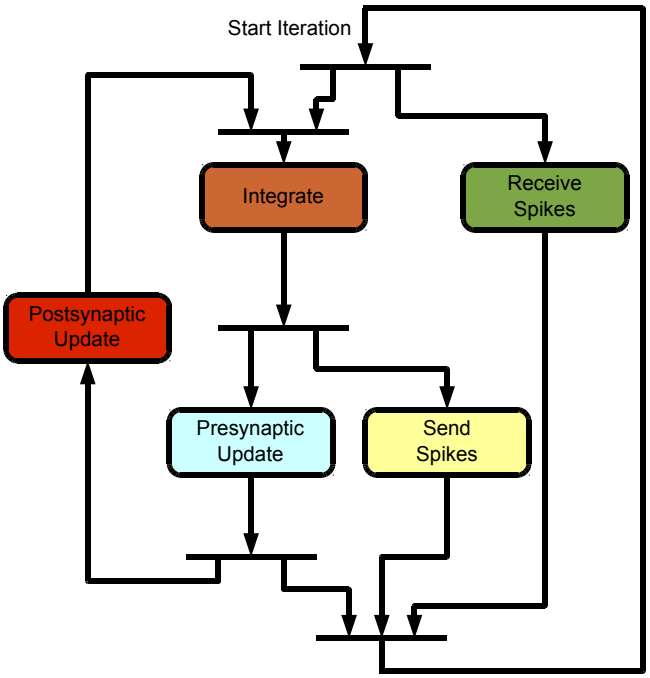



**Figure 3.20:** Flow chart for a single iteration.

Many of the operations of neural simulations can be performed in parallel to each other. Figure 3.20 illustrates how these bulk tasks are executed in HRLSim. The individual steps in a single iteration are

1. The communication thread starts receiving incoming spikes.
2. The computation threads waits for the synaptic updates from the previous iteration to finish.
3. The computation thread performs integration to generate the outgoing spikes.
4. The communication thread starts transmitting the outgoing spikes.
5. The computation threads starts the synaptic updates.
6. The communication thread waits for incoming spikes to be received and the outgoing spikes to be sent.

This parallelization maximizes the amount of overlapping computation and communication. Additional strategies employed by HRLSim are explored further below.

### 3.5.1 Network layout

Even with the C++ preprocessing, anticipating the network configuration at that time is unreasonably complicated. To compensate for this, the initial network is flattened into vectors occupying contiguous regions of memory. The network is reduced to three vectors, one containing the number of outputs for each neurons, one containing the indexes into the synaptic connection vector and a vector describing the post-synaptic connections of each neuron. This flattened system allows for optimized memory access and favors synaptic computations, which take up much of the computational time of a simulation.

### 3.5.2 Delayed STDP

There are 1000 times more synapses in the brain than there are neurons. The computation can be dominated by updating synaptic variables. Some of the slowest synaptic calculations are the STDP parameters, however they are only needed when a neuron fires an action potential. Rather than calculating these at each time step, HRLSim delays the computation until it is actually required. This avoids having to update synapses at each time step and removes the simulator's performance dependence on the number of synapses. In CPU simulations this is further optimized by using either precomputed values or approximations to the exponential decay functions. Initial benchmarks demonstrated a $38\%$ faster simulations.

### 3.5.3 Dynamic method selection

GPU performance is heavily dependent on memory accesses. Performance is highest when those accesses are aligned, so threads access consecutive memory blocks. This is difficult to predict when memory access is dependent on network activity. An intelligent selection method was added to HRLSim to provide a way to switch between two different postsynaptic updates, each with performance characteristics that are dependent on the activity. The first one accesses only the neurons that just fired. The second method iterates over the entire subnetwork of neurons updating only those that just fired.

To illustrate the benefit of switching between these, a network of $100,000$ neurons was simulated for $30$ seconds. When only the neurons that fired were accessed the simulation completed in $26$ seconds. It took $167$ seconds to complete if the simulation iterates through all of the neurons. However, when the simulator dynamically switches between the two, based on the number of neurons that fired, the simulation completed in $21$ seconds.

### 3.5.4 Kernel parallelization

As outline in Figure 3.20, the results of many of the computations are not needed until later in the current iteration or the subsequent iterations. Because of this, multiple CUDA streams are used to build queues of required operations. This allows much of the computation to be completed in parallel by employing the native CUDA mechanisms.

#### Memory layout

To optimize memory access, data structures are aligned along 128-bits. This is done because single instruction reads can be up to 128 bits wide in CUDA; avoiding overlapping memory calls. In addition to this, memory allocations are based on assumed sizes, rather

than the maximum possible. As a result, the memory usage is greatly reduced and there is only a minimal overall penalty from rare memory resizes.

### 3.5.5 Communication message packing

Chapter 4 deals with exchanging spikes but before that can happen the message packets need to be constructed. To optimize the increase in spikes generated by a GPU simulation compared to a CPU one, much of the heavy lifting is done on the GPU. A special kernel is employed to efficiently pack the spikes.

### 3.5.6 Example benchmarks

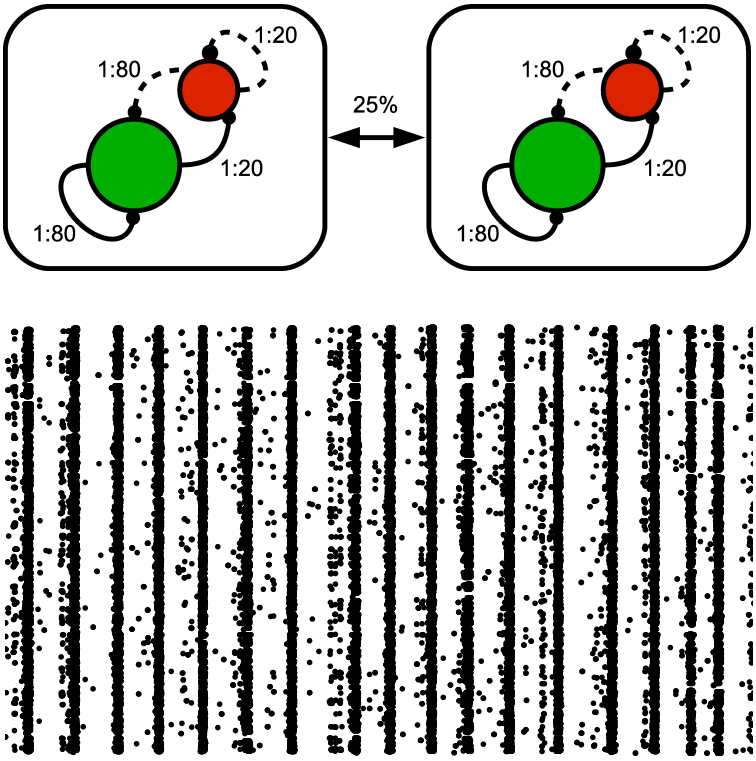



**Figure 3.21:** (Top) Two 80%excitatory / 20% inhibitory networks. (Bottom) Raster plot of 2000 neurons for 10 seconds from one of these networks.

A set of experiments that create an uneven distribution of work were created to demonstrate the capabilities of the HRLSim environment. The worst case scenario for a large-scale simulation is when one node is doing a disproportionate amount of work. The networks created for the benchmarks here do exactly that. These are weak-scaling, in that the number of neurons and synapses increases linearly with the number of nodes. Each node simulates a balanced inhibition network consisting of $80\%$ excitatory neurons and $20\%$ inhibitory neurons, 3.21 (top). These are connected to other networks with a probability of $25\%$. To avoid the activity of the overall network getting out of control the nodal networks are connected with a synaptic weight of zero. This forces a spike to be passed but does not affect the activity of the network.

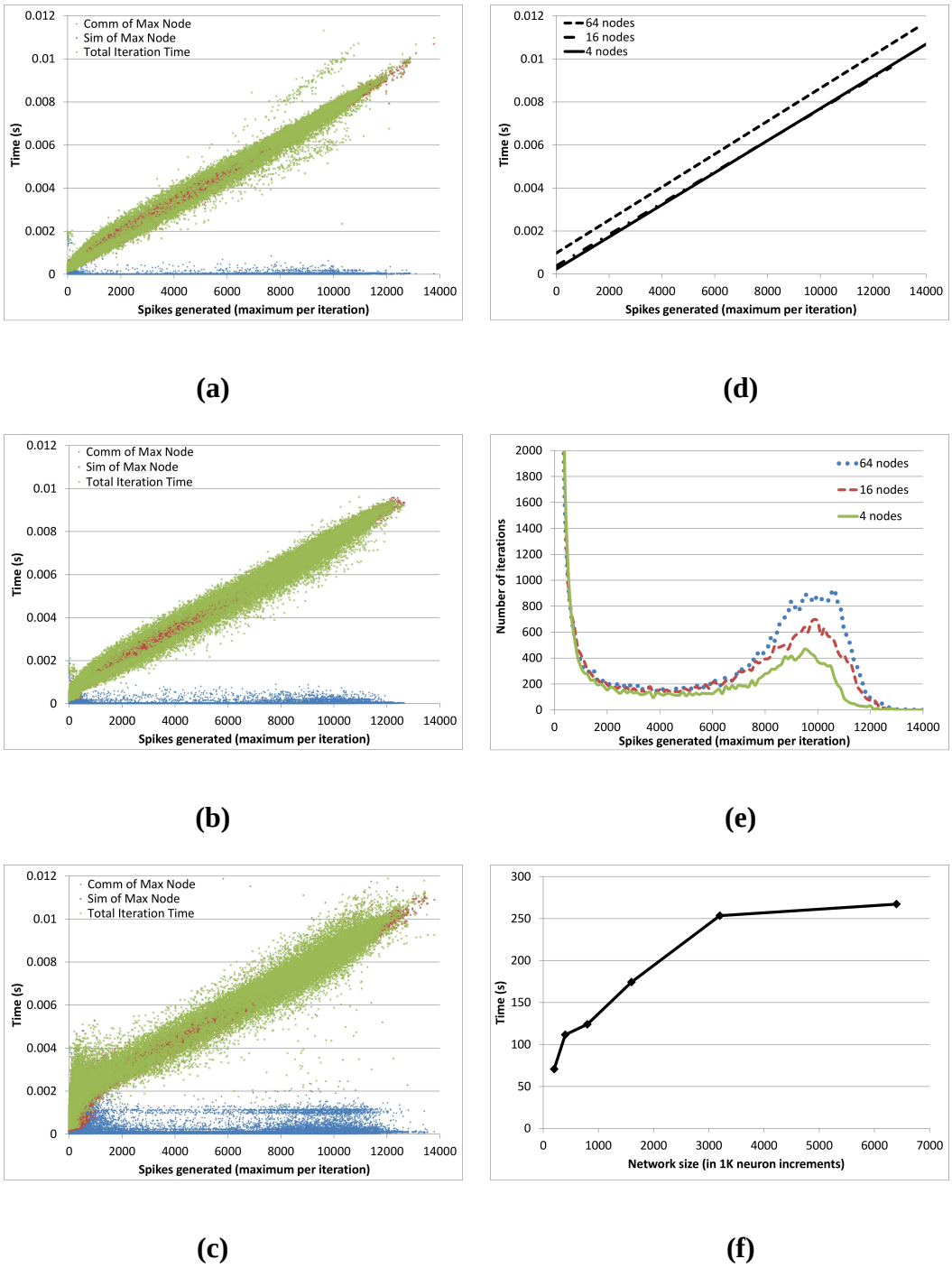



**Figure 3.22:** GPU Results, 100,000 neurons per node. (a), (b), and (c) show the runtime distribution on 4, 16, and 64 nodes, respectively. (d) shows the linear regression for plots (a), (b), and (c). (e) shows the histogram of maximum spikes per iteration. (f) shows how the runtime scales to network size.

The difficulty in simulating these networks is that the banded firing of Figure 3.21 (bottom), is uncorrelated between nodes. Compute nodes randomly becomes the most active of the cluster, slowing down the entire simulation. When examining the maximum amount of spikes generated on each node, shown in Figure 3.22 (e) there is a clear trend that shows the more nodes being used the more spikes that are generated (when examining the maximum spikes generated per iteration). The evaluation was performed on a cluster of 92 compute nodes, each with two Intel Xeon E5520 2.27GHz CPUs and two NVIDIA Tesla C1060 cards, with Infiniband communication.

The benchmarks were run on 2, 4, 8, 16, 32, and 64 GPU cards where each GPU card simulated a 100K neuron network. Figures 3.22 (a), (b), and (c) illustrate that the total time (green) scales linearly with the simulation time (red) and the communication time (blue) remains relatively constant.

The penalty from the increase in the number of nodes is observed in the trend line of Figure 3.22 (d). This can actually be explained by the histogram in Figure 3.22 (e). As outlined above, when the number of nodes increases so does the probability that one of them is running slowly.

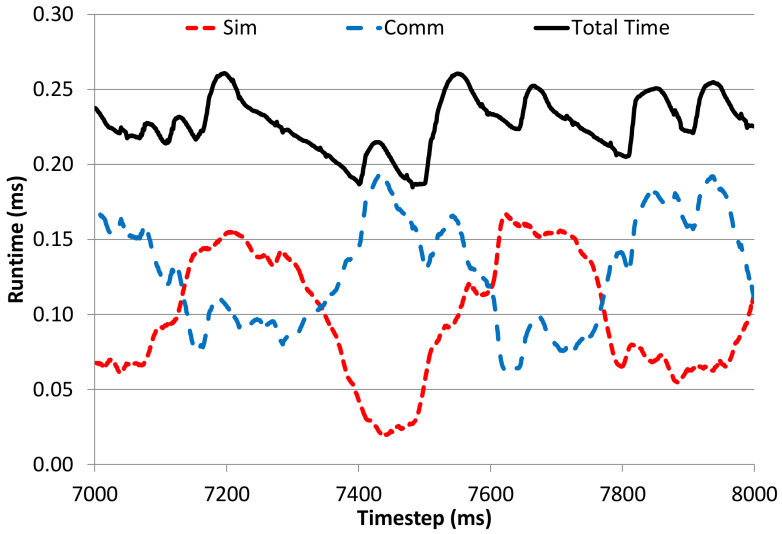



**Figure 3.23:** Example timing results for benchmark model execute on 16 nodes showing how simulation time and communication time correlate to the total time. The execution times were summed across all the nodes.

Figure 3.23 illustrates how well the computation and communication tasks are threaded. When there are a low number of spikes the computation takes a small percentage of the total simulation time and the communication dominates the performance. However, when the spiking activity increases the communication cost goes down since it can be effectively overlapped with the increased computation time.

## 3.6 Discussion

The designs presented here represent the choices made in several different simulation projects. They are a sampling of the neural simulator design space and as such, do not offer a complete view of the available neural simulators. The factors in designing these environments are dominated by two very important details. The first is the target level of abstraction. Although some environments, such as NEURON (Hines and Carnevale, 2007)

and GENESIS, (Bower and Beeman, 1998), cover a large range of physiological detail, most environments target specific segments of neural modeling. The high-performance simulators, such as those presented here, are dominated by point neuron and sub-threshold implementations. Performance is less of a concern in the detailed neuron simulators where only a small number of neurons are modeled.

The second factor is the target hardware infrastructure. The GPU based environments reported here are all exclusive to NVIDIA's CUDA platform. This restricts the potential audience of these environments and also helps explain previous simulator's specificity to the designer's personal project. The CPU based projects are more general however, an interesting trend can be seen where many of these are implementing performance optimizations for specific hardware (Plesser et al., 2007; Hines et al., 2011). When creating a new simulator there are many things that need to be considered. Some of the more important aspects are outlined below.

### 3.6.1 Extensibility

A trend of the models presented here is that the more extensible the design, Section 3.2, the more performance suffers. The GPU simulators, the highest performing designs, currently lack scalability. Adding new features to either design is tedious and difficult to implement. While porting these implementations to new hardware is almost completely unreasonable. The trade-off between performance and scalability is a choice that either complicate the design or restrict the environment to specific hardware and features.

Projects like OpenCL (Group et al., 2008) can abstract away some of the hardware dependence but that comes at cost in performance. Additionally, template based designs can improve extensibility but that brings an associated increase in code complexity and again, can introduce decreased performance.

The obsession with simulator performance is in direct conflict with the need for extensibility. The reality, however, is that users are going to continually request new features and code bloat is inevitable in any long-life software project. In addition, the field of neuroscience continually presents new information on brain function. Extensibility is an absolute requirement for remaining state-of-the-art.

### 3.6.2 Model sharing

Learning to use a given neural simulator is surprisingly difficult. Many users take the time to learn the specifics of a single one and stick with that, choosing familiarity over features or performance in some cases. This allegiance to a specific simulator is not necessarily bad but sharing models between researchers becomes a problem. If a model a researcher is interested in is not implemented on their simulator of choice, porting it can be an arduous task. Similarly, validating the results of a simulation study are now tied to a single environment, preventing others from building on existing research. The numerous simulation environments are a benefit to the field (Djurfeldt and Lansner, 2007), but the lack of model sharing is not.

Standardized interface projects such as PyNN (Davison et al., 2009) and NeuroML (Crook et al., 2007) are aimed at alleviating some of the problems with model sharing as well as the usability of simulators. With a common interface, researchers can take the time to learn the idiosyncrasies of a single language but can have access to many different neural simulation environments. As general simulators are planned and developed, including one of the standards is important for improving the chances of adoption in a saturated field.

# Chapter 4

# Efficient Spike Exchange in Distributed Neural Simulation

Efficiently passing spiking messages in a neural model is an important aspect of high-performance simulation. As the scale of networks has increased so has the size of the computing systems required to simulate them. In addition, the information exchange of these resources has become more of an impediment to performance. In this chapter we explore spike message passing using different mechanisms provided by the Message Passing Interface (MPI). A specific implementation, MVAPICH, designed for high-performance clusters with Infiniband hardware is employed. The focus is on providing information about these mechanisms for users of commodity high-performance simulators. In addition, a novel hybrid-method for spike exchange implemented and benchmarked.

## 4.1 Introduction

The highly distributed nature of the animal nervous system presents a unique challenge in theoretical and computational modeling of neurobiology. Whether these models are intended to provide a better understanding of biological function or to build more intelligent agents, the comparatively limited parallelization inherent in all modern computing architectures must be overcome to achieve models that accurately represent the highly parallel nature of biology. The current computing and software paradigms have prevented truly scalable neural models that can faithfully simulate biology in reasonable amounts of time. In addition, a compromise between biological realism and performance must be made. This is a concession that is often unacceptable to the overall performance of the task.

There are two major steps in simulating the nervous system, incrementally solving the governing equations and communicating the results to other parts of the system. We previously presented ways of improving the performance of the former by parallelizing the computations on clusters of General Purpose Graphical Processing Units (GPGPU) (Minkovich et al., 2012). The purpose of this work is to demonstrate where the spike communication can be optimized on generic high-performance computing architectures.

The effort to efficiently simulate spiking neural networks has a long history that spans hardware implementations (VLSI and FPGA) and the more popular highly distributed compute cluster implementations. Although hardware options are increasing in popularity with projects like SPINNAKER (Furber et al., 2012) and SyNAPSE (Merolla et al., 2011; Srinivasa and Cruz-Albrecht, 2012), they still cannot compete with the practicality and flexibility of generalized simulators. Even the aforementioned hardware options are generally supported by high-performance distributed simulation environments.

Recently, Hines et al. (2011) explored several different spike exchange methods on an

IBM Blue Gene/P (BP/P) cluster and concluded that point-to-point communication using the built-in standard Message Passing Interface (MPI) non-blocking MPI_Isend was the worst performing method. Of the top performing methods of that work, the MPI collective routine, MPI_Allgather, was among the best; often with simulation times comparable to the BP/P specific direct memory access routines.

Hardware such as the BG/P provide unprecedented performance per watt but come with a price point that can be out of reach to most computational neuroscientists. Because of this, commodity clusters using Commercial Off-The-Shelf (COTS) components are more prevalent in research labs. With the availability of GPUs, the architecture of COTS clusters has changed considerably. Unlike the BG/P architecture were there can be over 100,000 processors linked together, GPU based COTS clusters have much higher processing capabilities on a single node that share a common link to the rest of the cluster. The dense parallelization available on a single node allows for a much larger number of computations but results in a communication bottleneck as more information must be shared between nodes.

Morrison et al. (2005) presented a generic architecture for distributed neural computation. In that, they contend that the amount of time spent in communication is small compared to the amount of time required to update the neurons. This appears to be a reasonable statement so long as the number of compute nodes is small. However, as both the number of compute nodes and the number of neurons simulated increases the amount of time spent in communication becomes significant.

## 4.2 Methods

When distributing the network simulation, different portions of the model are simulated by separate computers in parallel. The neural model is integrated at each iteration, and the spiking information is sent to all of the neurons connected to those that fired.

Ideally, when parallelizing the simulation of spiking neural networks, the computational cost of the mathematical integration and synaptic computations is balanced with cost of communicating information between nodes (single computers within a cluster). Historically, as mentioned above, the communication time was significantly lower than the compute time. With the introduction of higher-performance architectures such as General Purpose Graphical Processing Units (GPGPU) and specialized neural hardware systems, this is no longer the case. However, the way spiking information is sent has not changed.

Almost all hardware and software simulation environments use a variant of address event representation (AER) (Boahen, 2000). The simplest and most efficient form of this is when a neuron fires an action potential, the neuron's unique Id number is sent to all of the nodes that contain post-synaptic neurons connected to the one that fired. In general, all of the neurons that fire during the current iteration can be collected and sent as a single packet to all of the connected nodes.

As the number of neurons that fired increases, the size of the data packets correspondingly increase. In this case, the time spent in communication is a direct correlation to the number of neurons that fired. Similarly, as the number of compute nodes increases so does the number of packets that need to be sent. In some cases, for both software and hardware based systems, this can prevent scaling up to desirable model sizes.

### 4.2.1 Dummy neurons

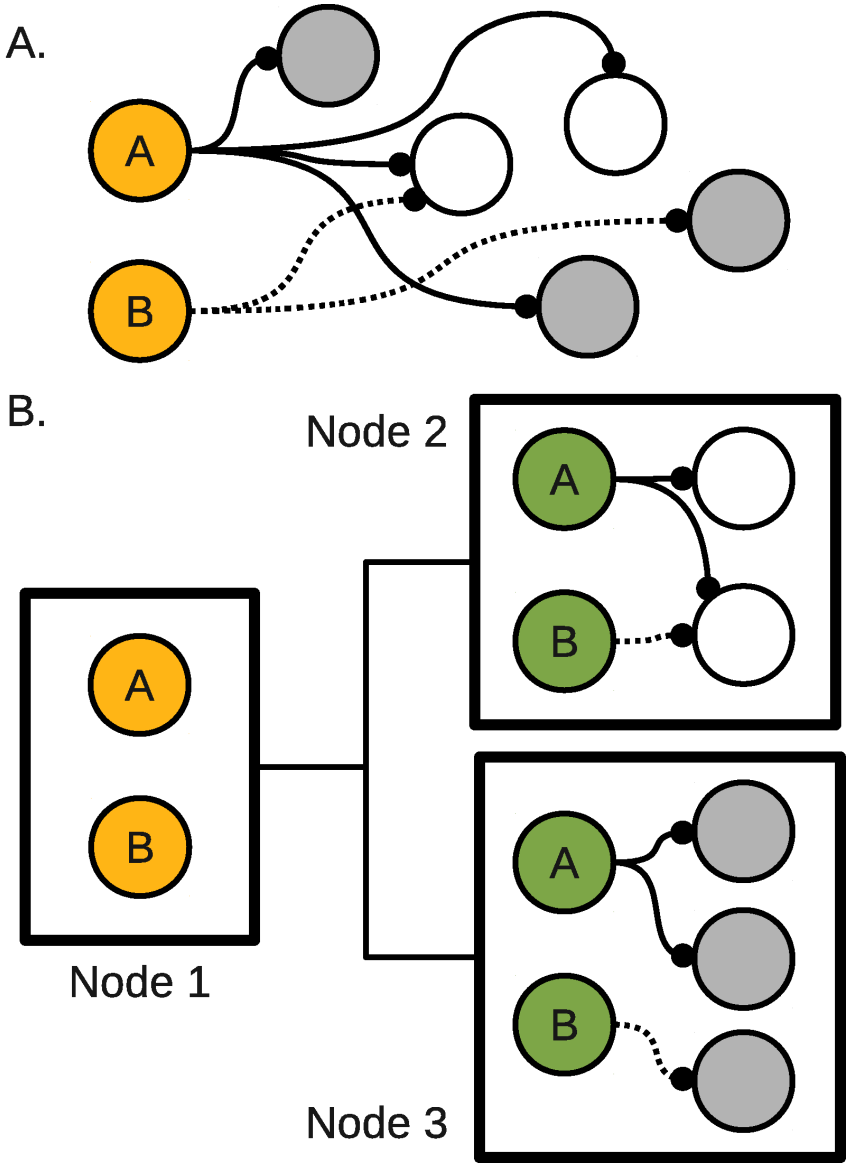



**Figure 4.1:** Dummy neurons. A. An example network connection. B. Distribution of the sample network among three nodes.

HRLSim uses the concept of dummy neurons to not only reduce the amount of information distributed for a spike event but also the complexity of updating the synaptic weights. Dummy neurons are essentially copies of pre-synaptic neurons that are located on remote

compute nodes. These copy neurons receive the spiking information from the remote neuron and then relay that to all of the locally connected post-synaptic neurons. In addition, the pre-synaptic information is computed at the dummy neurons, rather than on the remote node. This scheme is illustrated in Figure 4.1.

### 4.2.2 Rate independent message passing

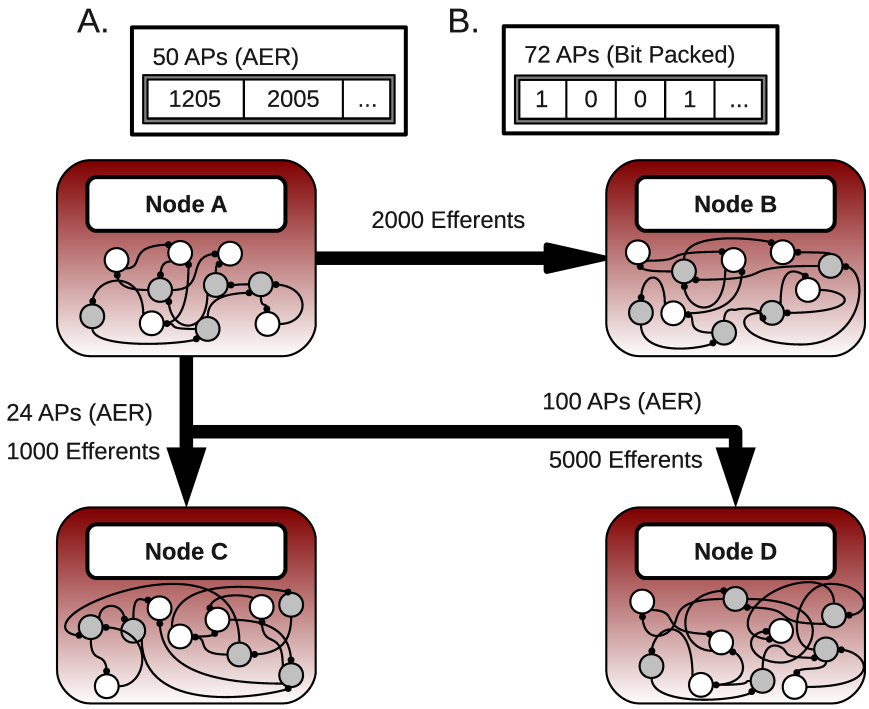



**Figure 4.2:** Hybrid message passing. A. The number of neurons that fired are below the threshold. B. When the number of neurons increases past the threshold the simulator automatically switches into the bit-packed mode.

The novelty of the hybrid message scheme lies in its deterministic performance, ambiguity to neuron firing rates and scalability greater than traditional message passing methods. At its core, the method reduces the firing information down to single bits in a packet. Essentially, each output neuron is represented by a single bit; where a ’1’ indicates that neuron fired, ’0’ indicates it did not. The key is that this is only done when firing rates are

high enough that it will reduce packet sizes. In addition, this bit-packing scheme is only performed between the nodes that meet the fire rate requirement.

Consider the case shown in Figure 4.2. There are four compute nodes each simulating a group of neurons. Focusing on node A, suppose that there are 2,000 neurons with projections to node B, 1,000 neurons with projections to node C, and 5,000 neurons with projections to node D. The maximum communication cost associated with transferring action potentials between the populations and the remote nodes is now a function of the population size. The theoretical cutoff fire rate is also a function of this as well.

The cutoff rate is the point where it is computationally cheaper to represent the neurons in a bit-packed notation compared to traditional AER. This transition point is shown through an example below and in Figure 4.2. Suppose that on Node A at a particular iteration, 50 neurons connected to B fire an action potential, 24 neurons connected to C fire, and 100 neurons connected to D fire. In this case the AER scheme is used to communicate between all nodes. If instead, 72 neurons that are connected to B fire and everything else stayed the same, then the bit-packed scheme is used only between nodes A and B. In this case, only 63 integers would have to be transferred instead of the 72 with the AER scheme.

A single byte at the beginning of each packet is use to facilitate the dynamic switching between message packing schemes. For the AER scheme this header byte indicates the total number of firings contained in the current message. For the bit-packed scheme this will be a negative value signaling the receiving node to process the packet as such.

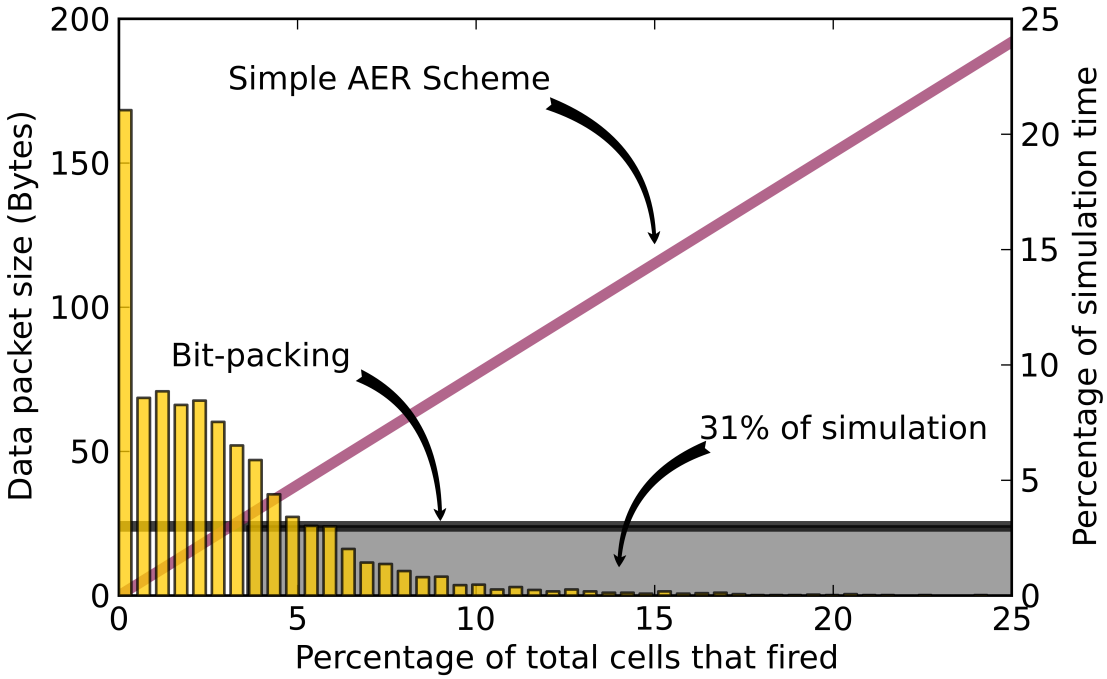



**Figure 4.3:** Example message passing.

Consider the action selection model of Thibeault and Srinivasa (2012), as a motivational example of a physiologically realistic model where the rate-independent message passing scheme would have a significant performance impact. The network models three micro-channels of the basal ganglia using 576 neurons. For this example, it is assumed that the 192 neurons in the Globus Pallidus External (GPe) are output projections whose spiking information must be passed to another node. Physiologically the GPe has a basal level of activity around 30 Hz which is a level where other message passing schemes show performance degradation.

It takes 18 integers to encode all 576 neurons, which is equivalent to encoding 3.125% of the total outputs with AER. Figure 4.3 illustrates the amount of simulation time spent for the different rates of spiking activity over a 5 second simulation. This was only the basal

level of activity and no inputs were given. The results show that 31% of the simulation time is spent in the region were more than 18 neurons fire.

### 4.2.3 Initial characterization

Four possible communication schemes were initially explored using a custom C++ code. Different message sizes, from 8 bytes to 1000 kilobytes, were exchanged 1000 times. The total time to send the messages was measured using the Linux Real-Time Timers. The tests were run for $4$, $8$, $16$, $32$, and $64$ physical machines with each machine containing $2$ processors that are treated as separate nodes by MPI. Simulations were run multiple times and the lowest time for each message size was recorded. This procedure was chosen to eliminate the periodic anomalies, such as scheduler synchronization, package updates or network file system activity.

### 4.2.4 Benchmark experiments

Here we explore two different aspects of spike exchange with MPI on general computing architectures. The first, explained further below, is the type of communication mechanism. The performance between peer-to-peer and collective communication using the included MPI functionality is analyzed to determine if one shows a clear benefit. This was completed for both Infiniband communication fabric and standard Ethernet.

The second aspect explored was both the benefit of bit-packing as well as when in a simulation to switch from AER. The pivot point is a multiplier used to determine when to switch to bit-packing. The number of neurons connected to a remote node that fire during a time step, $F$, is compared to the total number of connections to that node, $N$, divided by

$32$, the number of bits in an integer, times the pivot point, $P$.

$$F \leq P\frac{N}{32}. \tag{4.1}$$

Basically a pivot point of $1$ would correspond to the point in the simulation when the number of cells that fire is equal to the horizontal line of Figure 4.3. A pivot point of $0$ would mean that at every exchange the packets would be encoded with the bit-packing scheme. Currently only the Non-Blocking and Alltoall mechanisms have the option of bit-packing. Since the initial performance for the non-blocking method was higher than Alltoall, it was chosen for the bit-packing experiments. During the experiments pivot points of $0$, $1$, $2$, $3$, $10$, $20$ and $32$ are used.

For each of these, two different types of experiments were performed: strong scaling and weak scaling. The strong scaling experiments explore networks of the same size distributed over a larger number of compute nodes. The experiments are outlines in Table 4.1. In the weak scaling experiments the size of the network increases in direct correlation with the number of nodes, this is outlined in Table 4.2. Only communication time is measured in the benchmarks and each trial was run three times, with the lowest trial time recorded.

**Table 4.1:** Strong scaling experiments.

| Nodes | Cells | Connections |
|---|---|---|
| 4 | 2000000 | 1000 |
| 8 | 2000000 | 1000 |
| 16 | 2000000 | 1000 |
| 32 | 2000000 | 1000 |
| 48 | 2000000 | 1000 |
| 4 | 250000 | 10000 |
| 8 | 250000 | 10000 |
| 16 | 250000 | 10000 |
| 32 | 250000 | 10000 |
| 48 | 250000 | 10000 |

**Table 4.2:** Weak scaling experiments.

| Nodes | Cells | Connections |
|---|---|---|
| 4 | 2000000 | 1000 |
| 8 | 4000000 | 1000 |
| 16 | 8000000 | 1000 |
| 32 | 16000000 | 1000 |
| 48 | 24000000 | 1000 |
| 4 | 250000 | 10000 |
| 8 | 500000 | 10000 |
| 16 | 1000000 | 10000 |
| 32 | 2000000 | 10000 |
| 48 | 3000000 | 10000 |

### 4.2.5 Hardware

The Infiniband fabric is a hardware level communication system specifically designed for high-performance applications. It offers low-latency and high-bandwidth over short distances. In contrast, Ethernet hardware is a ubiquitous technology found on most modern computing architectures. It is primarily used for local-area connections and includes the

physical and data link layers of the Open System Interconnection model. Although in high-performance systems, hardware communication fabrics like Infiniband are more prevalent, evaluating the lower bandwidth and lower latency mechanisms is important for both remote applications (i.e. robotics), as well as inexpensive HPC clusters.

The benchmarks presented here were completed on a cluster of 92 compute nodes, each with two Intel Xeon E5520 2.27GHz CPUs and two NVIDIA Tesla C1060 cards, with Infiniband and Gigabit Ethernet communication backends.

### 4.2.6 Test suite

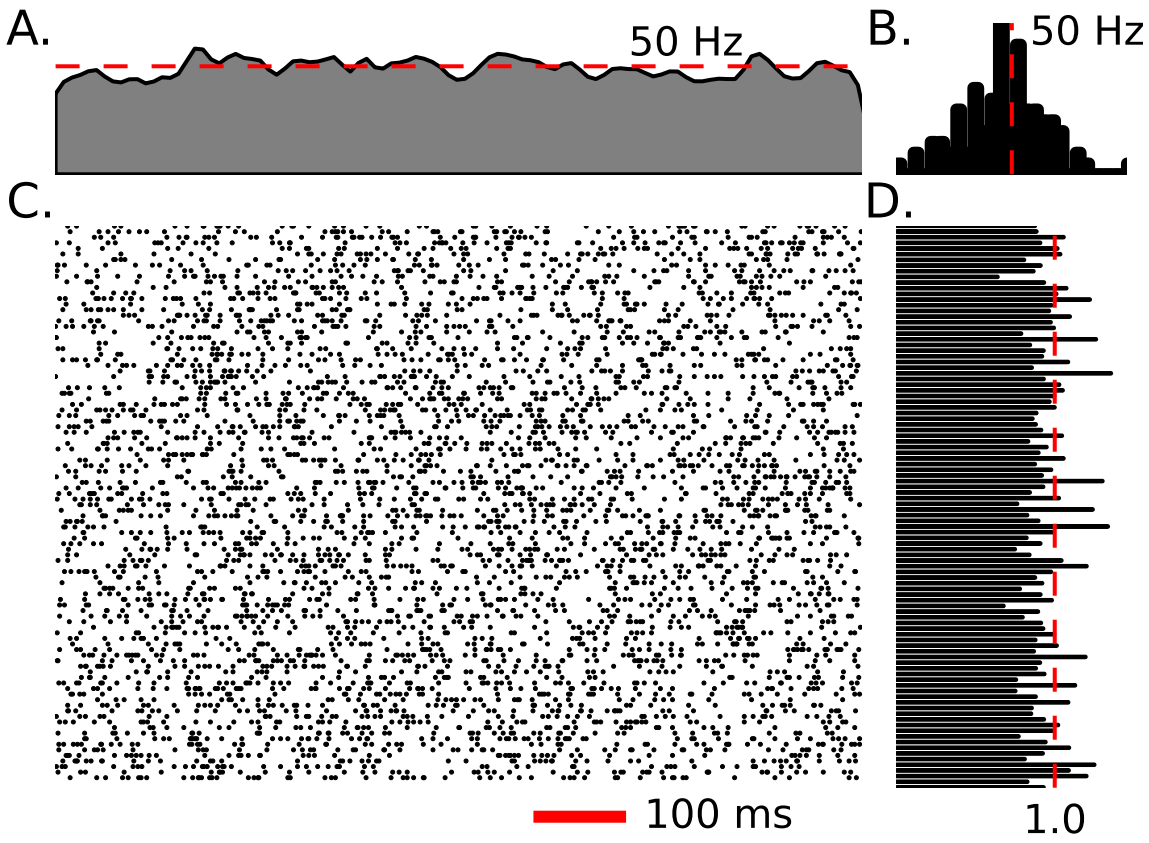



**Figure 4.4:** Example activity of 200 neurons from a $50\ Hz$ Poisson network. A. Fire rate of the network calculated using a Gaussian window. B. Neuron spike frequency histogram. C. Raster plot of spiking activity for 1 second. D. Coefficient of variation for the 200 neurons displayed in C.

A software suite was developed to facilitate the benchmarks. The suite consisted of network generation, neuron spike generation, and spike exchange, written in C++, along with job submission, results analysis and plotting, written in Python. The network is split randomly with each node simulating the same number of neurons. The connections are randomly selected from a uniform distribution. The neural activity is generated by a Poisson random point process with the center at the target frequency. Four target frequencies were used, $10$ Hz, $30$ Hz, $50$ Hz and $80$ Hz. The spike exchange was then controlled by one of three mechanisms, described below. Simulations are run for $1$ second simulation time. The Non-blocking and Alltoall code was written by the author and the blocking and bit-packing code was co-written by the author and Kirill Minkovich while at HRL Laboratories LLC.

### 4.2.7 MPI communication mechanisms

The simulations were performed using MVAPICH 1.7. MVAPICH is built on the MPICH2 libraries by The Network-Based Computing Laboratory at The Ohio State University. It is developed specifically for Infiniband architectures with optimizations that exploit the Infiniband hardware. There are a number of options for messaging passing provided by the standard. Four of those were explored in this project; these are described in detail below.

#### Point-to-point communication

The blocking P2P communication is accomplished with separate calls to *MPI_Isend* and *MPI_Recv*. The *MPI_Isend* command is a non-blocking function that takes in, as arguments, a buffer with the data to send, how much data to send and which node in the cluster to send it to. The receiving node will first call the function *MPI_Probe* to determine how much data the sending node has. It will then use *MPI_Recv* to copy the data to a local buffer. *MPI_Recv* will block until the sending node has completed its corresponding call to *MPI_ISend*.

Similar to the blocking communication, the non-blocking scheme uses calls to *MPI_Isend* but it uses the non-blocking call to *MPI_Irecv*. This method allows the underlying communication to be handled by the MPI threads.

Both point-to-point communication schemes requires a node to call *MPI_Isend* for each node it sends data to, as well as *MPI_Probe* and *MPI_Recv* for each node it is receiving data from. As a consequence of this, even if a node does not have any data to send, a dummy message must be sent to keep the receiving nodes from locking and waiting for data that will never arrive.

**Collective communication**

The MPI standard provides broadcast functions that allow the exchange of data to multiple nodes using a single optimized routine. The motivation behind these functions is to reduce code complexity while allowing developers of the MPI middleware to optimize the communication at the device level. To use the commands, the connection information as well as the send and receive buffers are setup the same as they were for the peer to peer communication above. A single call to the Alltoall function completes the exchange. Two methods employing these functions in neural simulation were developed and tested.

The collective operations use either a fixed buffer size, this is the case with *MPI_Alltoall*, or variable size buffers, used with *MPI_Alltoallv*. To the user, the execution is similar to the P2P methods described above except there is no call to *MPI_Probe*, instead a single byte header is used at the beginning of each message to indicate how much information was sent. Each sending node has a local buffer that is the same size as the receiving node's. If the amount of data the sending node has exceeds the buffer size, a resizing of the buffer is completed. The sending node will then send the maximum amount of data the receiving node can hold. However, the header will indicate the total amount of data the sender actual

needs to transmit. The receiving node then performs a resize and a peer to peer ISend/Recv is completed to transmit the remaining information. Generally, there are very few resizes and the initial buffer size can be made sufficiently large enough to reduce the likelihood of needing them.

Collective communication calls block processing so no other computations can be performed while spikes are exchanged. In order to allow for communication and the spike computations to occur in a parallel, the spike exchanges for collective communication is threaded. There is a cost associated with sending and receiving signals from the communication thread but ideally that is minimal compared to the benefits.

## 4.3 Results

### 4.3.1 Initial characterization

In a neural simulation it is unlikely a single node would pass a message of $1$ MB. Even a $200$ kB message under the AER scheme would mean that $50,000$ neurons on that node had fired. Figure 4.5 illustrates the results of the initial MPI experiments. For most cases the point-to-point mechanism are more efficient than the collective operations. The exceptions are for Alltoall for $64$ and $128$ nodes on Infiniband. This is particularly obvious for $128$ nodes; which is not an unexpected result. The Network-Based Computing Laboratory at The Ohio State University has put considerable effort into fine tuning the Alltoall algorithm in particular (Sur et al., 2005). The results of that effort were lower latencies compared to other methods.

The full characterization results are present in Figure 4.6. The most interesting of these, are the noticeable step jumps in communication time. For earlier Ethernet experiments and

almost all Infiniband experiments there is discontinuity in the timing curves. Why this is present is unclear. Preliminary investigations into the MVAPICH source code revealed switches in methodology as message sizes changed but whether or not that is the cause is unknown at this time.

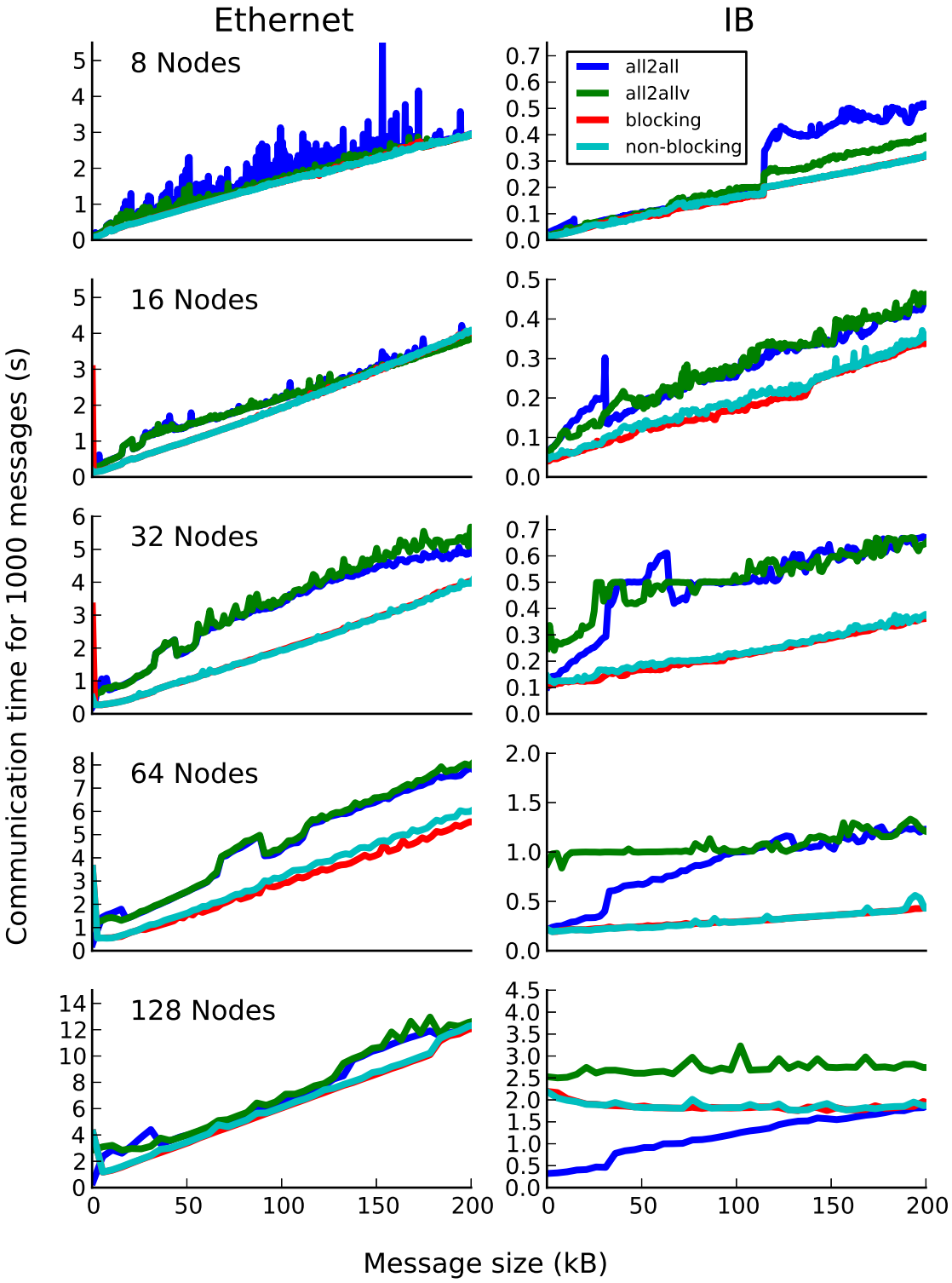



**Figure 4.5:** Communication characterization (small)

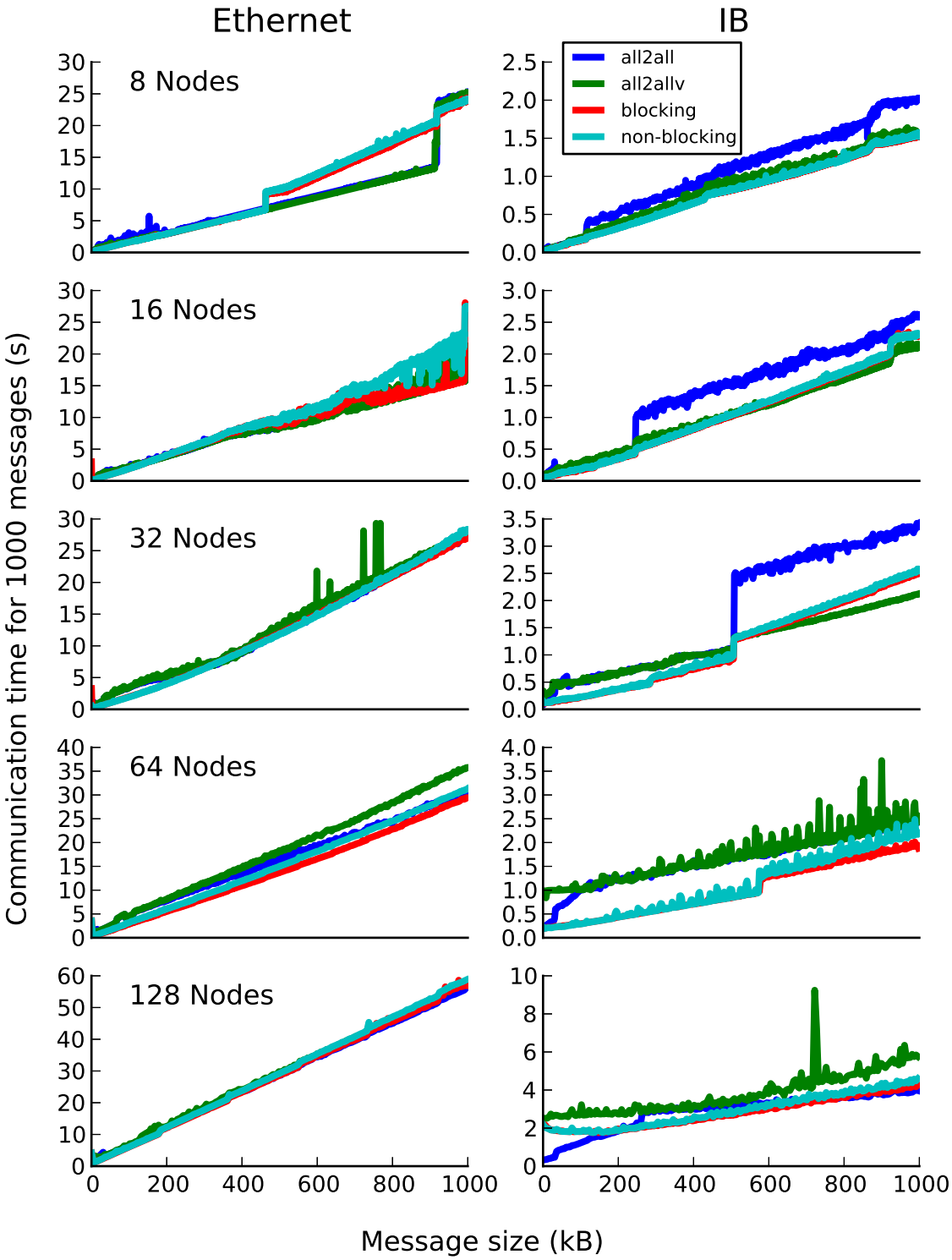



**Figure 4.6:** Communication characterization (large).

### 4.3.2 Communication method

#### Infiniband

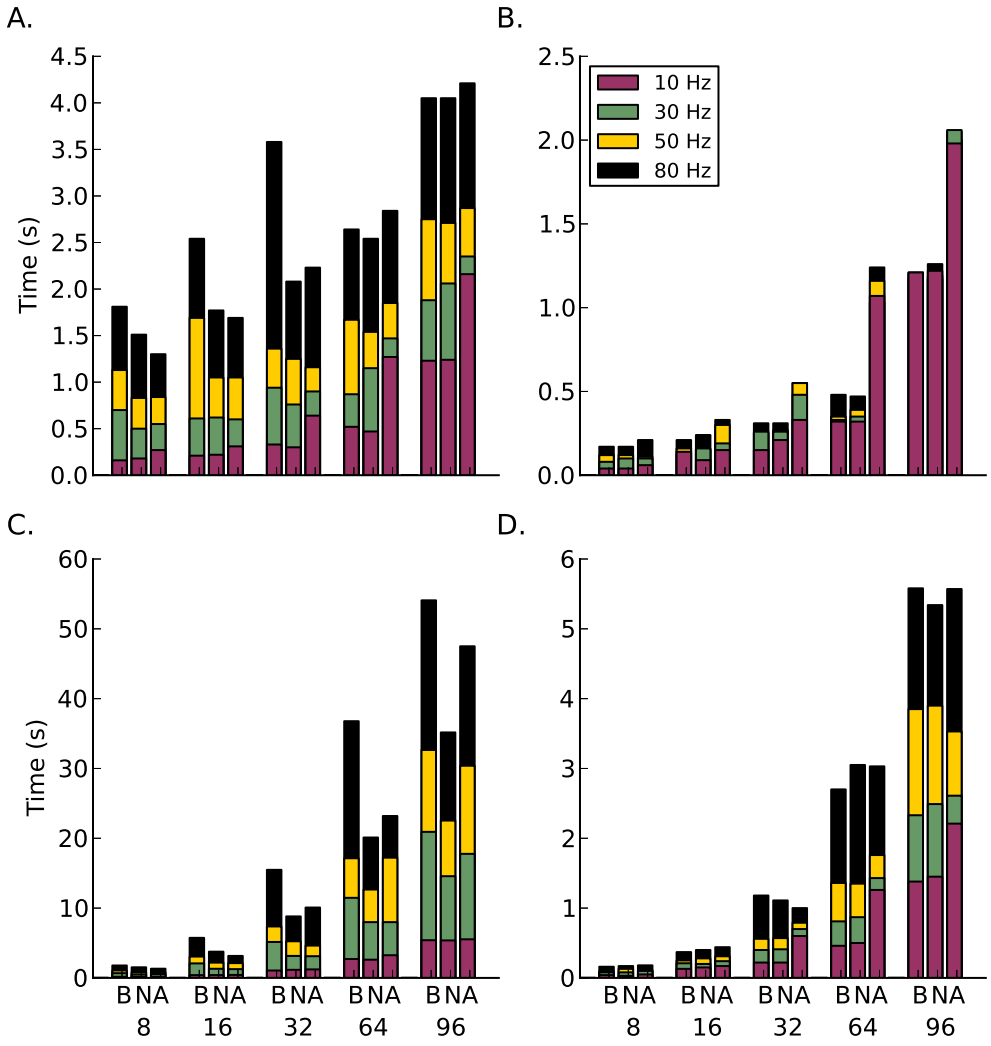



**Figure 4.7:** Results for communication method experiments on Infiniband. Along the x-axis are the different communication methods for the different number of compute nodes. The letters correspond to B-blocking, N-Non-blocking, A-Alltoallv. The subplots are, A. Strong scaling for $1,000$ efferent synapses. B. Strong scaling for $10,000$ efferent synapses. C. Weak scaling for $1,000$ efferent synapses. D. Weak scaling for $10,000$ efferent synapses.

The strong scaling results shown in Figures 4.7A and B reveal an interesting trend in the cost of increasing the distribution of a network. In theory, as a network is distributed over more compute nodes, the performance will increase. This is an effect of reducing the amount of computational work required by each node. However, as illustrated here, the communication cost increases with a corresponding increase in compute nodes. This is an important consideration for balancing the number of neurons per node with the number of nodes. Unless the network activity and the number of compute nodes is low, the simulations would be unable to run in real-time. The real-time mark is an important measurement for neural simulation. Models that can be run in real-time or faster, are required for embodied modeling; something that is important to the work presented in this paper.

Another interesting trend can be seen in the blocking communication results of Figure 4.7A. At first these were assumed to be anomalies. However, after running $12$ extra simulations for $8$, $16$ and $32$ nodes at both $50$ and $80$ Hz firing rates the results stayed consistent. It is still unclear why there is a drop in simulation time at $64$ nodes compared to $8$, $16$ and $32$.

The benefit of dummy neurons is illustrated in Figure 4.7B. Although there is a clear penalty to encoding more spike messages it is not dependent on firing as in Figure 4.7A. This likely due to the smaller number of neurons.

The weak scaling experiments shown in Figures 4.7C and D show how an increase in both neurons and nodes can affect the overall performance. Disappointingly, the correlation between the two is not linear; instead following a more exponential trend instead.

Overall on infiniband hardware the choice of communication method seems to favor the non-blocking method. This is slightly surprising in the context of the results of Section 4.3.1, where the blocking and non-blocking methods appeared to be identical. The non-uniform nature of the Poisson network is the likely explanation for the difference. The

non-blocking code allows message processing to happen out of order, favoring those that are sent earlier. There is obviously less time wasted waiting for messages to arrive in order.

### Ethernet

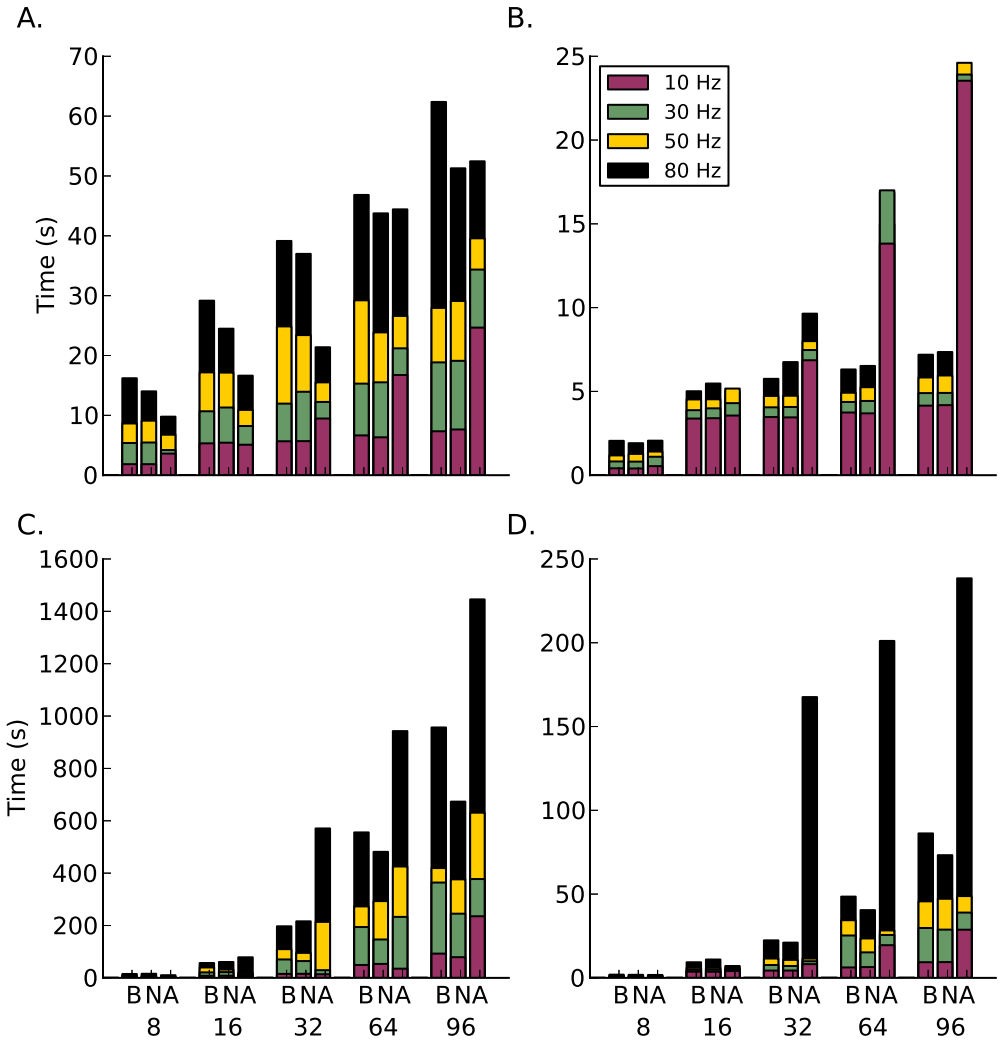



**Figure 4.8:** Results for communication method experiments on Ethernet. Along the x-axis are the different communication methods for the different number of compute nodes. The letters correspond to B-blocking, N-Non-blocking, A-Alltoallv. The subplots are, A. Strong scaling for $1,000$ efferent synapses. B. Strong scaling for $10,000$ efferent synapses. C. Weak scaling for $1,000$ efferent synapses. D. Weak scaling for $10,000$ efferent synapses.

The strong scaling experiments on Ethernet hardware in Figure 4.8A show a similar increase to that seen on Infiniband. However, the Alltoall method shows an advantage for higher rates of activity and lower number of nodes. Why this happens is again unclear, as is the step increase in communication time for $64$ and $96$ nodes. When the number of efferent connections is increased to $10,000$, a surprising plateau in the blocking and non-blocking schemes emerges between $16$ and $96$ nodes. The Alltoall scheme however, continues to trend upwards.

For the weak scaling experiments, Figures 4.8A and B, the Alltoall scheme actually performs considerably worse than the other two methods. With the Non-blocking scheme generally demonstrating better timings throughout the weak scaling experiments.

### 4.3.3 Bit-packing

#### Infiniband

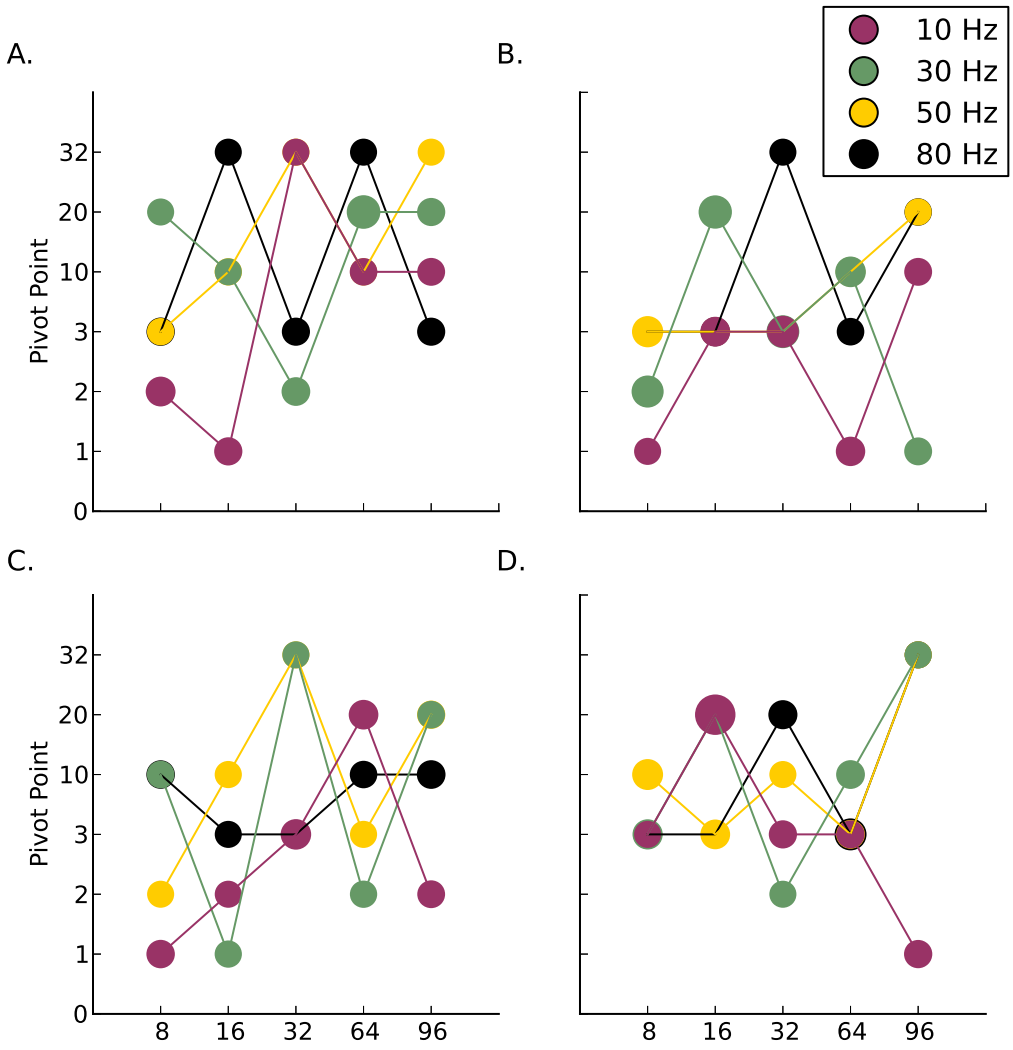



**Figure 4.9:** Pivot results: Infiniband. The number of nodes are listed along the x-axis. The markers indicate the pivot point that resulted in the best performance. The size of the markers are proportional to the performance benefit compared not bit-packing (Pivot = 32). This is in reference to the marker size in the legend. The subplots are A. Strong scaling for $1,000$ efferent synapses. B. Strong scaling for $10,000$ efferent synapses. C. Weak scaling for $1,000$ efferent synapses. D. Weak scaling for $10,000$ efferent synapses.

The bit-packing experiments on Infiniband hardware show no clear trend towards an optimal pivot point, Figure 4.9. On these plots the size of the markers is directly proportional to the increase in performance when compared to using the AER scheme exclusively. Although there is a benefit to bit-packing, that advantage is surprisingly small. This may be due to the the low-latency, high-bandwidth characteristics of the Infiniband fabric.

**Ethernet**

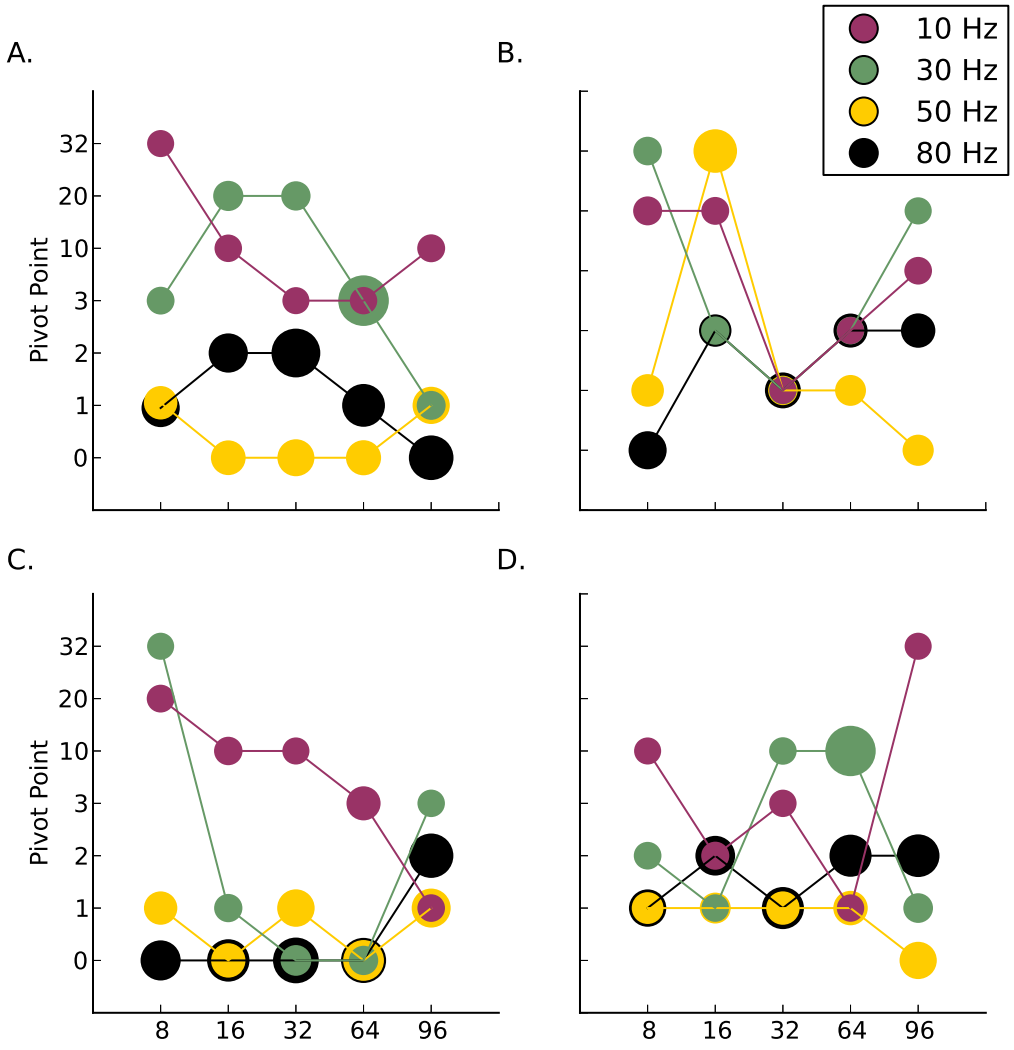



**Figure 4.10:** Pivot results: Ethernet. The number of nodes are listed along the x-axis. The markers indicate the pivot point that resulted in the best performance. The size of the markers are proportional to the performance benefit compared not bit-packing (Pivot = 32). This is in reference to the marker size in the legend. The subplots are A. Strong scaling for $1,000$ efferent synapses. B. Strong scaling for $10,000$ efferent synapses. C. Weak scaling for $1,000$ efferent synapses. D. Weak scaling for $10,000$ efferent synapses.

On the lower performance Ethernet hardware bit-packing offers important performance gains. The strong scaling results displayed in Figure 4.10A, show that as both the rate

and number of nodes increases so does the benefit of employing bit-packing. This trend is slightly less obvious for the case of $10,000$ efferents, Figure 4.10B but returns for the weak scaling experiments, Figures 4.10C and D.

### 4.3.4 Bit-packing vs. AER

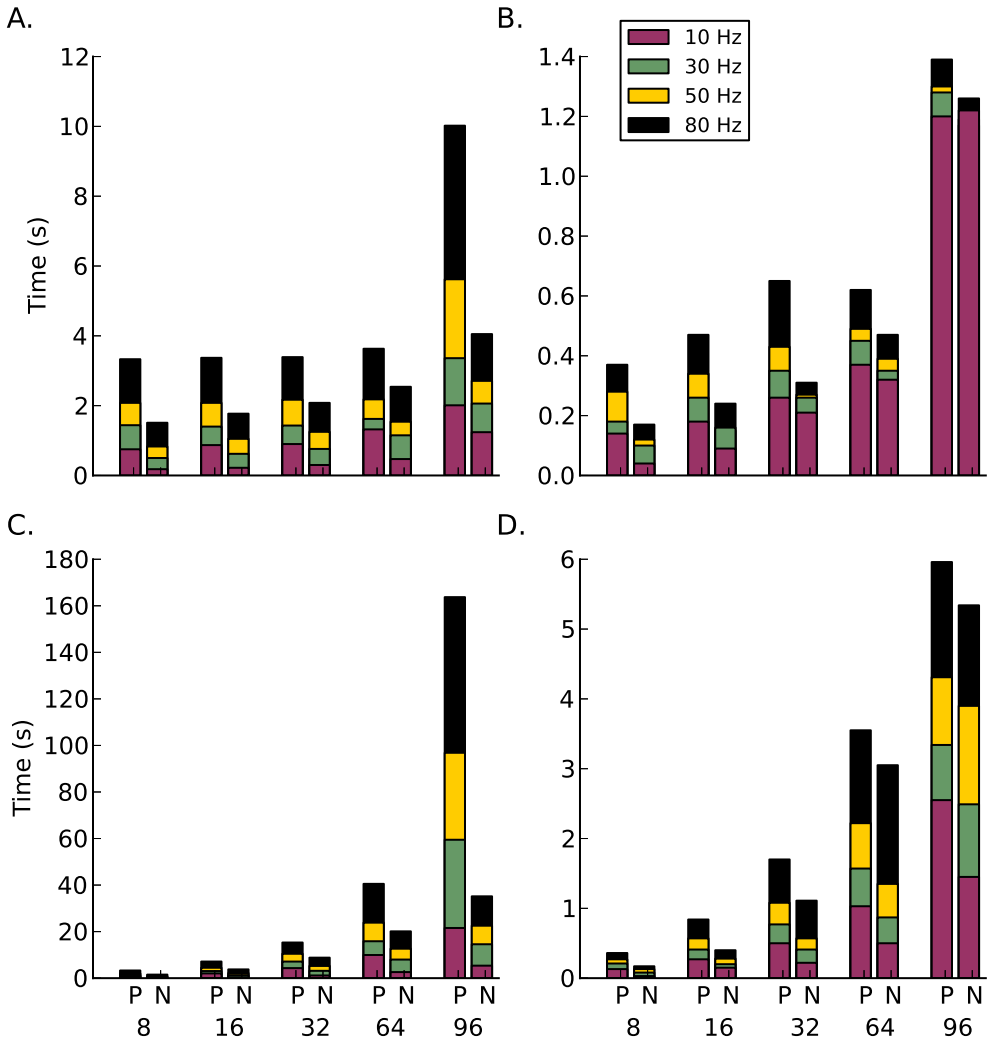



**Figure 4.11:** Bit-packing vs. AER: IB. Along the x-axis are the different two different pivot points for the different number of compute nodes. The letters correspond to P-bit-packing, N-No packing. The subplots are, A. Strong scaling for $1,000$ efferent synapses. B. Strong scaling for $10,000$ efferent synapses. C. Weak scaling for $1,000$ efferent synapses. D. Weak scaling for $10,000$ efferent synapses.

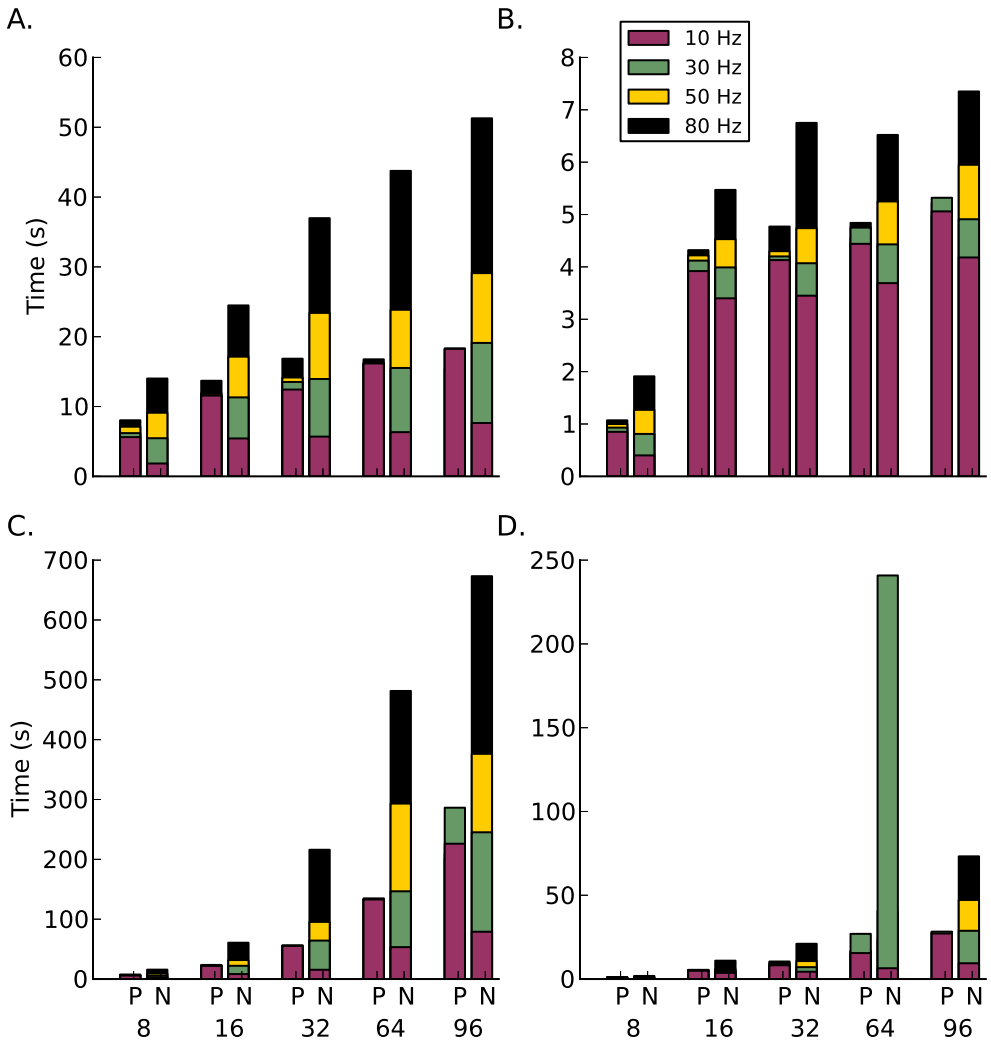



**Figure 4.12:** Bit-packing vs. AER: Ethernet. Along the x-axis are the different two different pivot points for the different number of compute nodes. The letters correspond to P-bit-packing, N-No packing. The subplots are, A. Strong scaling for $1,000$ efferent synapses. B. Strong scaling for $10,000$ efferent synapses. C. Weak scaling for $1,000$ efferent synapses. D. Weak scaling for $10,000$ efferent synapses.

In Figures 4.11 and 4.12 we compare the experiments for bit-packing during every spike exchange with those always using the AER scheme. In the Infiniband case, Figure 4.11, using a scheme that packs every spike transfer is less efficient than always using an AER scheme. However, the opposite result is found for the Ethernet case.

## 4.4 Discussion

Past research in spike exchange methods has been noticeably sparse. Many groups present a single communication mechanism with minimal justification for its selection. The majority of general simulation environments use the point-to-point blocking mechanisms in MPI (Wilson et al., 2001; Morrison et al., 2005; Pecevski et al., 2009). Morrison et al. (2005) combined that with the Complete Pairwise EXchange (CPEX) algorithm. At the time this was selected based on the assumption that it was more robust. However it was later stated that the collective, *MPI_allgather*, was more efficient on certain hardware (Eppler et al., 2007); benchmarks were not presented to support that claim. PCSIM also uses blocking communication with the CPEX algorithm (Pecevski et al., 2009).

The NEURON simulation environment is one of the few that use the collective *MPI_Allgather* as opposed to the point-to-point methods (Migliore et al., 2006). This decision is based on the simplicity of the implementation and that the performance of NEURON is dominated by the more complex models that are its niche. However, the use of NEURON on the BG/P Supercomputer was the motivating factor for work presented in Hines et al. (2011). This is the most current analysis of different spike-exchange methods but is unfortunately specific to the BG/P hardware. The work presented here is the first analysis aimed at the COTS hardware more readily available to the computational neuroscience community.

### 4.4.1 Choosing a communication mechanism

Selecting a spike exchange method is still a difficult problem. The type of hardware as well as the configuration can create situations where one method clearly outperforms. The results of this work suggest that a safe pick for COTS architectures would be the non-blocking

point-to-point communication methods. This is contradictory to the results found for Infiniband backend in Eppler et al. (2007), and the BG/P hardware in Hines et al. (2011). This analysis will need to be repeated as new hardware as well as more optimized communication methods are released. Ideally, we will package and release this work to provide an automated mechanism for selecting the highest performing communication method for a given hardware setup.

**Why not Alltoall?**

The results of the initial characterization of Section 4.5 suggest that for truly large-scale simulations, using the basic Alltoall mechanism would offer much higher performance. In the MVAPICH2 implementation this has been optimized for use with Infiniband hardware (Sur et al., 2005). The motivation behind the function is to reduce code complexity while allowing developers of the MPI middleware to optimize the functions at the device level.

The Alltoall collective requires that all nodes send the exact same amount of information to each node in the simulation. So message packets must be fixed size and overflow of those must be handled by a separate mechanism. For networks with high-activity and large size, at least large enough to justify $128$ nodes, this may prove beneficial. An Alltoall implementation was completed during development cycle of this project. When included in the HRLSim code the results for all test models was so much worse than the other methods that the code base was later abandoned. These results suggest that there may in fact be an important place for the method in the simulator. In the future we plan to update the original code base and rerun the benchmarks completed here to see if there is in fact a niche for the Alltoall method.

Another possible benefit of this method that was not tested, is the reduced overhead in creating the spiking messages. In the methods described above each node keeps a local

buffer that is filled with spiking information. These are contiguous in memory and require some form of locking to prevent multiple compute threads writing to the same location. With the fixed message size scheme, each thread can be assigned a section of the buffer. It can then be guaranteed that no other thread will be writing to that buffer. This would allow the removal of thread-blocking which may provide another point of optimization.

### 4.4.2 Hybrid message passing

With software simulation environments there is generally a computational cost associated with packing the spike message. In most cases it is insignificant or can be reduced by using GPGPU's which are designed for just such parallel tasks. The hybrid spike passing scheme has already proven effective in large-scale cluster based neural simulations by HRL (Minkovich et al., 2012).

In addition to large-scale simulations, this technique can also improve the performance of communication between neuromorphic architectures. These have traditionally used AER schemes. Take for example SpiNNaker (Khan et al., 2008), which was designed to use an AER communication to simulate a neural network with a firing rate of 10Hz. The final hardware was theoretically able to simulate networks firing up to $77.5$ Hz (Navaridas et al., 2009). Once the network is firing above this rate either spikes would have to be dropped or the whole system slowed down. The problem is that in biological system, even though the firing rate is on average $10$ Hz, there are times and regions, when the firing rate goes beyond $100$ Hz. Using the hybrid encoding scheme presented in this work, would allow for scaling to any firing rate.

### 4.4.3 Model complexity

The conclusions of this work are based on the idea that the neuron and synaptic computations are completed relatively quickly. Additionally, it is assumed that a single node can process a large number of neurons. This is the case for most point neuron implementations but as the complexity of the neuron model increases the time spent in numerical integration correspondingly increases. The cost associated with spike exchange is then less of a performance bottleneck and the benefit of optimization becomes negligible.

# Chapter 5

# Embedding Neural Models in Virtual Environments

## 5.1 Introduction

The concept of neurorobotics is dependent on the ability to immerse an embodied agent into an real or virtual world. Although it has been argued that using a physical environment is vital to creating intelligent systems (Krichmar, 2008), this is often impractical. Furthermore, this can create unnecessary complications during the development of novel neural theories. Employing a Virtual Environment (VE) however, presents its own unique issues. One that is often a hindrance to rapid model development is performance. The emphasis on creating detailed but slowly executing visual environments runs counter to the high-performance neural models of software simulators and more recently of high-performance neuromorphic processors.

With traditional game and VE software development the focus is on graphical rendering of the environment. Developing new worlds or elements requires creating a visual repre-

sentation of it; introducing both development and performance bottlenecks. This work is aimed at reducing these bottlenecks. By removing the restriction of creating and rendering the visual representation, researchers can focus on how the environment will influence the entity and its sensing systems. This can reduce development time while increasing performance. In addition, a system that can interact with both hardware and software models, without concerns for visualization, are extremely useful for the testing and development of neuromorphic models. The motivations for this development are:

- A system that can interact at fast and slow speeds without concerns for visualization are extremely useful for testing and development of neuromorphic models.
- The SyNAPSE hardware runs at different speeds, both faster and slower than real-time.
- Designers can focus on the important interactions rather than the minutia of creating compelling visible environments.
- Results can be plotted roughly during testing and later rendered in more detail for publishing and presentation.

The development of a virtual environment is similar to creating a level in a video game. Because of this similarity we decided to search for a design pattern from video game development. There are a remarkably large number of game engine design patterns available and there is no apparent consensus on where each one is appropriate. This made selecting a pattern to apply to this project a surprisingly difficult task. Here we present an simple virtual environment, a classic Pong style game. Its implementation using object-oriented design is presented first. This works well for basic tasks but for larger environments the complexity of the element interactions makes it undesirable. The lack of scalability and code reuse was the motivation for the entity-system design pattern that is then presented. Finally, we demonstrate how the Pong environment would be represented using the entity-system design paradigm.

## 5.2 Playing games

### 5.2.1 Object-oriented design

Object-oriented design is based around the concept of abstracting aspects of a software system into logical and reusable components. These components, or objects, can be dynamically created, or instantiated, in the software system to store data, perform computations and interact with other system components. As an example consider a code that would represent a wheel. It would contain all of the data (i.e. radius, width, or weight) and computations (i.e. torque, friction or distance traveled) that is required to represent that wheel. To create a car object we would now instantiate four separate instances of a wheel but, in the program all would use the same code that describes the wheel object. This allows larger software systems to be constructed of smaller objects that can be tested separately and reused throughout the project.

An additional layer of complexity that this introduces is the concept of inheritance. This allows class objects to extend an existing object by inheriting its data and function interface. This is a complex concept and is not completely covered here, but as an example consider a class that describes a shape. Imagine that the system using this class wants to get the shape's area; a calculation that is obviously dependent on the type.

One way to handle this would be to instantiate different objects, circles squares and triangles. The problem, is that the main system has to keep track of all of these objects and store them separately. If the designers later want to add a hexagon all of these storage containers must be redesigned, coded and tested.

Inheritance provides a way for the main system to simply store objects that are of type shape. The shape base class provides a function for calculating the area of a shape, we will call it *getArea*. The different types of shape classes will then inherit from the base class.

That inheritance creates a contract with the rest of the system promising that this class will provide an implementation of the *getArea* method specific to that shape. The main program can then create one storage container for shape objects and different shape classes can be be instantiated and passed to that container. When the area calculations are required the main project simply call the shape object's *getArea* method and the appropriate calculation is performed by the class.

When employed appropriately this pattern works well for many applications. To illustrate how to apply this to a virtual world we present an example of interfacing neural models to a simple Pong game. In this, a puck is given basic physics where it will move linearly through the game board and bounce off of the left, right and top walls. The player controls a paddle at the bottom of the board and must use that to reflect the puck and keep it on the board. If the puck gets past the paddle the player loses a life. Keep in mind that we are only creating the game elements, controller and interface to the neural model. The visualization is done after the simulations have been completed by a separate program that simply renders the script output from this environment.

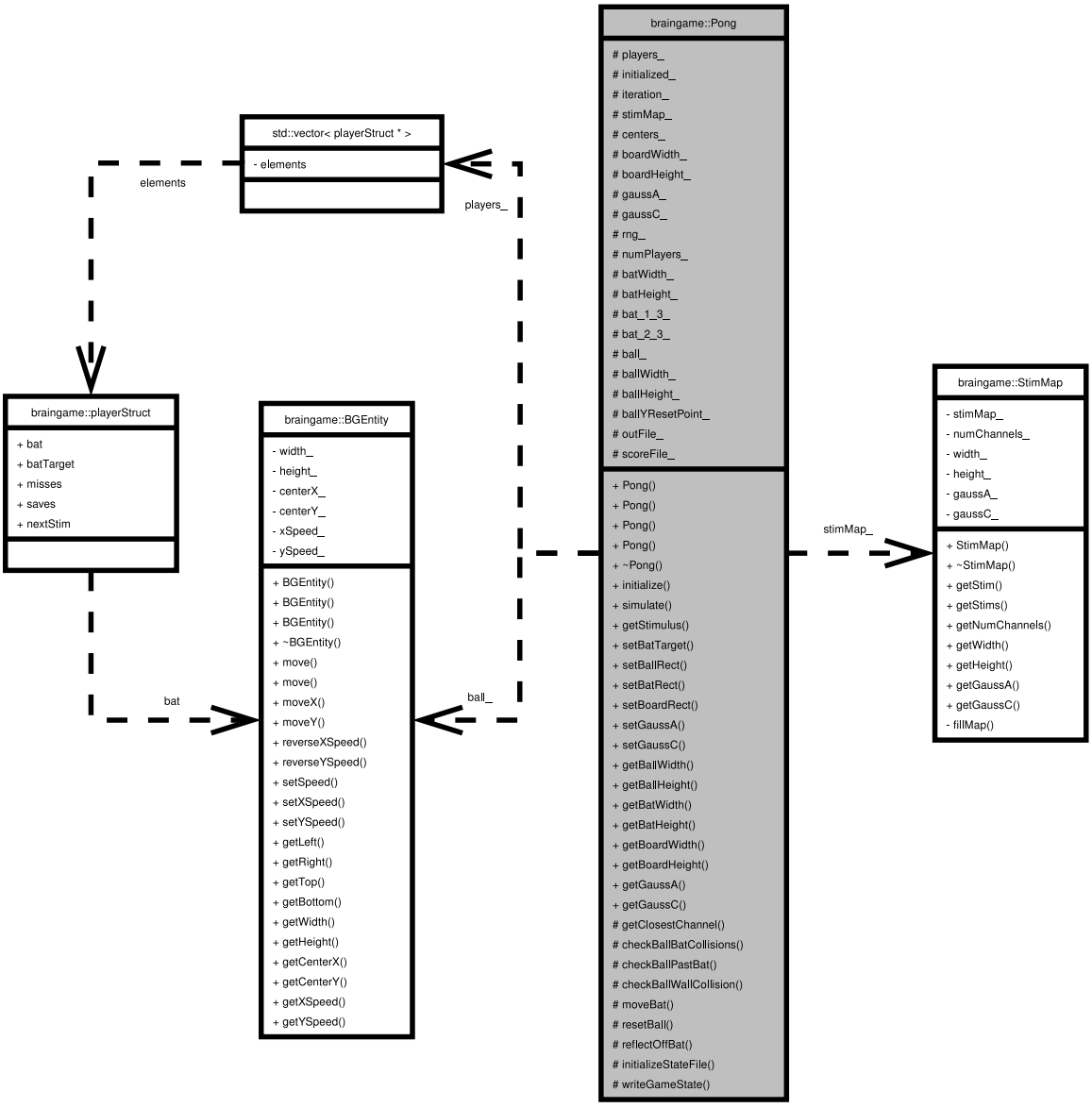



**Figure 5.1:** Pong class layout.

The objects for this system are relatively simple and the class diagram outlining each of the properties and functions is presented in Figure 5.1. The BGEntity class is used to represent the different elements that make up the game. In this case, that is the player's paddle and the puck. The StimMap class is used to encode the game space into neural stimulus based on the position of the puck. Finally, the controller for the game is implemented in

the Pong class.

This is a relatively simple environment with a small number of interacting elements. However the Pong class is responsible for much of the heavy lifting, such as collision detection, input and output processing, and game physics. For a small code base this is reasonable, but as the complexity of the task increases this design pattern will quickly become unmanageable. Details of the implementation are provided in Chapter 8.

### 5.2.2 Braingames

To support a reduction in VE development time, we propose an implementation of the Entity System Paradigm (ESP) (West, 2007). Unlike hierarchical design patterns, most notably the object-orient design pattern presented above, ESP provides better code-reuse and extensibility that is required in research and development. Traditionally, ESP has been used exclusively in video game development, however, it is an ideal solution for a generic VE engine. The features of this design are:

- A headless framework for creating high-performance virtual-environments capable of interfacing with neural simulations and neuromorphic hardware.
- Written in C++ with comprehensive unit test suite.
- Generic reusable components for simple environment generation.

### 5.2.3 Entity System Design

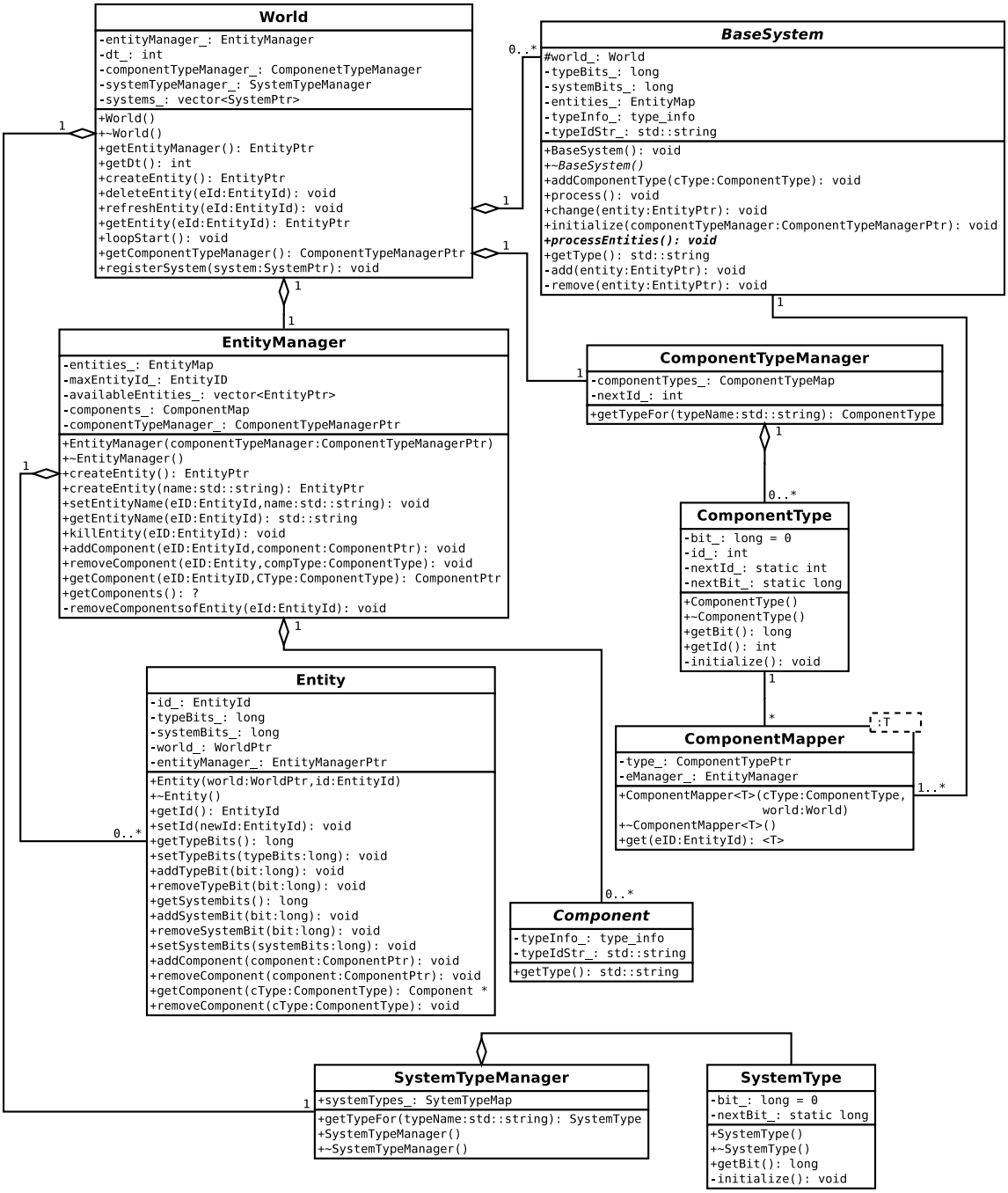



**Figure 5.2:** BrainGames Framework.

Component based design offers a way to decompose the different functional domains of the VE entities into their constituent components. In addition, it provides a means for layered abstraction without the negative impacts on performance and extensibility that many hierarchical object-oriented designs impose.

Entities represent groups of components; here every VE element is an entity. The components themselves do not contain any logic. Instead, a data-driven approach is taken and the components are nothing more than collections of data with exposed getter and setter functions. The control logic is implemented by the systems. Systems encapsulate the update functions for each of the components. The systems are responsible for modifying the data contained within the components. The different systems contain references to the components they are interested in. In fact, the systems never have a need to reference the entity object itself. This allows the addition and subtraction of systems in a clean and unobtrusive way. The removal of a component from an entity does not result in a broken hierarchy. Instead the systems previously using the components of that entity simply remove the references to it. The addition or modification of systems works exactly the same way; essentially creating a run-time plug-in interface.

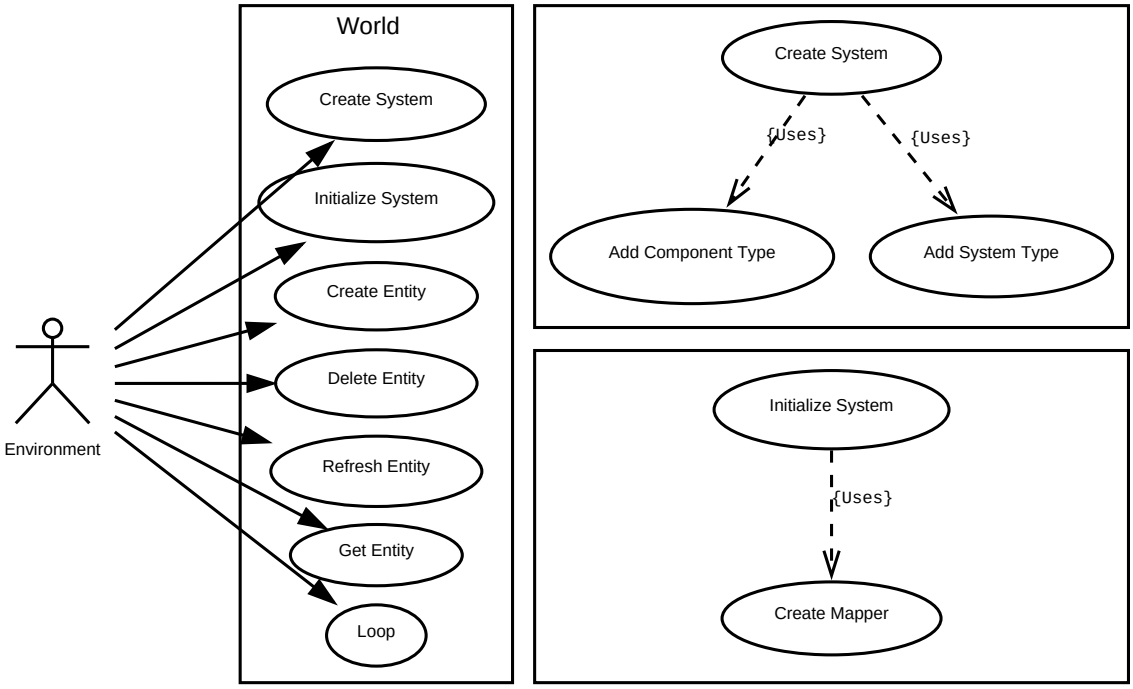



**Figure 5.3:** Basic virtual environment use cases with the world class of BrainGames.

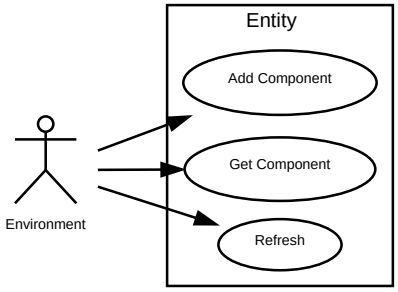



**Figure 5.4:** Basic virtual environment use cases with the Entity objects of BrainGames.

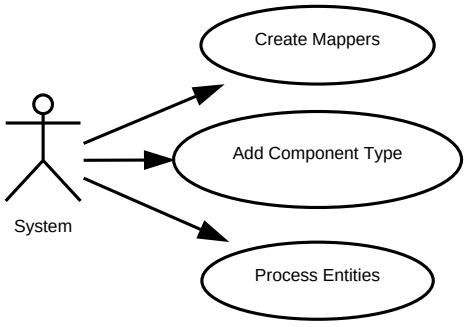



**Figure 5.5:** Basic system use cases.

The design for a C++ realization of this idea is presented in Figure 5.2. Use case diagrams for this concept are presented in Figures 5.3, 5.4 and 5.5. The use cases themselves are provided in Appendix A. This represents a minimal architectural design based on the ideas presented in Bilas (2007) and the framework of Arent and Costa (2012). Unlike game engines, this implementation focuses on the specific needs of the neurorobotics community. Mainly, many parallel sensory and motor loops that are the hallmark of most neurally inspired designs.

### Pong using ESP

Once an ESP framework is in place, creating the components for the Pong game becomes relatively simple. Consider this from the point of view of the main code. The world object is instantiated first along with the system objects, Listing 5.1.

```
1 // Create the world.
  world = new World();
3 // Create the systems.
  world.registerSystem( new MovementSystem() );
5 world.registerSystem( new CollisiontSystem() );
  world.registerSystem( new StimOutputSystem() );
7 world.registerSystem( new InputControlSystem() );
  world.registerSystem( new RecordingSystem() );
```

**Listing 5.1:** Instantiate the world and the required systems.

The Puck and the Paddle are then the only entities added to the world, Listing 5.2.

```
// Create the puck.
Entity puck = world.createEntity();
puck.addComponent( new Position() );
puck.addComponent( new Velocity() );
puck.addComponent( new Collidable() );
puck.addComponent( new Stimulus() );
// Create the paddle.
Entity paddle = world.createEntity();
paddle.addComponent( new Position() );
paddle.addComponent( new Velocity() );
paddle.addComponent( new Controllable() );
```

**Listing 5.2:** Create the Pong entities and add the components to them.

The environment then loops until an end condition is released. The implementation for each of the system objects defines the logic for the world. The systems loop through the entity references that contain all of the required components.

The motion of each of the entities is handled by the movement system. It uses the position and velocity components to determine the change in position. The collision system is then responsible for all objects in the game that move around and can collide with other game elements. In this case that is only the puck. The input system is responsible for gathering the input information from the neural model, processing it and updating the control system for the paddle. This could be further separated into separate input and control but would require a component to maintain the input information. The stim output system is used to create the neural stimulus based on the position of the puck. Finally, the recording system will collect the current state of the environment and save it for rendering off-line.

In the transition to the BrainGame architecture, the code presented in Figure 5.1 is moved into the separate system classes. In the ESP however, that code becomes cleaner and is logically abstracted, rather than contained in a single monolithic class. In addition, switching out systems to perform different tasks or adding players and entities is not as simple as adding a new entity.

## 5.3 Discussion

### 5.3.1 Similar Work

CASTLE (SET Corporation, 2012) provides a generic virtual environment that offers capabilities similar to those presented here. However, it lacks mechanisms for running simulations faster than real-time or without rendering the graphics to the screen. These restrict the performance of the neural models and their use in distributed computing. The latter is important not only for parameter searches but for stability analysis of deployable systems. Entities and environments are created in Blender with support for other 3D modeling software planned in the future.

Similarly, Webots (Michel, 2004) offers a comprehensive virtual environment along with a number of different modes, including a headless mode. However, Webots was not developed specifically for neurorobotics, and adaptation for neural applications requires custom software modules. In addition, the software requires a physical license for each instance; making distributed execution unreasonable.

Unlike the environments presented above, this design can be compiled directly into the simulation environment or it can be instantiated remotely. Additionally, the inherent support for parallel and distributed hardware further separates it from existing generic virtual environments.

### 5.3.2 Benefits of ESP

A difficult problem to address in the design of these types of systems is the communication between components. One option for addressing this would be to give each entity a reference to those it needs to send messages to. The problem is this creates a tight coupling

between components and changing any of those becomes extremely difficult. Another approach has been to create a message passing system. For larger systems this can become complex and again the coupling of components can be an issue. The advantage to ESP is that the systems take care of all communication between objects. They can easily access different classes of components associated with an entity. The system does not need to know everything about the entity, only that it contains the components it requires to do its update. Similarly, a component does not need to know anything about the other components in the entity, or anything about the entity itself for that matter.

The logic for the environment is contained entirely in the system objects. If designed correctly, those systems will essentially be autonomous units. Changing one will not affect the others. In addition, the components essentially do not change. Combined, these provide a level of code reuse that is important in the rapid development of neurorobotics.

Development and Testing also becomes more straightforward in ESP systems. In particular, multi-developer projects become more tractable, as developers can work on independent system objects. The common components are the only aspect that need to be enforced.

### 5.3.3 Real vs. virtual worlds

In Krichmar and Edelman (2005) it is argued that simulated environments introduce unwanted biases to the model. These are a product of the artificial nature of the input stimulus. In addition, they contend that simulated environments cannot compete with the noisy stimulus of the real world and cannot support many important emergent properties.

The chaotic nature of the real world is no doubt difficult to simulate however there are benefits to constraining initial model development to virtual worlds. The first is the time intensive nature of dealing with physical robots. Development of novel neural models

becomes extremely time consuming as repeating experiments must be completed in real time. Using a high-performance environment such as the one proposed here, models can be rapidly developed and tested. The tests can be automated and the results can be validated over multiple simulations in a realistic amount of time. The complexity of the worlds can also be increased providing a process for building up and exploring the unique properties of a given model.

Virtual worlds also provide a sense of control to the experiments. Different networks can be immersed in a common environment, allowing for a quantifiable comparison between them. As models are developed and refined, they can then benefit from the type of embodying championed by Krichmar and Edelman (2005). The virtual environments then become a compliment to their real-world counterparts.

# Chapter 6

# Analyzing Large-Scale Spiking Models

The additional performance provided by high-performance neural environments allows for very large neural simulations. However, this increase in model size includes a corresponding increase in data size. In order to not only process the increased amount of data in a reasonable amount of time, but also provide a rich and user friendly interface, a new set of analysis tools is presented here. These were developed to support the modeling efforts of both the projects presented in this paper and the SyNAPSE project as a whole.

## 6.1 Introduction

The HRLAnalysis package is an implementation of off-line visualization of spiking and network data for use with HRLSim. The emphasis is on processing the information of large-scale models as efficiently as possible while providing a rich feature set to users. To balance efficiency with usability, the extraction and analysis of the simulation data is performed with C++ and the plotting and further manipulation is handled in Python. The C++ code is interfaced directly with Python through the Boost package and can be called

as a normal python library, so from the users perspective there is no distinction between the packages. The analysis capabilities include:

**Individual cells**

- Individual cell spike count averages over the given time interval.
- Binned counts of the number of neurons within ranges of spike firing.
- Coefficient of variation for cells in the given interval.

**Populations**

- Extraction of spike times and cell indexes within the given interval.
- Average spike rates for the population. Currently, rectangular and Gaussian window functions are supported.

**Network analysis**

- Visualization of the network at the population level in the Graphviz DOT format.
- Average synaptic weights between populations.

## 6.2 Design and use

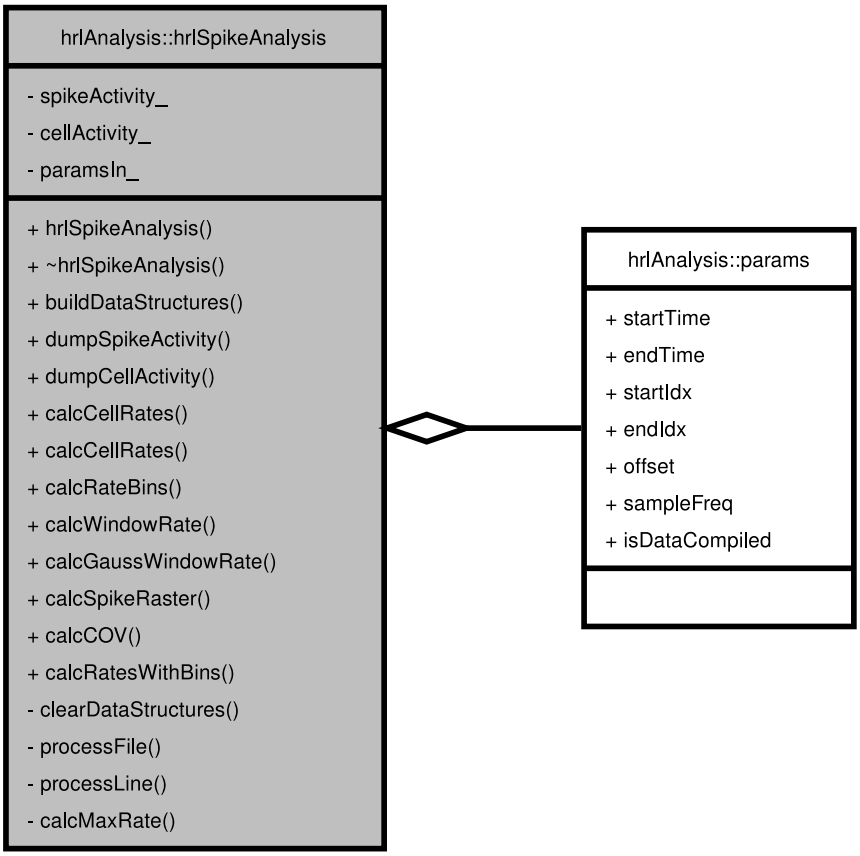



**Figure 6.1:** Spiking analysis class library.

Figure 6.1 illustrates the main spike analysis class. The spiking information is read in and processed by the library. The exposed methods are accessed by Python and the resulting information is passed around using STL vectors wrapped as Python lists. The network analysis library is handled in a similar way. Figure 6.2 outlines the methods exposed by the library.

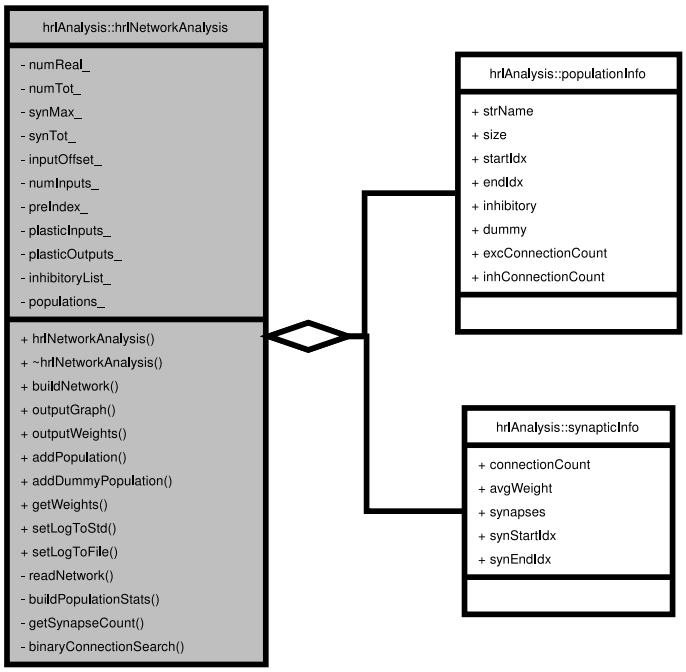



**Figure 6.2:** Network analysis class library.

### 6.2.1 Use within an experiment

The utility of HRLAnalysis lies in its integration with the HRLSim user experiments. When enabled, a configuration file is output based on the setup within the experiment. A call to the *Analyze* function with the experiment instructs it to output information about a particular population. For example:

```
1 // Create the population.
  E = build_net.NewPopulation(1400,NeuronKind().SetIzhikevich(a,b,c,d));
3 // Tell the simulator to create the analysis code for this population.
  E->Analyze(std::string("CA1"));
```

Where “CA1” is the name used for that population. When the simulation is executed a python file will be placed in the Data directory. For this simple example, that Python file will be:

```
#
# This file was automatically generated by the hrlAnalysis Software.
#

def getCellGroups():
    cellGroup = []
    cellGroup.append({"name":"CA1","startIdx":0,"endIdx":1399})
    cellGroup.append({"name":"CA2","startIdx":1400,"endIdx":2799})
    return cellGroup

if __name__ == "__main__":
    print getCellGroups()
```

This file can then be used by the provided analysis code or in your own analysis code, to inform the HRLAnalysis package about the populations of interest. The purpose of using an external file is to avoid the need to recalculate the location within the network of the populations of interest whenever a network is modified.

Users are also given the option of grouping together populations that are defined consecutively. This will create a single analysis group with cell indexes spanning all of the given model populations. At this time nonconsecutive populations are not supported. To use this feature you would give the same name in the calls to Analyze. For example

```
// Create the populations.
StriatumD1 = build_net.NewPopulation(
             NUM, NeuronKind().SetInhibitory().SetIzhikevich(a,b,c,d));
StriatumD2 = build_net.NewPopulation(
             NUM, NeuronKind().SetInhibitory().SetIzhikevich(a,b,c,d));
SNr = build_net.NewPopulation(
             NUM, NeuronKind().SetInhibitory().SetIzhikevich(a,b,c,d));
STN = build_net.NewPopulation(
             NUM, NeuronKind().SetIzhikevich(a,b,c,d));
GPe = build_net.NewPopulation(
             NUM, NeuronKind().SetInhibitory().SetIzhikevich(a,b,c,d));
// Tell the simulator to create the analysis code for the entire group
StriatumD1->Analyze(std::string("Channel"));
StriatumD2->Analyze(std::string("Channel"));
SNr->Analyze(std::string("Channel"));
STN->Analyze(std::string("Channel"));
GPe->Analyze(std::string("Channel"));
```

The resulting configuration file would look like:

```
#
# This file was automatically generated by the hrlAnalysis Software.
#

def getCellGroups():
 cellGroup = []
 cellGroup.append({"name":"Channel","startIdx":320,"endIdx":639})
 return cellGroup

if __name__ == "__main__":
 print getCellGroups()
```

### 6.2.2 HRLAnalysis

Once the simulation has completed and the configuration has been generated the analysis can be completed off-line using the python interface. Note that the use of the configuration file and the support scripts provided here is completely optional. The user only needs to tell python where the HRLAnalysis libraries are located and where the simulation output files are. At that point how the analysis is completed and what is done with it are up to the user. Provided below is an example of how the provided scripts use the configuration files and plot the results of the analysis.

#### Setup

The path and library can be imported using

```
# Add the library include path
sys.path.append(options.includePath)
# import the analysis library
import libHRLAnalysis
```

Similarly, using the generated configuration file Python needs to know where that is located

and dynamically import it. This can be done using:

```
# Import the configuration module.
(dir,fileName) = os.path.split(options.configFileName)
sys.path.append(dir)
try:
    exec('from %s import *' % fileName)
except:
    print "Error: Cannot find the requested configuration file: ",
        fileName
    raise
```

The configuration file provides one function that will return a dictionary containing information on all of the requested cell populations. Once the file is imported it can be called using:

```
# Get the cell groups and the parameters
cellGroups = getCellGroups()
```

The list of binary files can be extracted directly from the given directory using the following filter and regular expression code:

```
# Search for binary files in the search path.
binFiles = os.listdir(options.searchPath)
filterTest = re.compile("^spikes",re.IGNORECASE)
binFiles = filter(filterTest.search,binFiles)

for i in xrange(len(binFiles)):
    binFiles[i] = os.path.join(options.searchPath,binFiles[i])

# Sort the list of files.
binFiles.sort()
```

Finally, the analysis can be done each of the populations provided by the configuration file.

```
for cells in cellGroups:
    analyzeData(cells,options,binFiles)
```

**Running the spike analysis**

After collecting the cell group information and the full paths to the binary files, the analysis object for a particular cell group can be constructed:

```
analysis = libHRLAnalysis.hrlSpikeAnalysis()
analysis.buildDataStructures(libHRLAnalysis.py_list_to_vector_string(
    binFiles),options.startTime,options.endTime,cellGroup['startIdx'],
    cellGroup['endIdx'])
```

The resulting analysis object can be used to fill in the desired analysis data structures. The spike raster data will fill two arrays, one containing a spike time and the other containing a cell index. Keep in mind there will be redundant spike times. This was originally intended for use in plotting the spiking activity as a raster plot. This is collected using:

```
# Create the vectors to hold the spike timing information
times = libHRLAnalysis.vector_int()
spikes = libHRLAnalysis.vector_int()
# collect the information
analysis.calcSpikeRaster(times,spikes)
```

The population level average spiking activity can be computed using:

```
# Get the Cell Rates.
windowRates = libHRLAnalysis.vector_double()
if options.windowRateType == "binned":
    analysis.calcWindowRate(windowRates,options.WindowSize,options.
        StepSize)
else:
    analysis.calcGaussWindowRate(windowRates,options.WindowSize,
        options.StepSize)
```

There are several functions for getting individual cell information either separately or in a single call. Here the runAnalysis.py script is gathering the average spike count rate over

the given interval as well as using that information to create bins of cell counts with activity in spike rate ranges.

```
# Get the Window Rates and the Binned Rates
cells = libHRLAnalysis.vector_int()
rates = libHRLAnalysis.vector_double()
counts = libHRLAnalysis.vector_int()
freqs = libHRLAnalysis.vector_double()
analysis.calcRatesWithBins(cells,rates,freqs,counts,100)
```

The coefficient of variation is calculated using:

```
# Get the COV analysis.
covCells = libHRLAnalysis.vector_int()
COV = libHRLAnalysis.vector_double()
analysis.calcCOV(covCells,COV)
```

### Running the network analysis

The network analysis can be completed by constructing a representation of the populations.

```
    networkAnalysis = libHRLAnalysis.hrlNetworkAnalysis()

    numOutputs = 3;
    numInputs = 4;
    outputStart = 0;
    inputStart = 210;

    for i in range(10):
        networkAnalysis.addDummyPopulation("ctx_%i"%(i+1), inputStart
            +(numInputs*i), inputStart+(numInputs*i)+numInputs - 1)

    for i in range(10):
        networkAnalysis.addPopulation("str_%i"%(i+1), outputStart+(
            numOutputs*i), outputStart+(numOutputs*i)+numOutputs - 1)

    networkAnalysis.buildNetwork('Data/net.bin')
    networkAnalysis.getWeights(fileName)
    networkAnalysis.outputWeights('populationNames.dat','weights.dat')
    networkAnalysis.outputGraph('network.dot')
```

### 6.2.3 Plotting the results

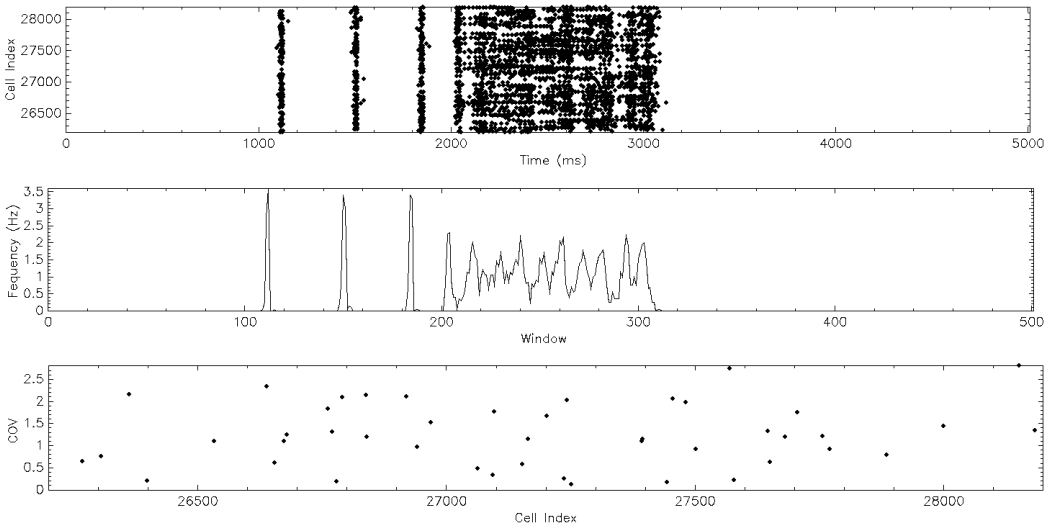



**Figure 6.3:** Simple plot using Biggles package

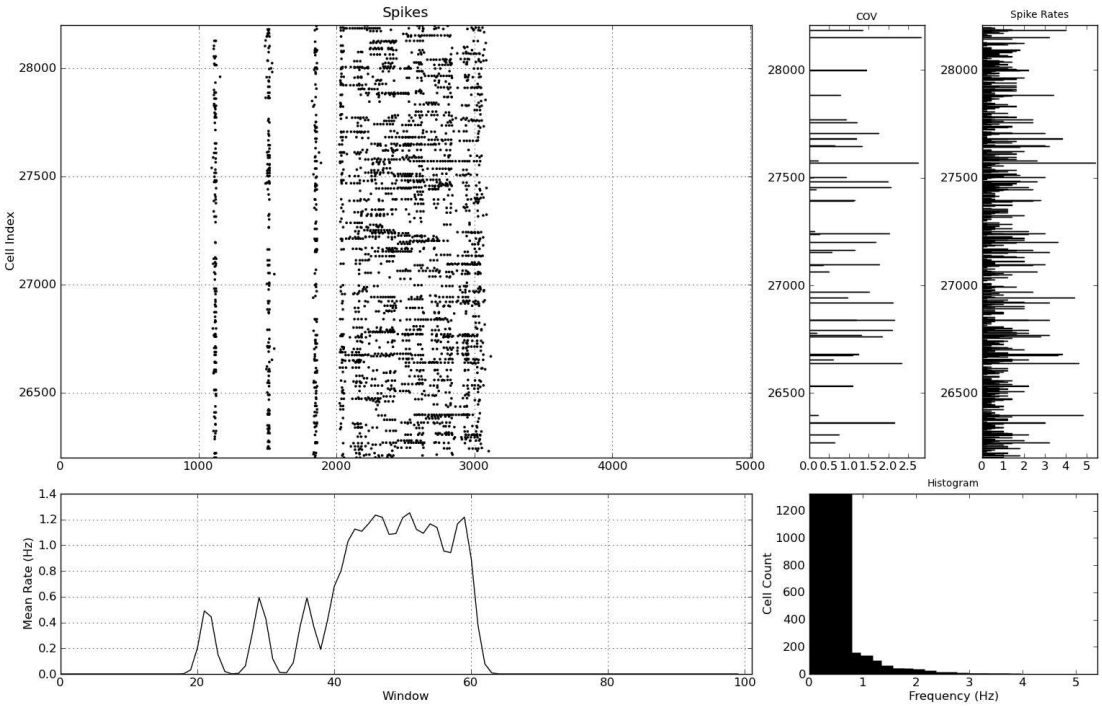



**Figure 6.4:** More detailed plot using Matplotlib package

There are several examples of plotting the calculated information provided by the suite. The main two use either the Python Biggles library or the Matplotlib library. The Biggles plots are simpler but the library is efficient. Unfortunately it is also less flexible as far as options and output formats are concerned. An example of calling the provided Biggles interface is:

```
import hrlAnalysisPlt_biggles
plotter = hrlAnalysisPlt_biggles.spikePlotterBiggles(cellGroup['name'
    ],options.startTime,options.endTime,cellGroup['startIdx'],cellGroup
    ['endIdx'])
plotter.plotRaster(times,spikes)
plotter.plotWindowRate(windowRates)

if len(COV) > 0:
        plotter.plotCOV(covCells,COV)

plotter.savePlot(os.path.join(options.outputPath,cellGroup['name']+'.
    png'))
```

The results of this command for the sample data provided with the package is shown in Figure 6.3.

The second plotting interface uses the Matplotlib library. This can be called using:

```
import hrlAnalysisPlt
plotter = hrlAnalysisPlt.spikePlotter(cellGroup['name'],options.
    startTime,options.endTime,cellGroup['startIdx'],cellGroup['endIdx'
    ])
plotter.plotRaster(times,spikes)
plotter.plotWindowRate(windowRates)
if len(COV) > 0:
        plotter.plotCOV(covCells,COV)

plotter.plotCellRates(cells,rates)
plotter.plotSpikeBins(freqs,counts)

plotter.savePlot(os.path.join(options.outputPath,cellGroup['name']+'.
    png'))
plotter.closePlot()
```

The resulting plot is presented in Figure 6.4.

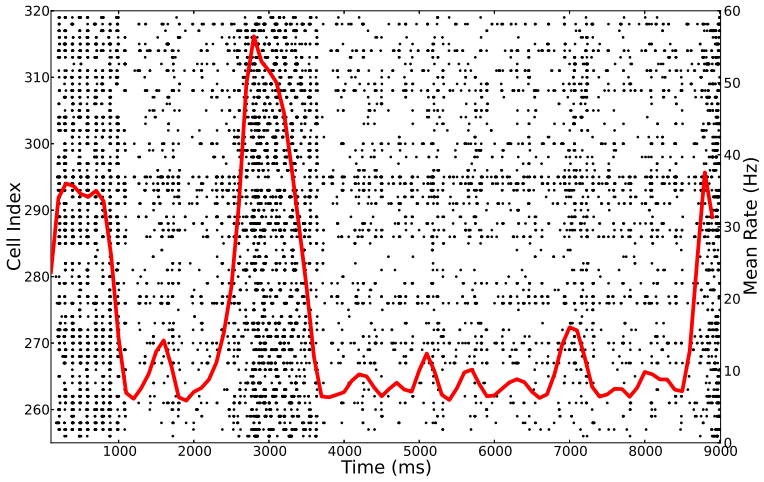



**Figure 6.5:** Example of overlaying raster activity with spike rate.

Since the returned data is treated in Python as lists, users are not restricted to the examples provided here. Figure 6.5 is an example of overlaying a raster plot with spike rate information. Similarly, the suite has been used for live animations of neural activity as well as for the construction of off-line animations and publication quality plots.

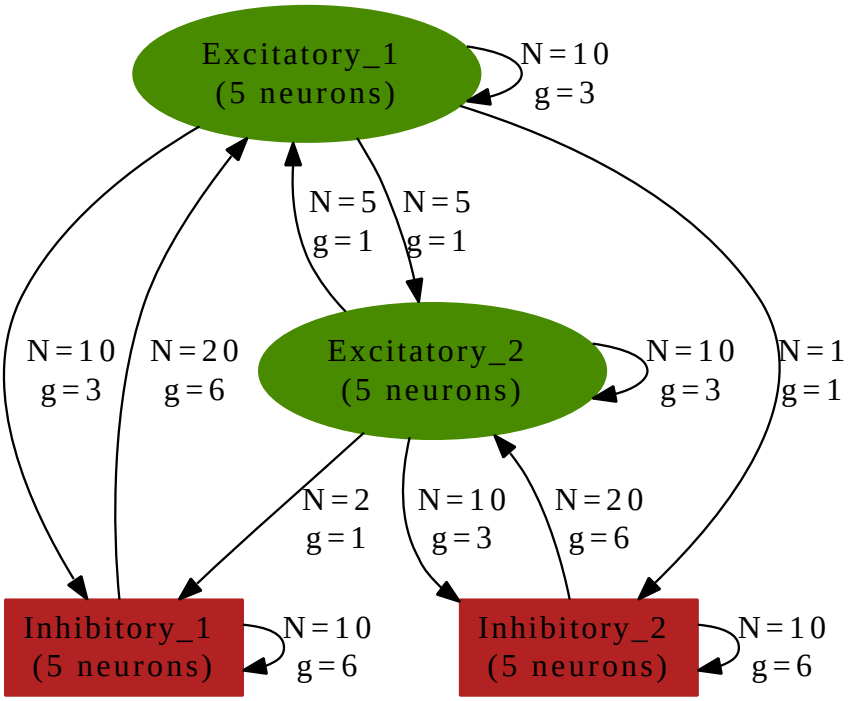



**Figure 6.6:** Example graph produced by graphviz.

The network analysis can return information about the average synaptic weight between populations as well as a graphical representation of the network at the population level. The network output uses the dot format of the Graphviz package. There are a number of free interpreters and converters for this format so users can also leverage other network analysis tools. Figure 6.6 illustrates a network created using Graphviz. Notice the number of connections and the average synaptic efficacy is included on the edges. The synaptic weights between populations can also be returned by the network analysis code. Figure 6.7 is an example of this.

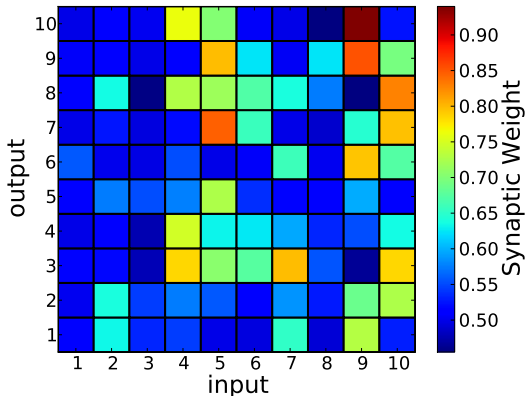



**Figure 6.7:** Example average synaptic weights.

## 6.3 Discussion

Although this work is geared towards processing data formatted by HRLSim, adapting it to other formats is simply a matter of overloading the input functions. The core analysis and high-level python code can all remain the same. Using this, support for other simulators will be added as needed.

Analyzing large data sets is a problem in numerous research areas; there are entire fields dedicated to processing "big data." In computational neuroscience there is a continual trend towards larger and more complex models. Currently, the tools for analyzing these models do not exist. Researchers are left to their own techniques and utilities for processing results. The HRLAnalysis package does handle large data sets on reasonable time scales. Processing $10$ seconds of results from a network with $100,000$ neurons can take minutes in MATLAB as opposed to a few seconds with HRLAnalysis. The included tools however, are still derived from theories based on small-scale network analysis. Fortunately, adding functionality to the HRLAnalysis libraries is straightforward. The comprehensive unit test suite that is included can shield existing functionality from additions users make to the

software. This helps ensure the package remains extensible as the HRLSim environment matures.

# Part II

# Modeling the Basal Ganglia

# Chapter 7

# Background

## 7.1 The basal ganglia

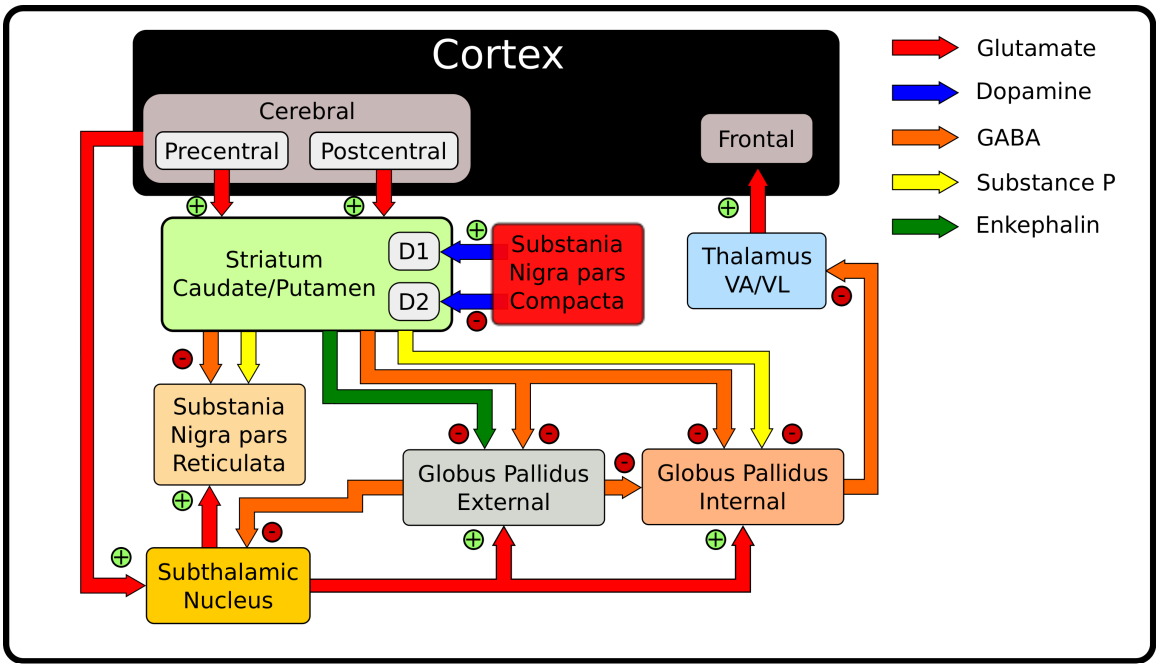



**Figure 7.1:** Basal ganglia box and arrow diagram.

The Basal Ganglia (BG) is a phylogentically ancient structure spanning the telencephalic and mesencephalic regions of the nervous system. This sub-cortical structure plays a role in a number of cognitive and behavioral phenomena that include action-selection, action-gating, timing, reinforcement-learning, working memory, fatigue, apathy, goal-oriented behavior and movement preparation. In addition, it is the epicenter of a number of neurological disorders that include Parkinson's Disease and Huntington's Disease as well as psychiatric disorders such as schizophrenia and obsessive compulsive behavior. Like many sub-cortical structures the BG has a topographic organization that is maintained throughout the nuclei that compose it. This organization has been the foundation on which most BG theories are grounded. However, new findings have revealed a more complex functional anatomy than previously believed.

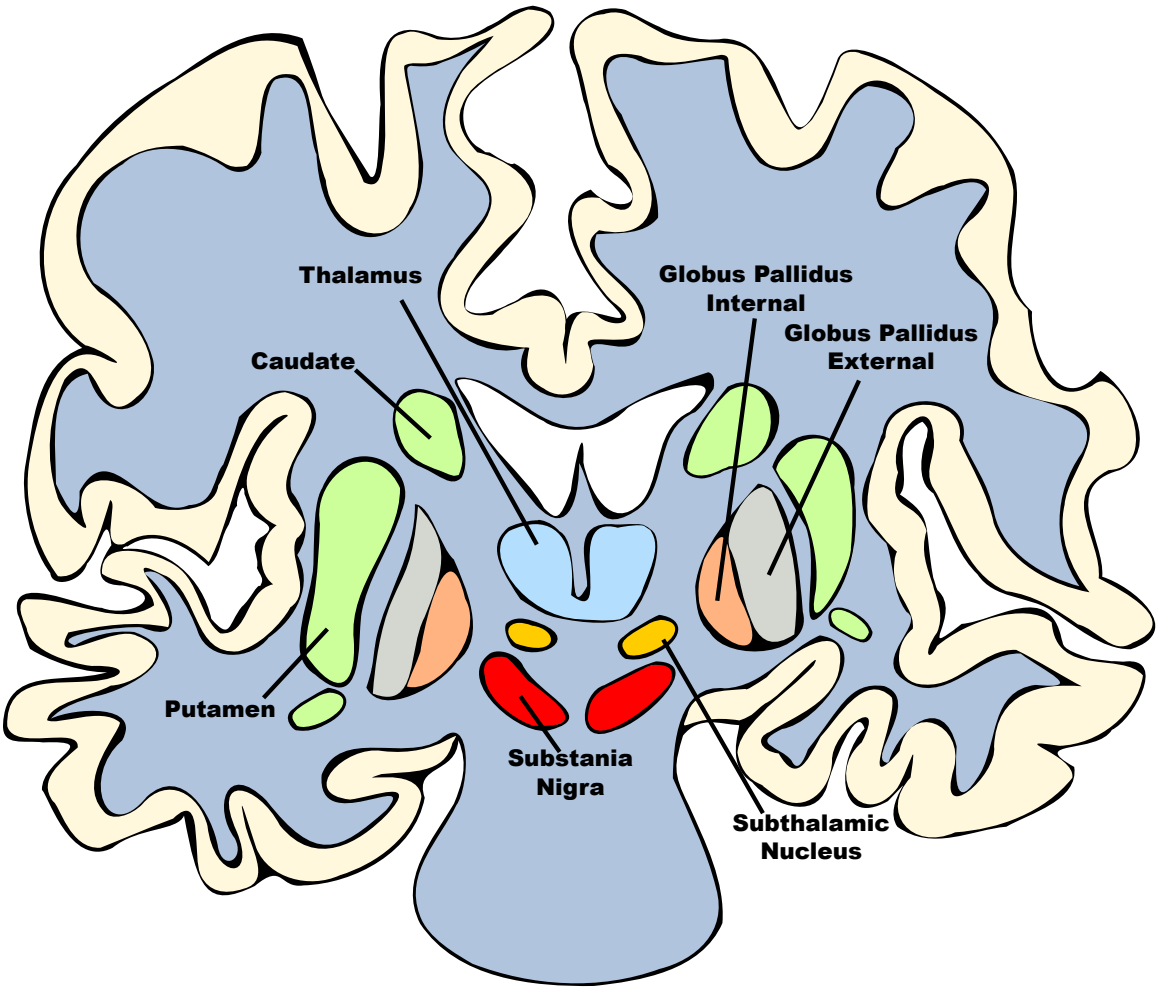



**Figure 7.2:** Basal ganglia nuclei in a coronal brain slice.

### 7.1.1 Anatomy

Historically there are six nuclei in models of the BG, as illustrated in Figure 7.1. These are the Striatum, the external segment of the Globus Pallidus (GPe), the internal segment of the Globus Pallidus (GPi), the Subthalmic Nucleus (STN), the Substania Nigra pars compacta (SNc), and the Substania Nigra pars reticulata (SNr). The major input into these nuclei come from a large number of cortical areas. In fact, almost every layered neocortical region contain outgoing connections from layer V into the striatum of the BG (Bolam et al., 2000;

Gerfen and Bolam, 2010). The output connections are split mainly between the brainstem and thalamus. Figure 7.2 shows the physical relation of these structures in a coronal slice of a fictitious mammalian brain.

### Multiple pathways

There is a separation of the functional anatomy of the BG nuclei that forms parallel paths through the organ. These paths have been defined as the "direct" pathway, so named because the input neurons directly connect to the output nuclei, the "indirect" pathway, named as such due to the elongated path through the inner BG nuclei, and the "hyper-direct" identified by cortical innervations directly connecting to the subthalamic nucleus. The functional significance of these paths is still debated however their existence is not. Historically the pathways have been functionally separated as the "go" (direct) and the "no-go" (indirect) pathways (O'Reilly, 2006; Cohen and Frank, 2009; Shouno et al., 2009; Chakravarthy et al., 2010; Krishnan et al., 2011). However, the role of the "hyper-direct" pathway is gaining more interest. We describe the major BG anatomy below.

### Striatum

In primates the striatum can be separated into two functional regions, the caudate and the putamen. The caudate is primarily innervated by prefrontal cortical connections. Whereas the putamen receives afferents preferentially from the motor and somatosensory regions (Gerfen and Bolam, 2010), they are often included as a single unit since there is no clear demarcation between the two. In addition, there are overlaps in cortical input to the putamen (Gerfen and Bolam, 2010).

The recipients of the cortical inputs are medium-size spiny GABAergic neurons that comprise about 95% of the striatum (Oorschot, 2010). These projection neurons are sepa-

rated based on the subcortical targets they project to. The “direct” pathway neurons directly innervate the output nuclei of the BG. Those neurons that comprise the “indirect” pathway connect to the intermediate structures as described above. The remaining 5% of neurons are interneurons that do not project beyond the boundaries of the striatum.

**Globus Pallidus External**

The external segment of the Globus Pallidus (GPe) is considered part of the indirect pathway and is composed mainly of spontaneously active inhibitory neurons that utilize GABA for neurotransmission. In the traditional BG models the inputs into the GPe are GABAergic inhibitory connections from striatum as well as glutamaterigc excitatory inputs from the subthalamic nucleus. The major output targets are a feedback connection to the subthalamic nucleus and a forward connection to the globus pallidus internal segment. However, in addition to these the GPe also contains projections to the substania nigra and back to the striatum.

**Globus Pallidus Internal**

The internal section of the Globus Pallidus (GPi) is one of the major output nuclei of the BG. This area contains mostly inhibitory neurons with a high level of basal activity ( 30Hz) (Humphries et al., 2006). The major input connections are inhibitory innervations from the GPe and the striatum as well as excitatory innervations from the subthalamic nucleus. The GPi innervates the thalamus and among other things is involved in limb and trunk movements.

### Subthalamic Nucleus

The Subthalamis Nucleus (STN) appears to contain only one type of neuron that is excitatory and releases glutamate (Gerfen and Bolam, 2010). Input into the STN arise from the GPe but also directly from the cortex. The latter innervations have been labeled by some as the “hyperdirect” pathway since this avoids the striatum directly.

### Substania Nigra pars compacta

Included here only for completeness, the Substania Nigra pars compacta (SNc) is at the core of the dopaminergic system of the midbrain. These neurons are spontaneously active and provide tonic and phasic releases of dopamine at about $5$ Hz (Cohen and Frank, 2009). The neurons of the SNc are densely connected and principally output to the patch/matrix layout of the striatum. The ventral region of the SNc connects to small islands or patches spatially segregated in the striatum. Whereas the neurons of the dorsal SNc project to the regions surrounding the patches, referred to as the matrix (Gerfen and Bolam, 2010). The functional implications of this organization is still unknown.

### Substania Nigra pars reticulata

The Substania Nigra pars reticulata (SNr) is the other output nuclei of the basal ganglia and is responsible for head, neck and eye movements. The SNr is comprised mainly of GABAergic inhibitory neurons and similar to the GPi it has a high basal level of activity. The SNr receives inputs from STN and striatum and outputs to the superior colliculus, the thalamus and the pedunculopontine nucleus.

**Thalamus**

The primary role of the thalamus is believed to be to modulate and process the information entering the cortex (Sherman and Guillery, 2002). The output neurons of the BG play are prominent in modulating thalamic activity. A model thalamocortical relay neuron is used here to model that influence. These are bimodal neurons that alternate between a tonic firing mode and a burst firing mode depending on the voltage and time dependent $Ca^{2+}$ T-current (Sherman, 2001).

### 7.1.2 Neurocomputational modeling of the basal ganglia

Computational models of the basal ganglia provide a mechanism for generating novel hypothesis about BG function as well as providing direct interpretation of empirical results (Cohen and Frank, 2009). These computational models can not only bridge the high-level behavioral studies with the low-level electrophysiological experiments but also create novel working theories that are directly applicable to artificial intelligence. There are a number of BG models presented in the literature, each one selecting a different level of biological realism depending on the question begin asked. A selection of relevant publications that pertain to action-selection and reward-learning are presented here. Since the majority of BG models deal with disease states, a number of models that deal with Parkinson's disease are given as well. These models present both the dynamics and functional anatomy that this proposal is concerned with. Only the most relevant literature is presented so this list is necessarily incomplete.

### 7.1.3 Action selection

The model of Gurney Gurney et al. (2001) utilized rate based neurons to demonstrate action selection in the basal ganglia. This was later extended into the spiking domain by Humphries et al. (2006). This work is explained further below. The contraction model of Girard et al. (2008) created an action selection network for use in embedded robotics. The network was constructed using a dynamical theoretic approach and rate coding. The resulting model was integrated into a robot as an action selection mechanism for deciding between seven possible actions during a survival task. Shouno et al. (2009) and later Igarashi et al. (2011) designed a model of binary channel selection and output timing. The resulting network demonstrated probabilistic action selection and timing that they considered important for promoting behavioral variability during exploration and exploitation.

### 7.1.4 Reward learning

Izhikevich (2007b) solved the problem of credit-assignment by combining dopaminergic modulation with an eligibility trace that could reward neurons that fired in appropriate temporal-patterns. This mechanism allowed a test network to learn stimulus-specific responses. Chorley and Seth (2011) combined that mechanism with the dual path model of Tan and Bullock (2008). The resulting network demonstrated a number of physiologically relevant dopamine responses that the Izhikevich (2007b) model did not explore. For a review of reinforcement learning models refer to Cohen (2008) and Cohen and Frank (2009). In addition, Atallah et al. (2004) presents a review of earlier computational models of the basal ganglia with a focus on interactions between the hippocampus, cortex and BG as it relates to a number of functions including stimulus-response learning.

### 7.1.5 Combined models:

Brown et al. (2004) combined the frontal cortex with the BG to demonstrate how primates can integrate reactive and planned behaviors in a rate-based network. The simulations predicted how dopaminergic signals could guide the learning of saccadic eye movements. The model of Stewart et al. (2012) was based on the rate model of Gurney et al. (2001). It employed LIF neurons and the neural engineering framework (Eliasmith and Anderson, 2003) to convert the variables of the original model into populations of spiking neurons. Although this model was capable of learning the desired input output functions, it was not done through classical reinforcement learning. Instead, this was accomplished using a feedback function that would compare the error between the model output and desired output.

## 7.2 Parkinson's disease

Parkinson's disease is a neurodegenerative disorder characterized by a marked loss of dopaminergic neurons in the SNc. The fundamental symptoms of Parkinson's disease are tremor: the involuntary movements of the body and limbs, rigidity: stiffness in the muscles, bradykinesia: slowed movement, and postural instability: difficulty maintaining an upright position (Bezard et al., 2010). In addition, there are a number of secondary and non-motor symptoms that may be present. With no known cause or cure, Parkinson's disease affects tens of millions of people throughout the world (Bergman et al., 2010).

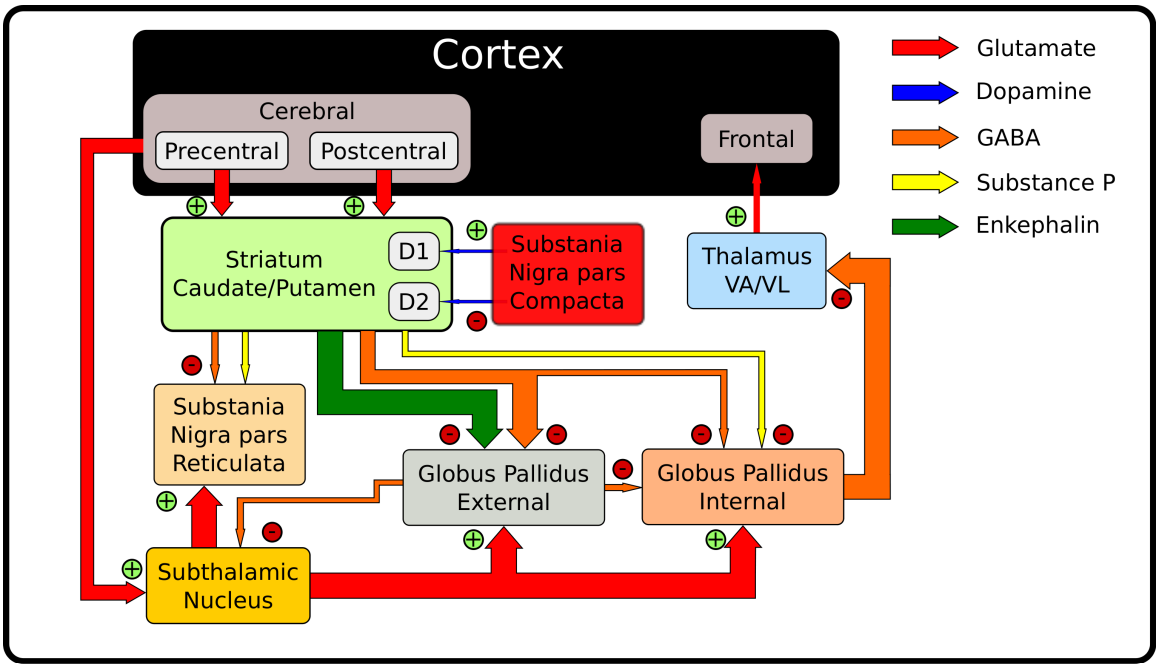



**Figure 7.3:** Parkinsonian Basal Ganglia

The loss of constant inhibitory influence on the striatum from the SNc results in an increased level of activity throughout the BG, Figure 7.3. It is still unclear how these increases in activity correspond to the symptoms of PD. Despite this gap in knowledge, there have been some therapeutic options that have demonstrated a benefit in alleviating the symptoms of PD. The reality however, is that there is no known way to stop the disease's progression.

Treatments of PD generally begin pharmacologically; dopamine replacement using levodopa is one such early treatment option. Unfortunately, not only is the the dosing schedule difficult and its benefit eventually deteriorates completely but there can be negative side-effects, such as dyskinesia: impediment of voluntary movement or hypertonia: increased muscle tone (Fahn et al., 2004). As the clinic benefit of dopamine replacement is lost many patients are treated with deep brain electrodes that constantly stimulate the BG.

### 7.2.1 Deep brain stimulation

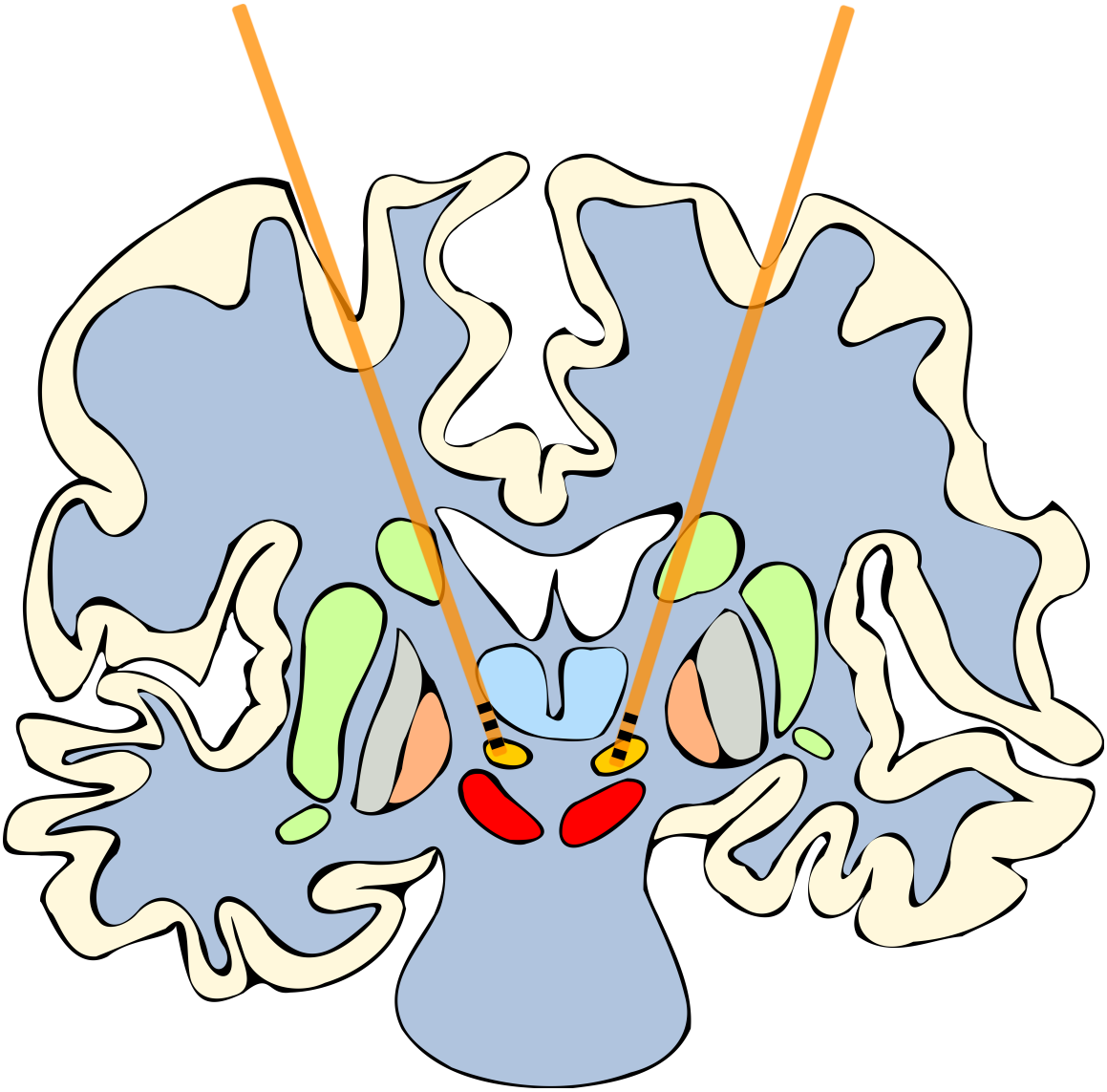


**Figure 7.4:** Example deep brain stimulation electrode placement.

Deep brain stimulation (DBS) was first reported in the early eighties but did not become a serious treatment option until a decade later (Montgomery, 2012). Prior to this, other than pharmacological treatments, PD patient's only other option was surgical ablation (Purves et al., 2007). DBS consists of two bilateral electrodes implanted in the deep structures of

the BG. Figure 7.4 illustrates how the path of the electrodes. Connecting wires are then implanted subcutaneously to the patients shoulders where battery and open-loop control electronics are located.

There is an immediate clinical benefit associated only with the placement of the electrodes, this is referred to as the microthalamotomy effect (Chang et al., 2009). A high-frequency electric pulse is then applied to the electrodes; creating an electric field within the target region of the brain. Clinicians will generally experiment with the frequency and amplitude of the electric pulse to find the region of highest benefit. Finding that point is an inexact science and periodic adjustments throughout the lifetime of the patient are required; as that point moves as compensatory mechanisms in the BG attempt to regulate the activity.

Despite the proven clinical benefit of DBS, there is no clear explanation for that benefit. This has been the focus of a number of research projects but some of the most compelling theories have come from computational studies of the BG.

### 7.2.2 Computational models of Parkinson's disease

Rubin and Terman (2004) offered the first explanation for the paradoxical therapeutic effects of deep brain stimulation (DBS) in a Parkinsonian BG. This was utilized by Feng et al. (2007) to explore feedback control of the DBS protocols. Similarly, it was extended by Pirini et al. (2009) to include the striatum and sensorimotor cortices. In Pascual et al. (2006) the overall usefulness of this and any model of Parkinson's disease was explored. Although the result was that this type of modeling can be useful, care should be taken when making specific claims about model results and predictions. The work of Frank (2005) looked at dopamine modulation and the cognitive deficits it had in medicated and non-medicated Parkinson's disease. Hahn and McIntyre (2010) explored a network of the subthalamopallidal (STN,Thalamus,GPe,GPi) network under deep brain stimulation. Leblois

et al. (2006) created a model of the BG without the GPe but included a feedback from the thalamus into the cortex. The resulting model explored only the direct and hyper-direct pathways in action selection. The network demonstrated a loss of action selection capabilities as dopamine input was reduced and predicted that this occurs before the oscillatory patterns associated with Parkinsonianism were present.

# Chapter 8

# Reward-Learning and Action-Selection in an Embodied Agent

Adding value to action-selection through reinforcement-learning provides a mechanism for modifying future actions. This behavioral-level modulation is vital for performing in complex and dynamic environments. In this chapter we focus on four classes of biologically inspired feed-forward spiking neural networks capable of action-selection via reinforcement-learning. These networks are embodied in a minimal virtual agent and their ability to learn two simple games through reinforcement and punishment is explored. There is no bias or understanding of the task inherent to the network and all of the dynamics emerge based on interactions with the environment. Value of an action takes the form of reinforcement and punishment signals assumed to be provided by the environment or a user. The variation in the four classes arises from different levels of network complexity based on differences in network architecture, the nature of network interactions including the interplay between excitation, inhibition and reinforcement, and the degree of bio-fidelity of the model. The models obey the constraints of neuromorphic hardware that are currently being developed,

including the DARPA SyNAPSE neuromorphic chips for very low power spiking model implementations. The simulation results demonstrate the performance of these models for a variant of classic pong as well as a first-person shooter. The results suggest that these models could serve as a building block for the control of more complex robotic systems that are embodied in real and changing environments.

## 8.1 Introduction

The combination of action-selection and reinforcement-learning in biological entities is essential for successfully adapting and thriving in complex environments. This is also important for the effective operation of intelligent agents. However, strategies for embedding artificial intelligence have resulted in agents with limited demonstrable emergent properties. Because of this, it is still unreasonable to deploy a neurorobotic entity and expect it to learn from and perform in its environment the same way biological entities can. Similarly, neural models require complex and varied input signals in order to accurately replicate the activity observed experimentally. One strategy for creating this complex stimuli is through immersing a model in a real or virtual environment capable of providing the feedback necessary for the model to extract value and interact appropriately. These are part of the motivations for the DARPA SyNAPSE program. Through the creation of low-power neuromorphic architectures both suitable for efficient remote operation and capable of replicating many of the biologically salient features of neural systems, the program can reduce the technological and theoretical barriers of embodied modeling.

Action selection is the appropriate negotiation of competing signals. In the mammalian nervous system the complex circuitry of the Basal Ganglia (BG) is active in gating the information flow in the frontal cortex by appropriately selecting between input signals.

This selection mechanism can affect simple action all the way up to complex behaviors and cognitive processing (Cohen and Frank, 2009). Although overly simplified, it can be helpful to relate the BG to a circuit multiplexer, actively connecting inputs to outputs based on the current system state.

Reinforcement or reward learning (RL) is the reinforcement of actions or decisions that maximizes the positive outcome of those choices. This is similar to instrumental conditioning where stimulus-response trials result in reinforcement of responses that are rewarded and attenuation of those that are not (Chakravarthy et al., 2010). Reinforcement-learning in a neural network is an ideal alternative to supervised learning algorithms. Where supervised learning requires an intelligent teaching signal that must have a detailed understanding of the task, reinforcement learning can develop independent of the task without any prior knowledge. Only the quality of the output signal in response to the input signal and current contextual state of the network is needed.

In this work we focus on three different classes of small biologically inspired feed-forward spiking networks capable of action-selection and reinforcement-learning while immersed in a virtual environment. Each is suitable for realization on the neuromorphic hardware developed under the SyNAPSE project and provides a theoretical framework for testing future novel reinforcement-learning algorithms. These networks are embodied in a minimal virtual agent and their ability to learn a simple ping-pong game through reinforcement and punishment is explored. There is no bias or understanding of the task inherent to the network and all of the dynamics emerge based on interactions with the environment. Value of an action takes the form of simple reinforcement and punishment signals.

This concept is then extended by exploring how these networks can be combined to perform more complex actions. Towards this goal, a first-person shooter was developed. A model combining multiple RL networks was then constructed and trained to target and

shoot the most appropriate enemy.

Beyond supporting hardware validation, the resulting models are ideal for simple robotic embodiments. In addition, these are capable of demonstrating reinforcement-learning and action-selection in different ways.

## 8.2 Design and Methods

### 8.2.1 Neuron model

The neural model supported by the initial SyNAPSE hardware is the Leaky-Integrate and Fire (LIF) neuron. The LIF model is defined by

$$C_m \frac{dV}{dt} = -g_{leak}(V - E_{rest}) + I. \tag{8.1}$$

Where

$C_m$ is the membrane capacitance.

$I$ is the sum of external and synaptic currents.

$g_{leak}$ conductance of the leak channels.

$E_{leak}$ is the reversal potential for that particular class of synapse.

As the current input into the model neuron is increased the membrane voltage will proportionally increase until the threshold voltage is reached. At this point an action potential is fired and the membrane voltage is reset to the resting value. The neuron model is placed in a refractory period for 2 milliseconds where no changes in membrane voltages are allowed. If the current is removed before reaching the threshold the voltage will decay to $E_{rest}$. The LIF model is one of the least computationally intensive neural models but is still capable of replicating many aspects of neural activity (Burkitt, 2006).

The connections between neurons are modeled by conductance-based synapses. The general form of that influence is defined as

$$I_{syn} = g_{max} \cdot g_{eff} \cdot (V - E_{syn}). \tag{8.2}$$

Where

$g_{max}$ is the maximum conductance for that particular class of synapse.

$g_{eff}$ is the current synaptic efficacy between $[0, g_{effmax}]$.

$E_{syn}$ is the reversal potential for that particular class of synapse.

To simulate the buffering and re-uptake of neurotransmitters, the influence that a presynaptic action potential has on a neuron can be decayed based on a specified time constant. This process is abstracted using

$$\tau_{syn} \frac{dg_i^{syn}}{dt} = -g_i^{syn} + \sum W_{ji} \delta \left(t - t_j\right). \tag{8.3}$$

An Euler integration method is used with time step $\tau = 1ms$.

Learning at the synaptic level is achieved through the spike-timing dependent plasticity rules defined by Song *et al.* Song et al. (2000).

$$g_{eff} \rightarrow g_{eff} + g_{effmax} F(\Delta t)$$

Where

$$\Delta t = t_{pre} - t_{post}$$

$$F(\Delta t) = \begin{cases} A_+ e^{\left(\frac{\Delta t}{\tau_+}\right)} \\ A_- e^{\left(\frac{\Delta t}{\tau_-}\right)} \end{cases}$$

if $(g_{eff} < 0)$ then $g_{eff} \rightarrow 0$

if $(g > g_{effmax})$ then $g_{eff} \rightarrow g_{effmax}$

The global parameter values used in this study are presented in Table 8.1.

**Table 8.1:** Global model parameters.

| **Parameter** | **Value** | |
|---|---|---|
| $C_m$ | 1. | (pF) |
| $\tau_{ge}$ | 5. | (ms) |
| $\tau_{gi}$ | 100. | (ms) |
| $E_{exc}$ | 0. | (mV) |
| $E_{inh}$ | $-80.$ | (mV) |
| $V_{rest}$ | 0. | (mV) |
| $A_+$ | 0.025 | |
| $A_-$ | 0.026 | |
| $\tau_+$ | 20. | (ms) |
| $\tau_-$ | 20. | (ms) |

### 8.2.2 Networks

Three possible embodiments of this idea are presented here. Initially, each of these networks have no knowledge or inherent understanding of their environment. The behavior is learned through feedback from the environment in the form of reward and punishment signals encoded as either random or structured spike events. These signals strengthen or weaken the synaptic connections between neurons; reinforcing the appropriate action.

Although a technological breakthrough, the initial SyNAPSE hardware has limited functionality. This has direct implications on the types of neural networks it can support. There are two phase 1 chips that have been developed, each with a common set of limitations. The most significant is the lack of fixed weight synapses. All connections, including those coming from dummy neurons, are plastic. This presents a number of problems but the most egregious is that it effectively eliminates the use of inhibitory neurons. In addition,

each neuromorphic processor (NP) has very different timing properties, number afferent inputs, and number of usable neurons.

**Excitatory only network**

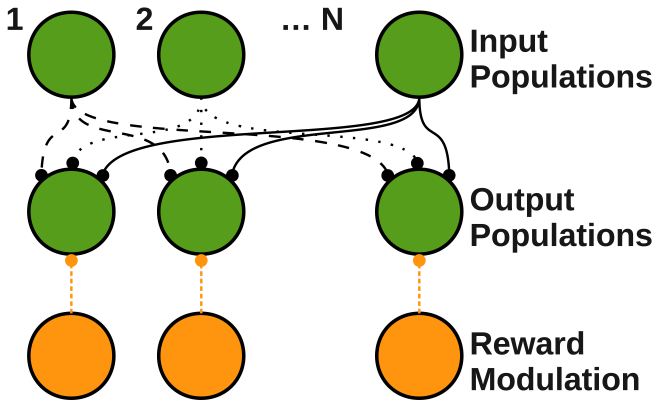



**Figure 8.1:** Excitatory neuron only network.

**Table 8.2:** Parameters for the excitatory only network.

**A. Neuron parameters**

| **Neural Region** | **Neurons Per Channel** |
|---|---|
| Input | 3 |
| Output | 3 |
| Reward | 1 |

**B. Connections**

| **Source → Destination** | **Synaptic Conductance** $(g_{max}) \cdot (g_{eff})$ | **Number of Incoming Connections (total)** |
|---|---|---|
| Input → Output | $(10.0) \cdot (0.25)$ | 15 |
| Reward → Input | $(10.0) \cdot (1.0)$ | 1 |

The first model explored was a simple feed-forward network that consists entirely of excitatory neurons, Figure 8.1. This network is compatible with the first phase neuromorphic

processors of the SyNAPSE project. Notice that the input layer is divided into channels represented by a population of neurons. The parameters are presented in Table 8.2. The total size of the network is 70 neurons. Note that each output neuron receives a maximum of 16 inputs. These connections are randomly created from the entire input population to ensure that there is no bias between input and output channels. The connections between the reward populations are focused projections within a channel.

**Lateral-inhibition network**

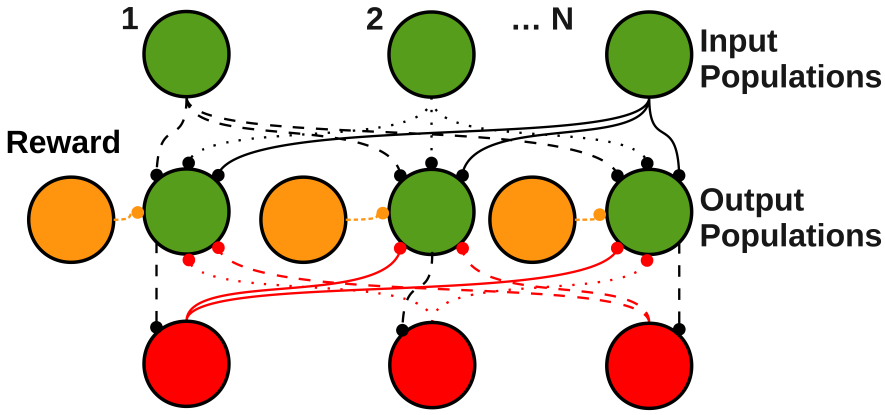



**Figure 8.2:** Lateral-inhibition network.

**Table 8.3:** Parameters for the lateral-inhibition network.

**A. Neuron parameters**

| **Neural Region** | **Neurons Per Channel** |
|---|---|
| Input | 3 |
| Output | 3 |
| Inhibition | 3 |
| Reward | 1 |

**B. Connections**

| **Source → Destination** | **Synaptic Conductance** $(g_{max}) \cdot (g_{eff})$ | **Number of Incoming Connections (total)** |
|---|---|---|
| Input → Output | $(10.0) \cdot (0.25)$ | 15 |
| Output → Inhibition | $(10.0) \cdot (1.0)$ | 15 |
| Inhibition → Output | $(10.0) \cdot (1.0)$ | 15 |
| Reward → Input | $(10.0) \cdot (1.0)$ | 1 |

As an extension of this idea, lateral inhibition between the output populations is added, as shown in Figure 8.2. This creates an on-center off-surround network where the most active population suppresses the other output populations. Not only is this a more biologically realistic network but it also offers more control in the selection process. The parameters of this model are included in Table 8.3. A key aspect of this network are the diffuse connections of the inhibitory interneurons. These populations project to every other output population; excluding the channel of which they are a part of.

### Basal ganglia direct pathway

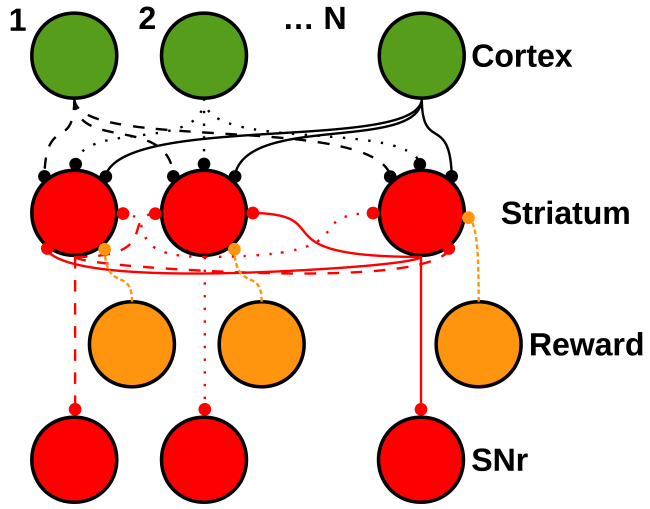



**Figure 8.3:** basal ganglia direct pathway network.

**Table 8.4:** Parameters for the basal ganglia direct pathway.

**A. Neuron parameters**

| Neural Region | Neurons Per Channel |
|---|---|
| Cortex (Ctx) | 4 |
| Striatum (Str) | 3 |
| Substania Nigra pars reticulata (SNr) | 3 |
| Excitatory | 9 |
| Reward | 6 |

**B. Connections**

| Source → Destination | Synaptic Conductance | Number of Incoming Connections (per channel) |
|---|---|---|
| Ctx → Str | 0.1 | 4 |
| Str → Str (diffuse) | 10.0 | 3 |
| Excitatory → SNr | 0.08 | 3 |
| Str → SNr | 10.0 | 3 |
| Reward → Str | 10.0 | 6 |

The third network presented is an implementation of the direct pathway of the Basal Ganglia (BG), Figure 8.3. This network emulates the physiological activity of the BG direct pathway where the neurons of the SNr are tonically active, firing around 30 Hz. This basal activity is suppressed by the inhibitory afferents of the striatum, resulting in a dis-inhibitory mechanism of action. Learning occurs between the cortex and the neurons of the striatum to develop the appropriate input-output channel combinations.

Physiologically neurons in the SNr are tonically active. The LIF neuron however, is not capable of replicating that spontaneous activity. To compensate, a Poisson random excitatory input is injected into the SNr populations. In addition, low-level uniform random noise is injected into the network.

**Combined reward learning network**

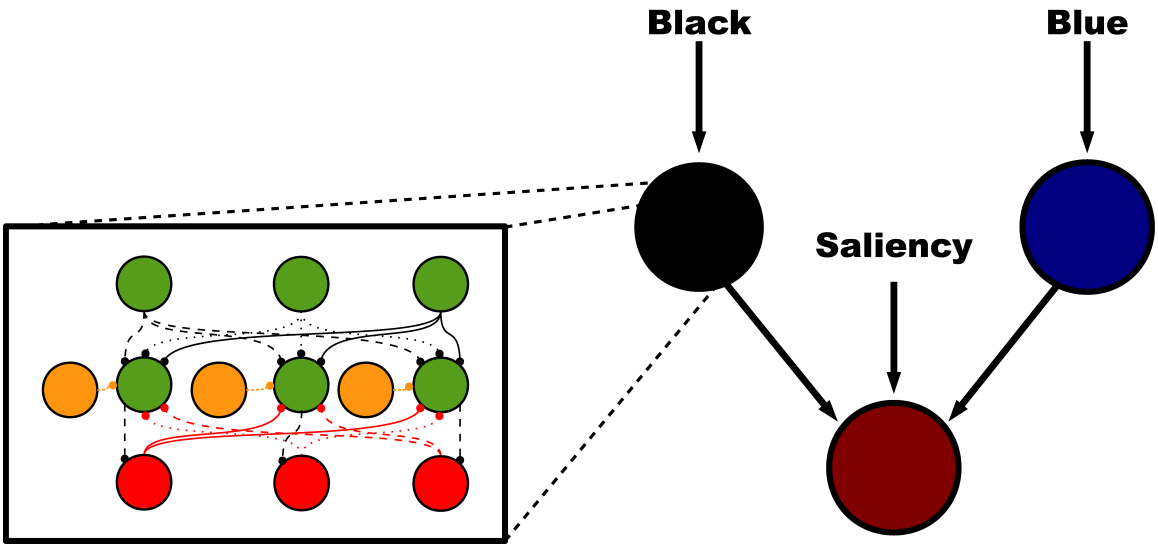



**Figure 8.4:** FPS Control Network

Although simple, these networks are capable of distinguishing competing inputs under noisy conditions. They can also be used as building blocks to perform more complex tasks. To illustrate this concept we combine three of the lateral-inhibition networks of Section

8.2.2. Each of the networks is divided into multiple channels with the outputs of two of the channels directly connecting to the corresponding input channel of the third, Figure 8.4. The connections are made one-to-one at a weight of $0.5$, with output channel $1$ connected to input channel $1$, output channel $2$ to input channel $2$, and so on.

As illustrated in Figure 8.4, each of the three networks receives a different input signal. Through reinforcement the network can learn to appropriately respond to different combinations of inputs. In this case, these are used to play a first-person shooter, described below.

**Stimulus learning**

Learning in these networks is driven by a conditioned stimulus injection. Stereotyped spiking signals sent to an input population and all of the reward populations. The timing of the signal is delayed for the target channel so the synaptic learning between the input population and the desired output populations is potentiated, while all other channels are depressed. The stimulus period lasts for either $300$ or $500$ $ms$.

### 8.2.3 Games

**Pong**

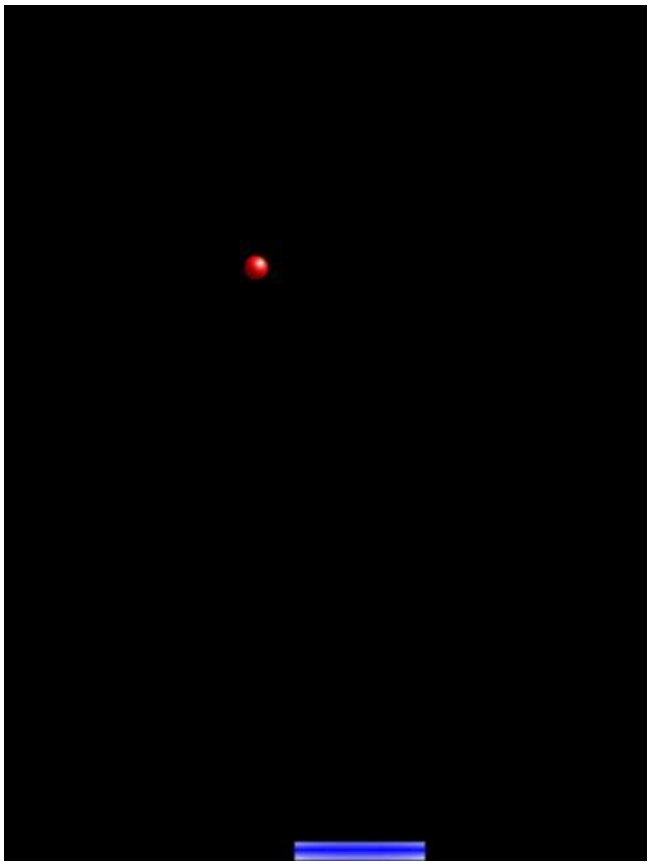


**Figure 8.5:** Pong game mock-up.

To illustrate the capabilities of these networks a pong style virtual environment was implemented. Figure 8.5 is a mock-up of that environment. This version of the game has a single player controlling the paddle at the bottom of the board. The puck bounces off of the left, right and top walls with minimal physics that change the speed of the puck based on the angle of incidence with the wall. The player has to move the paddle to block the puck from falling through the bottom of the game board.

The game was developed in different stages. First, A mock-up of the game was created

in Python using PyGame (Shinners, 2012). A game controller was then developed in C++ using the concepts of Section 5. The C++ controller has no visualization capabilities. It compiles directly into the HRLSim experiment and provides the virtual environment for the networks. The output of the environment is recorded by the controller and can then be played back by the Python visualizer.

The game engine was extended to support multiple players. In these instances two players control paddles on opposite sides of the board. A player scores a point when the puck gets past the opposing players paddle. Using this the Phase 1 excitatory network and the Phase 2 lateral-inhibition networks competed against each other.

In addition, a live real-time version was developed with a socket server to support connections from an external player. This embodiment does not synchronize with the external player so both the game and the network can run at different speeds. The board is sampled by the player and commands to control the paddle are sent after processing. This was created for coupling with the neuromorphic processors.

The position of the puck in the game space is sent to a number of discretized neural channels. Each of these channels represents a vertical column of the game board. The input signal is Poisson random spike events with a rate determined by a Gaussian curve, described below. This provides a noisy input signal with overlap between channels. The networks signal, through a winner-takes-all mechanism, the position of the paddle.

The stimulus into the network is determined by the location of the puck relative to each of the spatial channels. The location of the puck on the map determines the peak amplitude and center of a Gaussian function defined as

$$f_{X_c}(X^*) = ae^{-\left((X_c - X^*)^2/2c^2\right)} \tag{8.4}$$

Where

| | |
|---|---|
| $a$ | Peak amplitude of the Gaussian function. |
| $b$ | Center of the Gaussian function. |
| $c$ | Spatial width or $\sigma$ of the Gaussian function. |
| $X_c$ | The non-dimensional location of the channel. |

The peak amplitude and Gaussian center are defined as

$$a = Y^* \cdot R_{max} \tag{8.5}$$

$$b = X^* \tag{8.6}$$

Where

| | |
|---|---|
| $Y^*$ | Non-dimensional location of the puck in the $y$ dimension. |
| $R_{max}$ | Maximum input stimulus in $^{Spikes}/_{s}$. |
| $\Theta^*$ | Non-dimensional location of the puck in the $x$ dimension. |

This is visualized in Figure 8.6 for the three different spatial widths, $c$, used for the experiments presented here. The reward or punishment to the network arrives when the puck reaches the bottom of the game board.

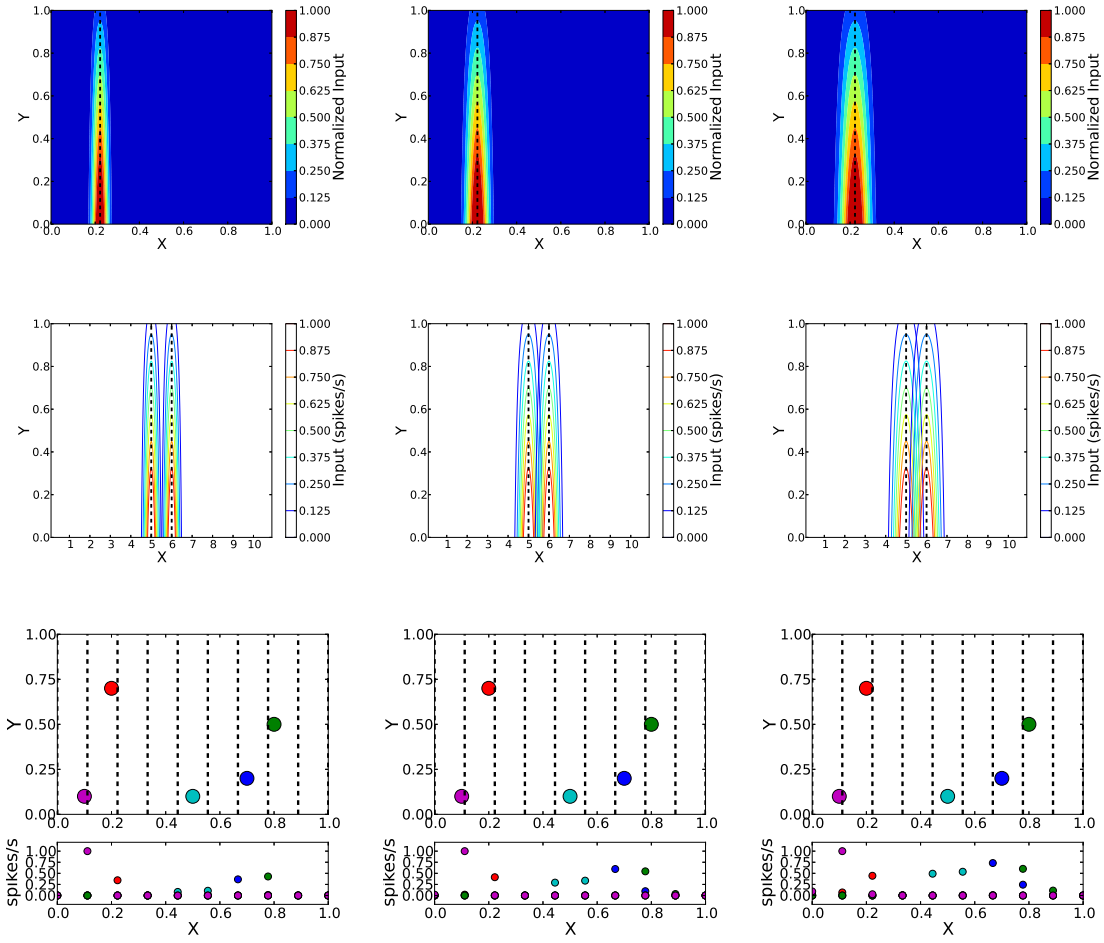



**Figure 8.6:** Pong game board disrectizations for a $10$ channel network. Three different spatial widths, $c$, are used for these experiments, $0.025$ (left column), $0.035$ (middle column), $0.045$ (right column). In the top row are the stimulus maps for channel $2$ for each of the spatial widths. The middle row illustrates the overlap between two consecutive channels. The bottom row shows how the location of the puck (top) translates to input stimulus for each of the $10$ channels (bottom).

### 8.2.4 Pong controller

The paddle in the simulated pong environment is controlled by a simple proportional controller. The environment receives discrete locations from the neural network. The location on the screen that the paddle has to move to is calculated based on these discrete locations. Its velocity in the $X$ direction is defined by

$$V_x = V_{max} \cdot P. \tag{8.7}$$

The variable $P$ is the output of the proportional controller defined by

$$P = k \cdot e. \tag{8.8}$$

Where $k$ is the gain variable and $e$ is the error between the target and current locations

$$e = X_{Location} - X_{Target} \tag{8.9}$$

The output of the proportional controller, $P$, is a piecewise linear function that is dependent on the distance from the target.

$$P = \begin{cases} -1 & -e < -\frac{1}{k} \\ 1 & e > \frac{1}{k} \\ e - k & |e| \leq \frac{1}{k} \end{cases}$$

This ensures that the speed of the paddle does not exceed the maximum defined velocity. The pivot point $\frac{1}{k}$ is calculated by setting $k \cdot e = 1$. In addition, the proportionality constant $k$ is less than $1$ to ensure that the paddle slows down as it gets closer to its target.

**Neuralstein first-person shooter**

The first-person shooter (FPS), Neuralstein, is similar to one of the original FPS games, Wolfenstein (id Software, 2012). This implementation is a rail-shooter where the player moves along a specified path. The player controls its forward movement along that path,

where it is aiming and when to take a shot. Similar to the pong environment described above this was implemented at different levels of abstraction. The game engine and visualization was developed in Python, with the latter using PyGame (Shinners, 2012) and the Pyggel library (Pyggel group, 2012). The game engine is abstracted away from the visualization to facilitate faster simulations. Communication with the simulations is provided through a socket server. The engine and the simulation are synchronized so the performance is determined by the slowest component.

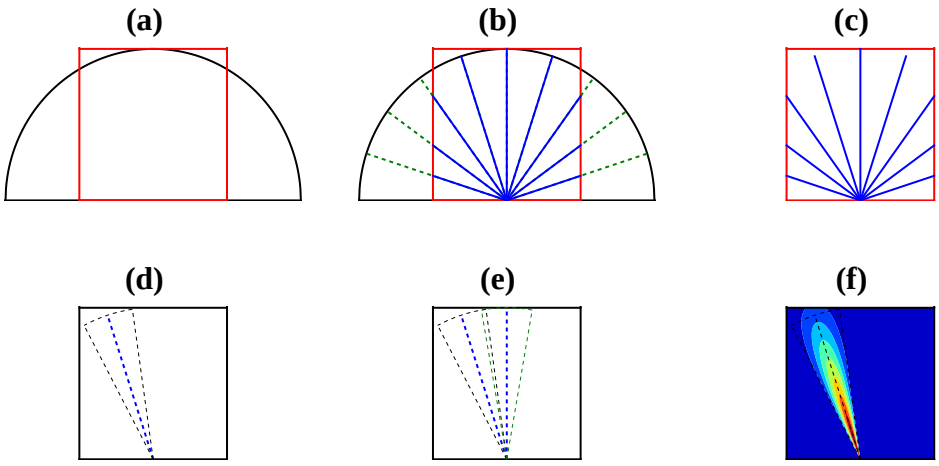



**Figure 8.7:** FPS Discretization. (a) A rectangular frame is taken from the hemispherical point-of-view (POV) of the player. (b) The POV space is discretized into equal segments (channels). (c) The resulting frame segments the players view of the world. (d) Each of the channels is centered along equal angular steps about the space with arc-lengths defining the stimulus regime for that channel. (e) The channels are constructed with overlapping stimulus regions to create a noisier environment for the networks to negotiate. (f) The stimulus space for a single channel is defined by a Gaussian function that is railed to the segment boundary.

The game board is discretized based on the players perspective. The hemispherical point-of-view (POV) for the player is partitioned into a rectangular region, Figure 8.7 (a). The POV is then segmented into discrete channels with centers at equally spaced angles along the hemisphere, Figure 8.7 (b). This defines the center for each of the channels that are represented by the network, Figure 8.7 (c). The channels creates a pie shaped region

of interest, Figure 8.7 (d), which have arc lengths with a $10\%$ overlap between channels, Figure 8.7 (e). Each of the segments defines that channels stimulus map, which is described by a Gaussian function, Figure 8.7 (f).

$$f_{\Theta_c}(\Theta^*) = ae^{-\left((\Theta_c-\Theta^*)^2/2c^2\right)} \tag{8.10}$$

Where

| | |
|---|---|
| $a$ | Peak amplitude of the Gaussian function. |
| $b$ | Center of the Gaussian function. |
| $c$ | Spatial width or $\sigma$ of the Gaussian function. |
| $\Theta_c$ | The non-dimensional angular location of the channel. |

The peak amplitude and Gaussian center are defined as

$$a = r^* \cdot R_{max} \tag{8.11}$$

$$b = X^* \tag{8.12}$$

Where

| | |
|---|---|
| $r^*$ | Non-dimensional location of the puck in the radial dimension. |
| $R_{max}$ | Maximum input stimulus in $Spikes/s$. |
| $\Theta^*$ | Non-dimensional angle of the puck. |

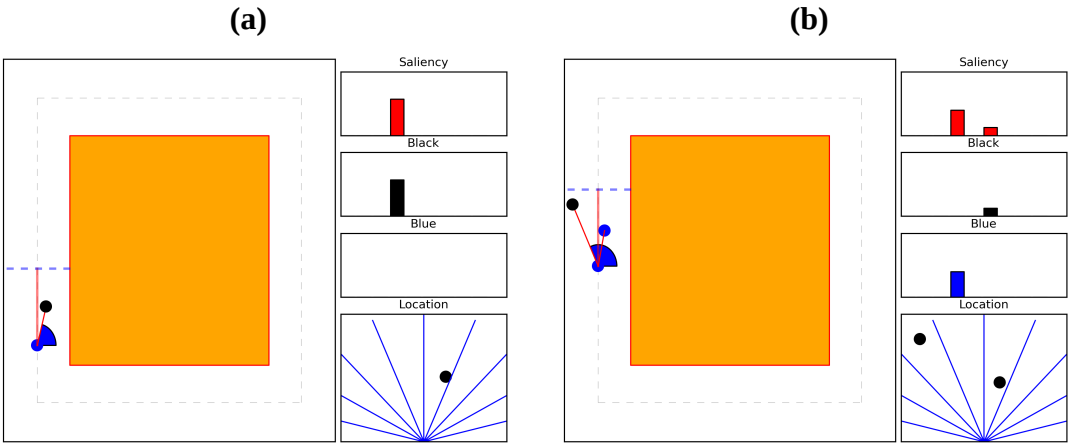



**Figure 8.8:** Example stimulus encoding and FPS game board. (a) Enemy only. (b) enemy and friend.

The overall arena is a square track with equal width, Figure 8.8. As the player moves through the environment game elements enter into the view of the player. Elements in the players POV are picked up and their location in that view creates the input stimulus injected into the saliency channels of the network.

There are two types of game elements right now. The primarily black characters are considered dangerous and the characters with blue accenting are considered innocuous. Each of these creates a different input into the black and blue channels respectively. It is assumed that a separate mechanism identifies the element and determines which channel is stimulated. For this implementation the game engine directs the stimulus. Figure 8.8 (a) illustrates the stimulus for a black element in the players POV. Figure 8.8 (b) illustrates what the stimulus for two different game elements would be.

## 8.3 Results

### 8.3.1 Excitatory only network

#### Basal activity

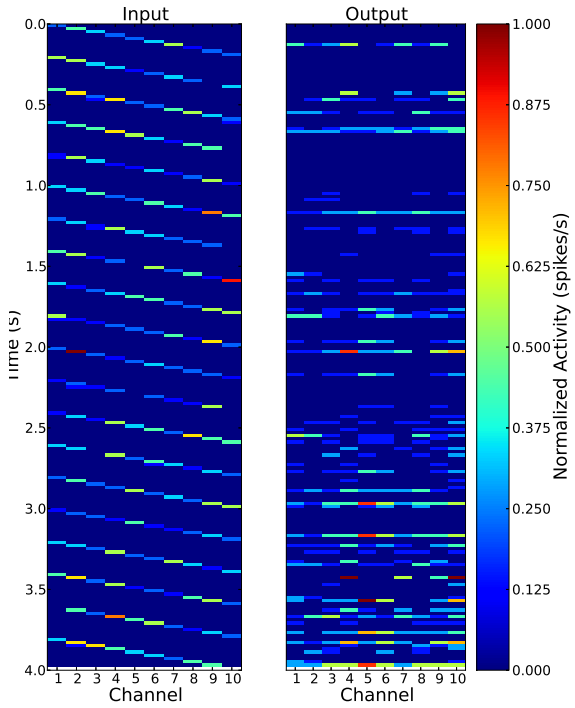



**Figure 8.9:** Basal activity of the excitatory only network. Notice that the time is moving from the top of the chart down. The x-axis is the respective input or output channel and the rate is normalized based on the peak output of the network.

Initially all of the networks are randomly constructed with no intentional bias between channels. Figure 8.9 illustrates the random response of an excitatory only network to changing input channels. Raster plots of this activity is presented in Figures 8.10 (a) and (b). The plot in Figure 8.10 (c) is the average synaptic efficacy value between input and output channels. Figure 8.10 (d) highlights the input/output pair with the highest average synaptic weight. Theoretically this marks the output population that should be maximally activated when that input population is activated. This is important for identifying how

well the networks are learning.

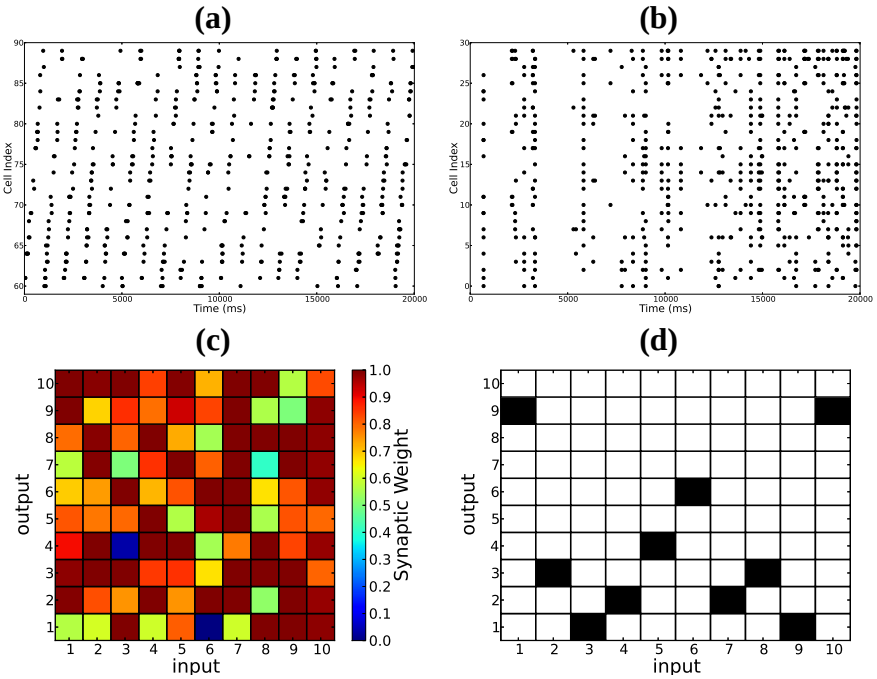



**Figure 8.10:** Results of 20s of Basal level activity. (a) Input raster. (b) Output raster. (c) Average synaptic weights for each of input-output pairs. (d) Location of maximum average synaptic weight between each pair.

### Learning capabilities

An important characteristic of this class of networks is its ability to learn arbitrary output responses. The random initial weights can be driven, through the stereotyped reward feedback, to desired values. Here we demonstrate the networks ability to learn one set of associations and later learn whole new set of associations. This scenario is illustrated by the spiking activity presented in Figure 8.11 (a). The stages, marked by the letters in the center are:

A. The network is initialized with all input/output connections have a synaptic USE value of 0.25; as illustrated in Figure 8.12a by the heat map of the average weights between input/output populations.

B. A Poisson random input is injected into consecutive channels for 10 seconds to establish the basal activity of the network. The resulting average synaptic weight matrix is shown in Figure 8.12b

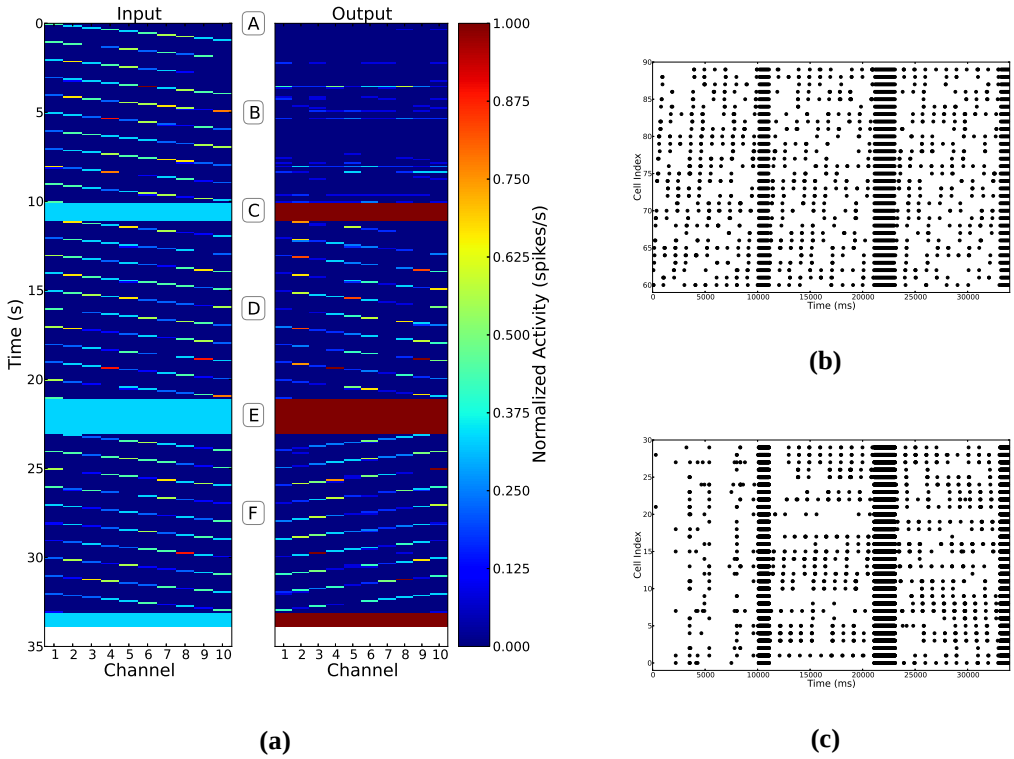



**Figure 8.11:** Excitatory network: reward-learning scenario. (a) Activity rate map of the example scenario. Activity was calculated using a moving Gaussian weighted window. (b) Spike raster of the input populations. (c) Spike raster of the output populations.

C. Alternating reward signals are sent to establish single input/output pairs. The weight matrix is now dominated by the diagonal shown in Figure 8.12c.

D. The repeated Poisson input signals from B are injected for 10 seconds. After this, the weight matrix shown in Figure 8.12d demonstrates further potentiation of the established input/output pairs and a continued depression of the other connections.

E. An opposite set of input/output associations are established using alternating reward signals. For stable retraining of the network the reward protocol needs to be about

twice as long as the original training. The new weight matrix is shown in Figure 8.12e.

F. 10 seconds of the repeated Poisson inputs illustrate the newly established input/output pairs, Figure 8.12f.

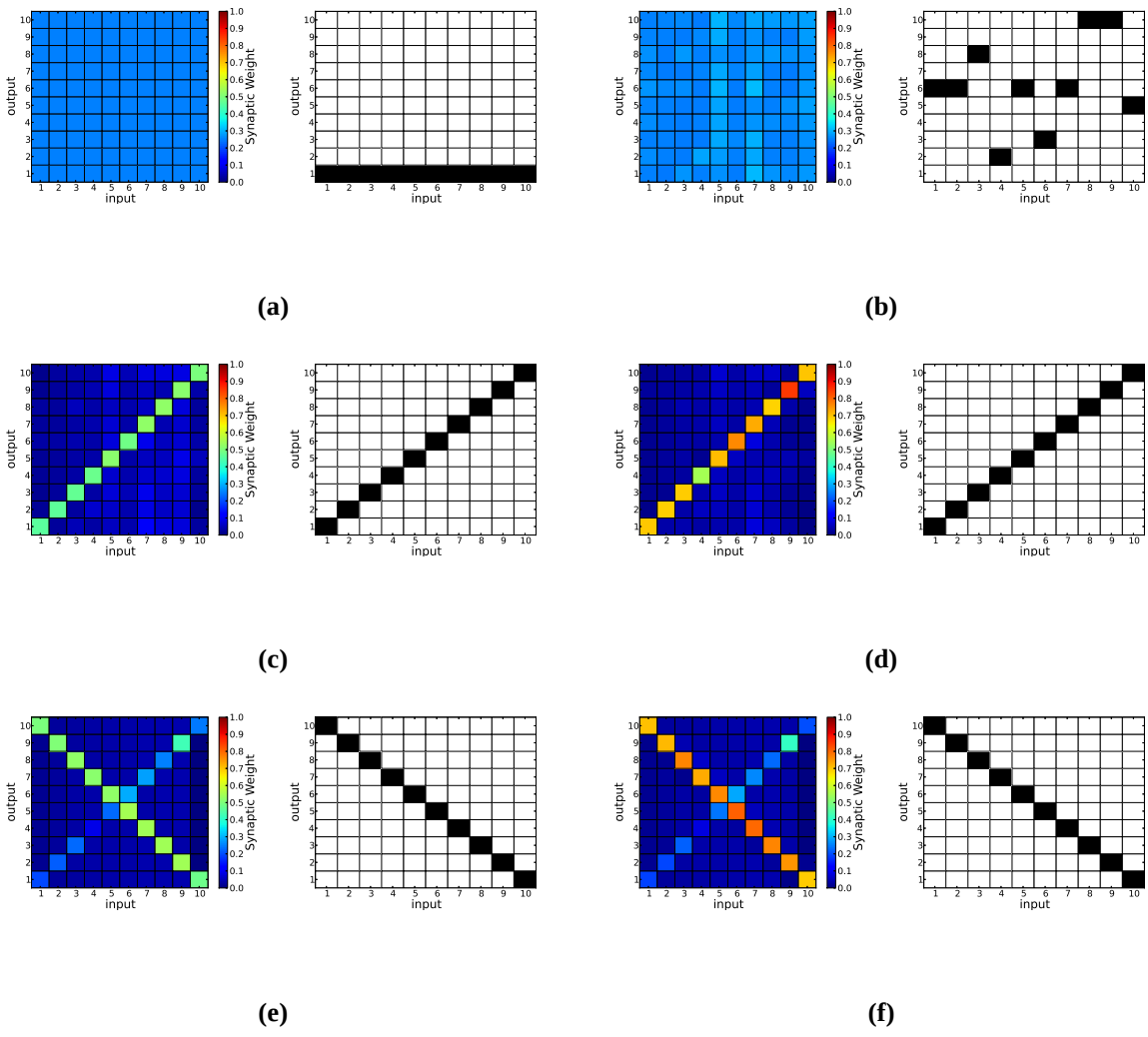



**Figure 8.12:** Average and maximum synaptic weights between input/output pairs after learning corresponding to Figure 8.11. (a) 0 sec (b) 10 sec (c) 11 sec (d) 21 sec (e) 22 sec (f) 33 sec

**Pong environment**

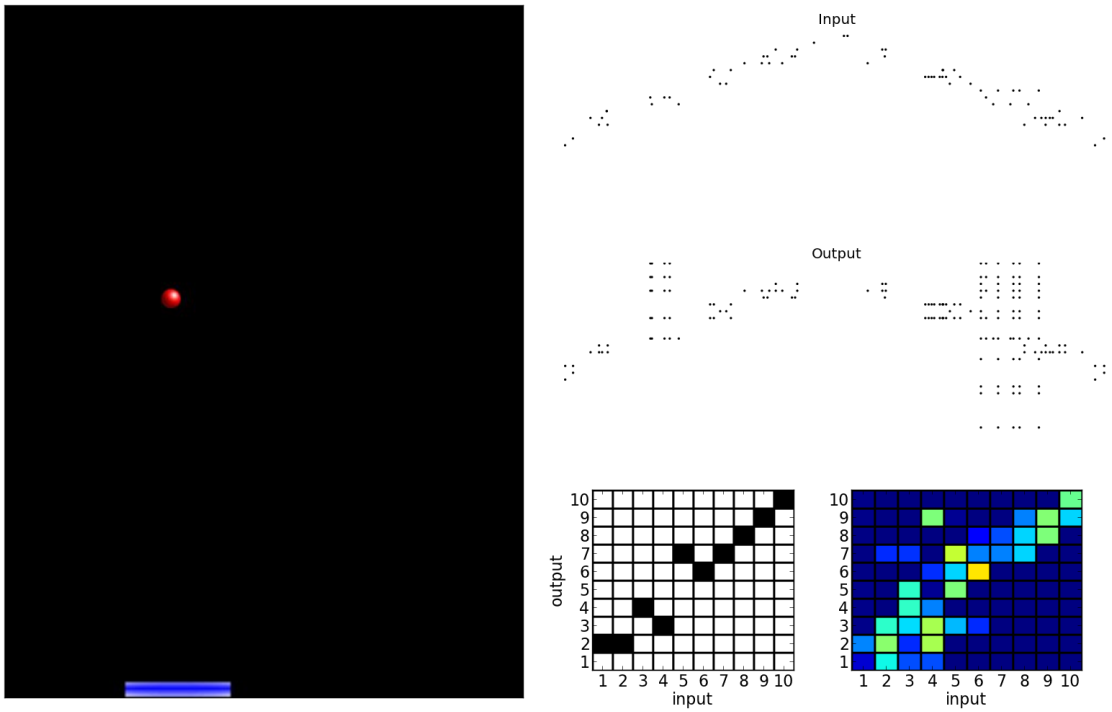



**Figure 8.13:** Example of the excitatory network learning a one-to-one rule for pong.

When immersed in the pong virtual environment, the excitatory only network learns the correct input/output pairs to play the game, Figure 8.14. In this case the game requires that the network move the paddle to the channel that matches the location of the puck, a one-to-one association. Figure 8.13 illustrates the network learning this. The rules of the game can then be changed, as they were in the learning example above. The network now has to associate the location of the puck with the opposite paddle location, a one-to-ten association, Figure 8.15.

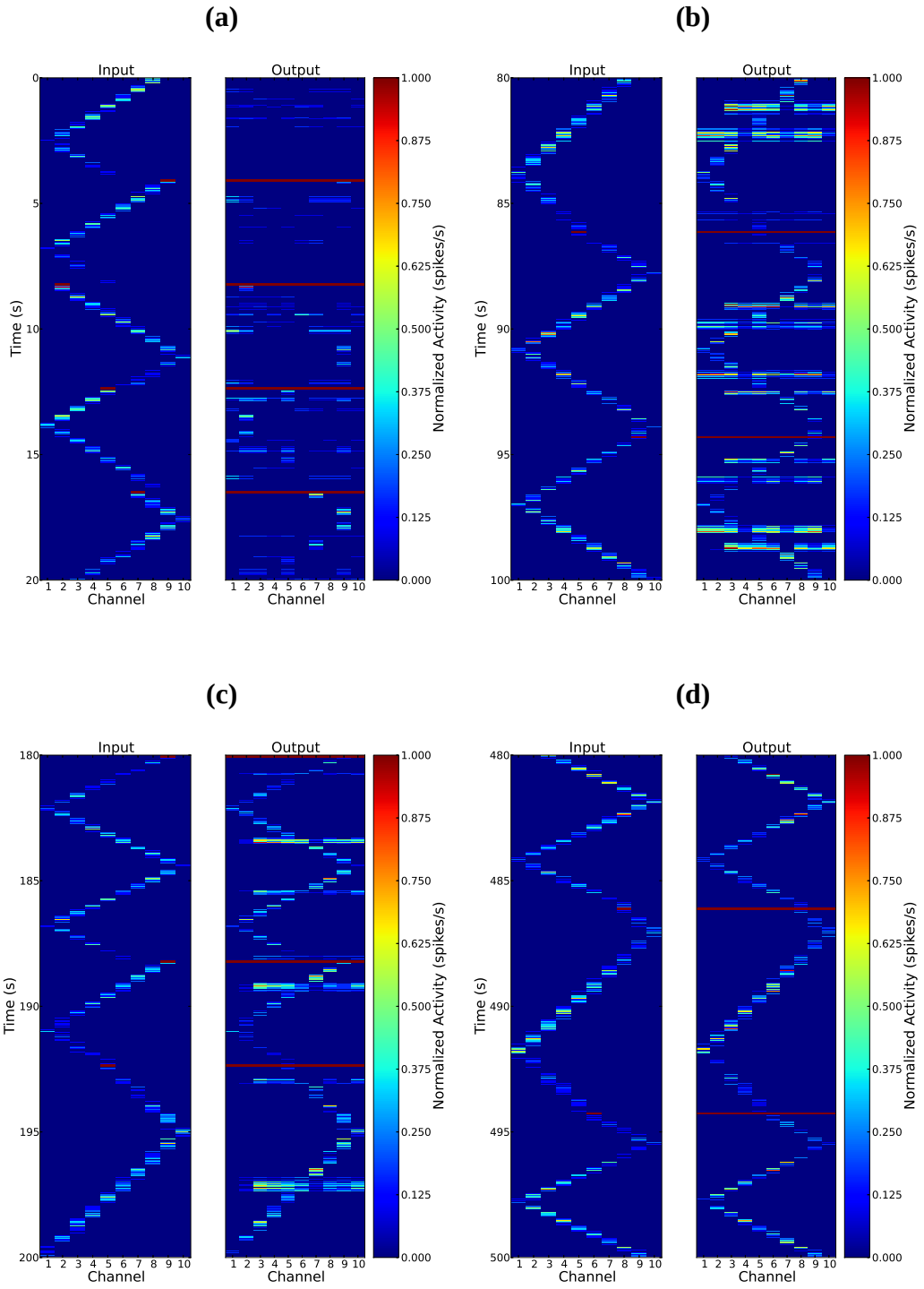



**Figure 8.14:** Excitatory network pong game play. The input signal corresponds to the position of the puck. Notice that the time course runs along the y-axis from top to bottom. The x-axis is the corresponding channel. (a) $0 - 20$ seconds. The network has minimal responds. (b) $80 - 100$ seconds. As the network receives feedback from game the required input/output pairs begin to form. (c) $180 - 200$ seconds. The pair correlations become more defined. (d) $480 - 500$ seconds the pairs are completely defined and the output directly follows the input.

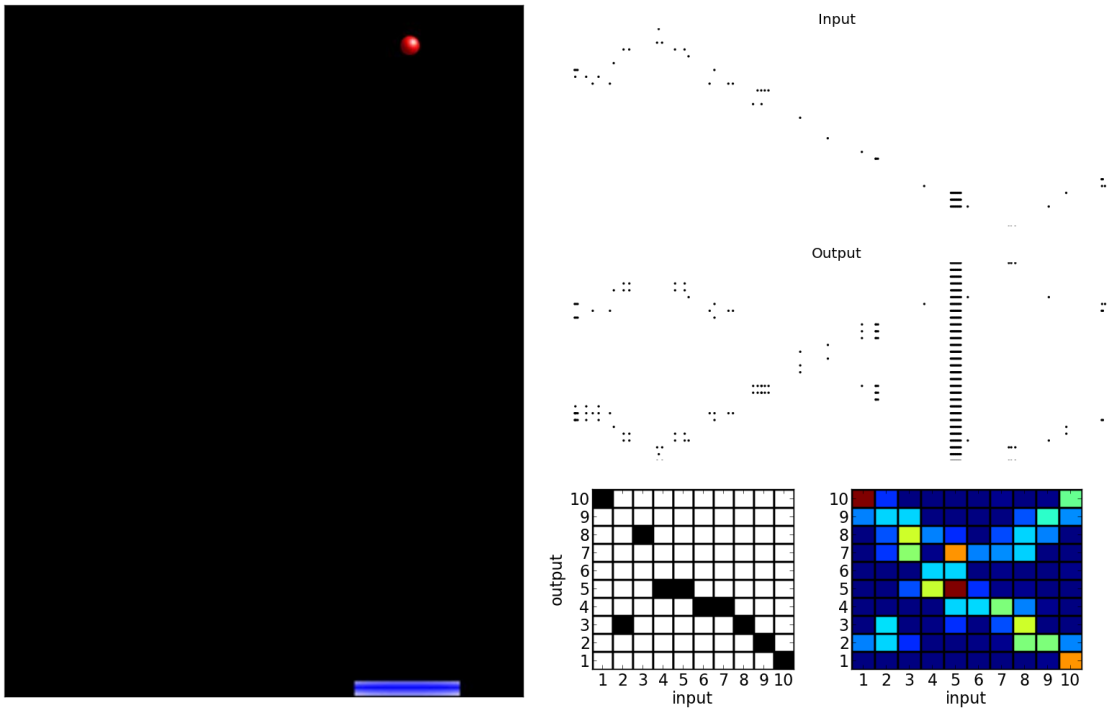



**Figure 8.15:** Example of the excitatory network learning a new set of pong rules. This time a one-to-ten input/output rule is required.

### Pong performance analysis

There are a number of factors that determine how well the network performs in the game task. The first is the spatial width of the Gaussian stimulus curve, $c$. This affects the overlap between channels, the larger the value of $c$ the larger the overlap between channels. We use three spatial widths, $0.025$, $0.035$, $0.045$. The next factor is the peak of the Gaussian stimulus curve. The larger the value the more active the input channels become. Two input peaks, $R_{max}$, are used, $10$ Hz and $40$ Hz. Finally, the length of reward is an important factor. This determines how long a stimulus lasts for. Two values are chosen for this analysis, $300$ ms and $500$ ms.

For each combination of these parameters, $5$ simulations of $500$ seconds were run. The accuracy, $(saves/opportunities) \cdot 100$, is computed for $25$ second blocks. The average and

standard deviation of these blocks is plotted below.

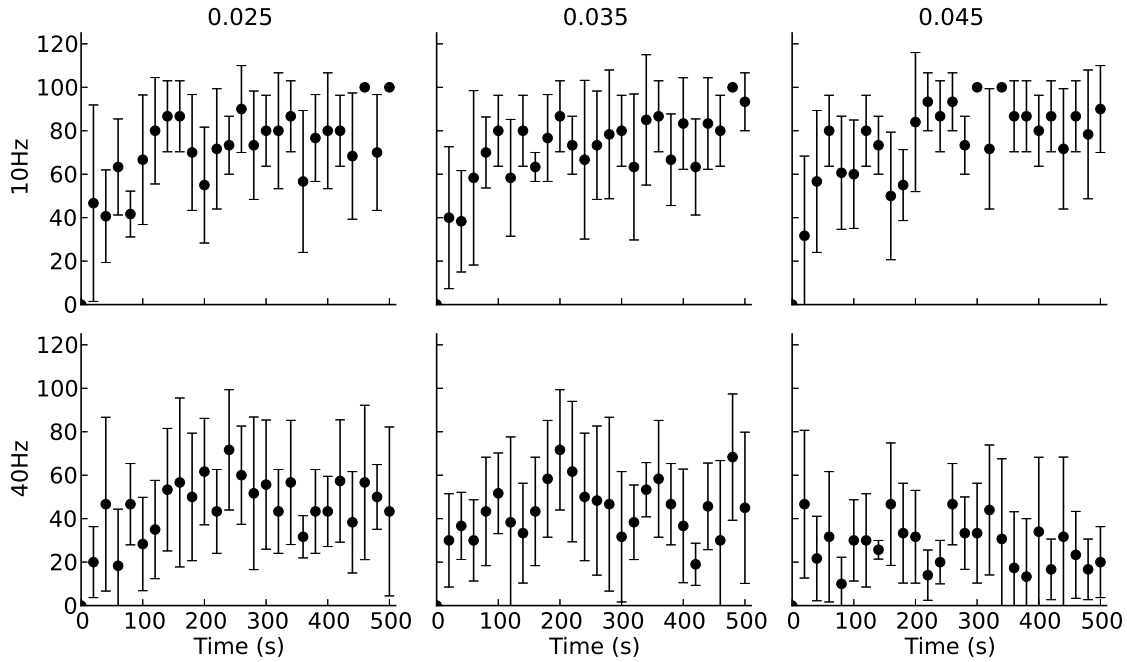



**Figure 8.16:** Excitatory network pong performance for 300 ms reward. The y-axis is the accuracy of the network. A value of 100 means the network blocked all of the pucks in that 25 second block. The column titles correspond to the spatial width of the input stimulus. The row labels indicate the Gaussian peak value.

Figure 8.16 presents the results when the reward is 300 ms long. The network struggles to perform when the input stimulus is too high (40 Hz results). For the lower activity stimulus this network has very similar performance throughout the different spatial widths.

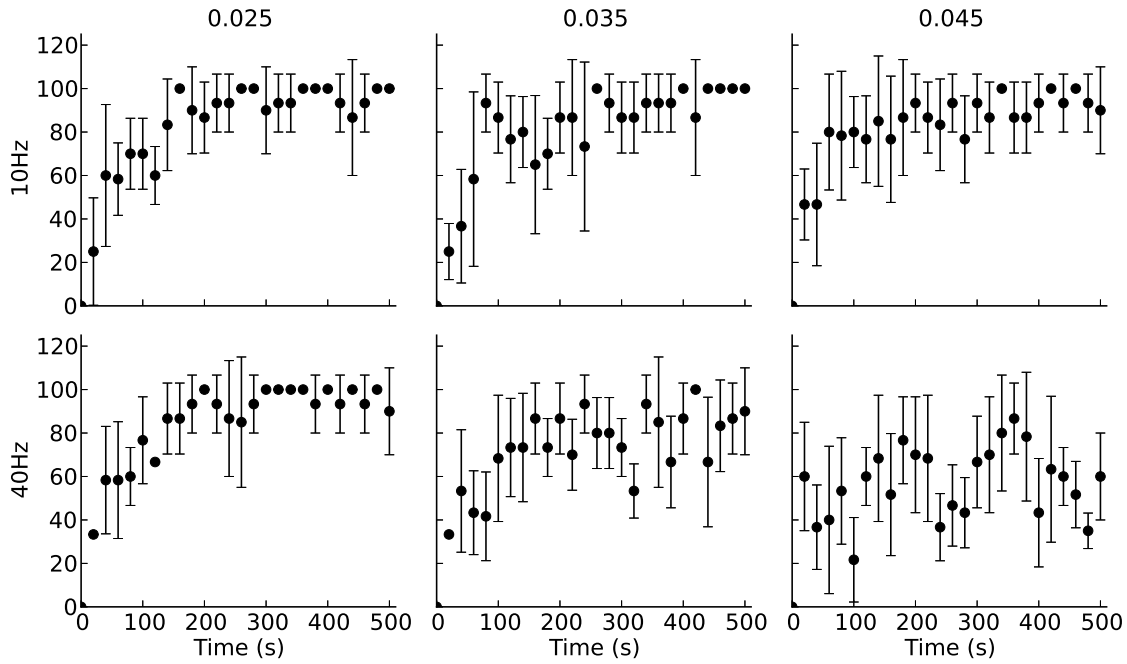



**Figure 8.17:** Excitatory network pong performance for $500$ ms reward. The y-axis is the accuracy of the network. The column titles correspond to the spatial width of the input stimulus. The row labels indicate the Gaussian peak value.

This network benefits most from a longer reward period. Not only does the variability in the $10$ Hz peak input results go down but the overall performance is increased throughout the spatial widths. As the peak input stimulus is raised, the results for $c = 0.025$ are comparable to the lower input stimulus. However, this performance is lost as the spatial width is increased.

### 8.3.2 Lateral inhibition network

#### Basal activity

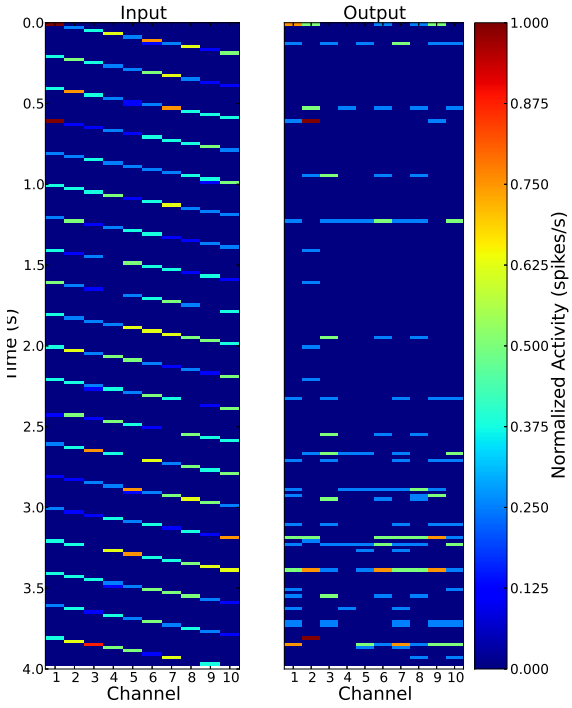



**Figure 8.18:** Basal activity of the LI network. Notice that the time is moving from the top of the chart down.

As with the excitatory only network there is no bias between channels in the lateral-inhibition (LI) network. The basal activity of a sample network is presented in Figure 8.18, with the rasters and synaptic weights presented in Figure 8.19. The interneurons suppress the overall activity of the output populations resulting in lower rates and more control than the excitatory only network. This directly affects the changes in the output weights, which are much smaller than in the excitatory case.

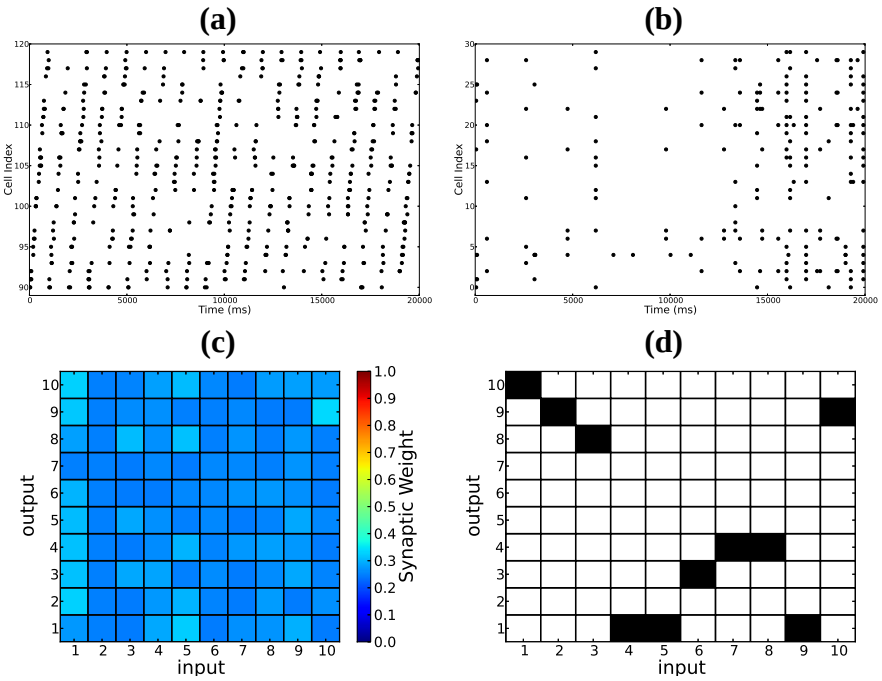



**Figure 8.19:** Results of 20s of Basal level activity for the lateral inhibition network. (a) Input raster. (b) Output raster. (c) Average synaptic weights. (d) Location of maximum average synaptic weights.

## Learning capabilities

Similar to the excitatory only case, Figure 8.20 demonstrates the LI network's ability to learn different channel associations. The stages match those discussed in Section 8.3.1.

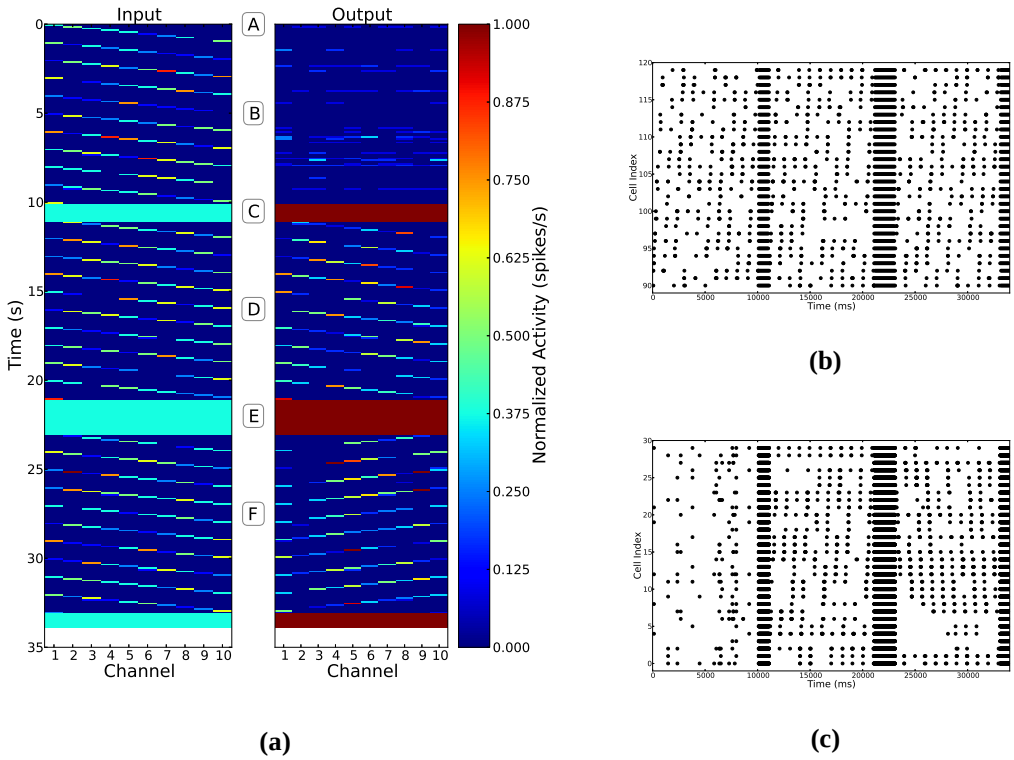



**Figure 8.20:** Example lateral inhibition network reward-learning scenario. (a) Activity rate map of the example scenario. Activity was calculated using a moving Gaussian weighted window. (b) Spike raster of the input populations. (c) Spike raster of the output populations.

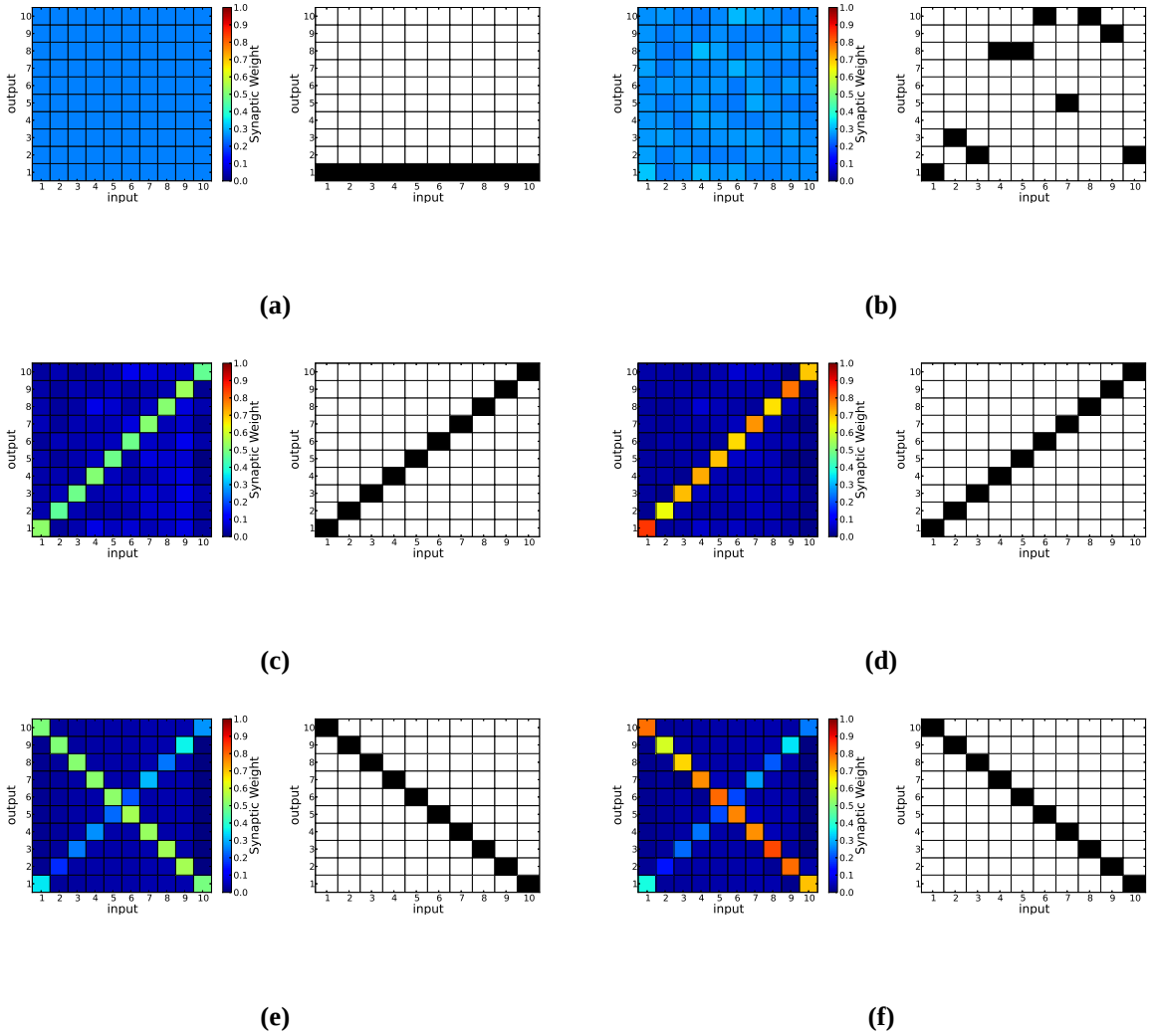



**Figure 8.21:** Average and maximum synaptic weights between input/output pairs after learning corresponding to Figure 8.20. (a) 0 sec (b) 10 sec (c) 11 sec (d) 21 sec (e) 22 sec (f) 33 sec

**Pong environment**

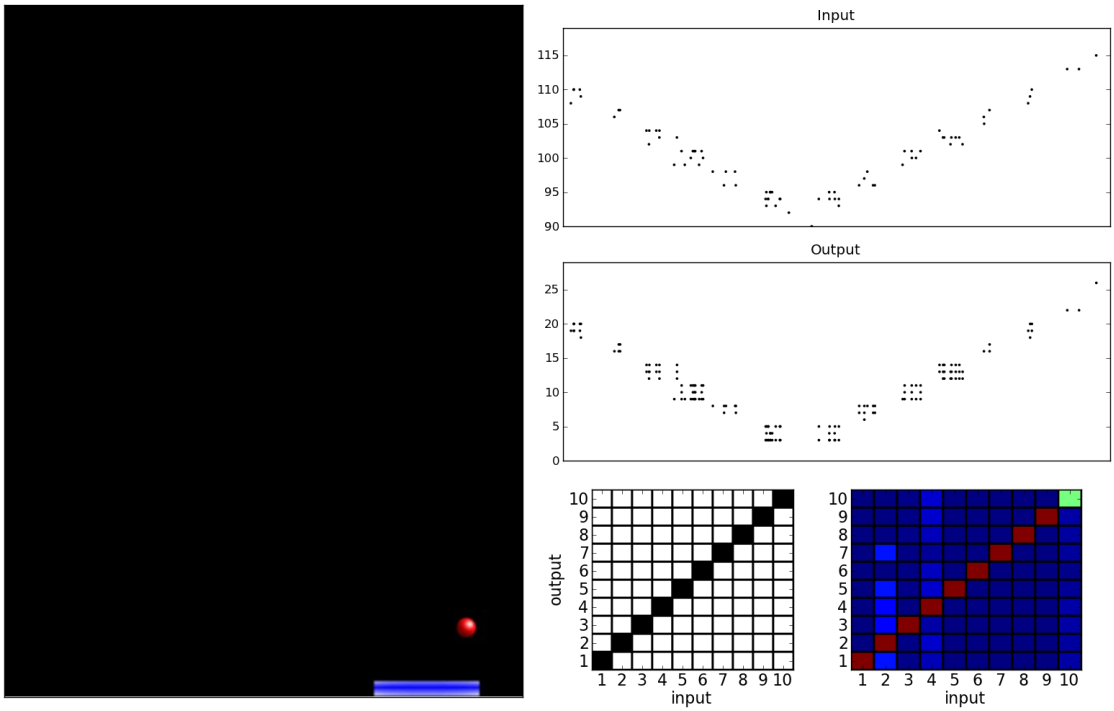



**Figure 8.22:** Lateral-inhibition network learning a one-to-one rule for pong.

Comparing the example game playing of the excitatory network in Figure 8.14, with the LI network in Figure 8.22, reveals the stability the interneurons provide. The output populations begins to track the input populations much earlier in the experiments. Similarly, Figure 8.22 demonstrates a clear diagonal in the synaptic weights and the new set of associations is learned with clear stability, Figure 8.24.

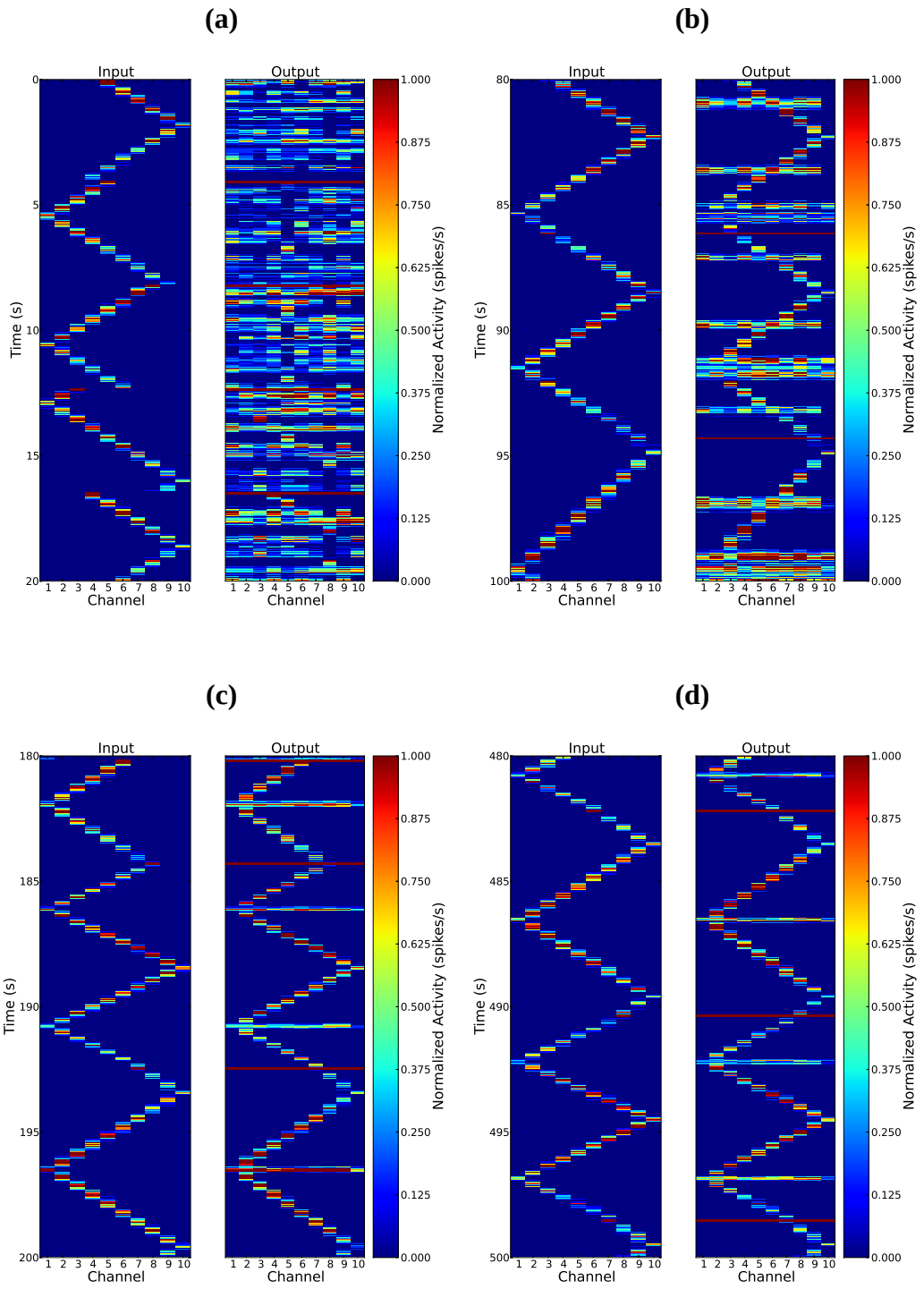



**Figure 8.23:** LI network pong game play. The input signal corresponds to the position of the puck. The time course runs along the y-axis from top to bottom. (a) $0 - 20$ seconds. (b) $80 - 100$ seconds. (c) $180 - 200$ seconds. (d) $480 - 500$ seconds.

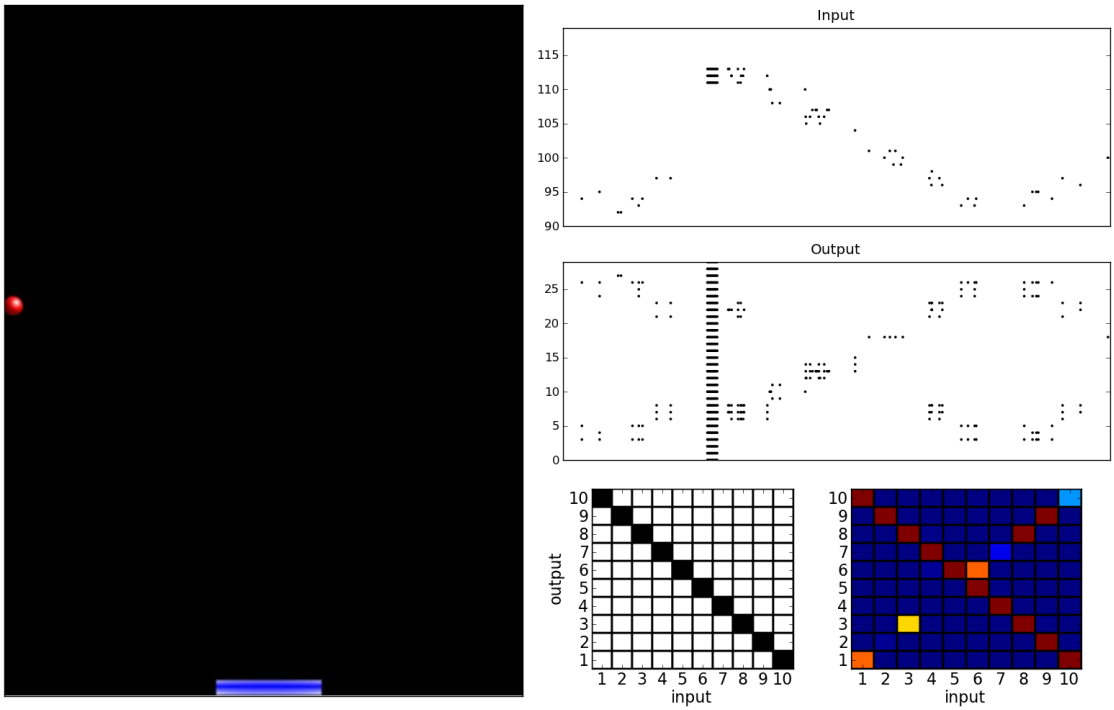



**Figure 8.24:** Example of the LI network learning a new set of pong rules. This time a one-to-ten input/output rule is enforced.

**Pong performance**

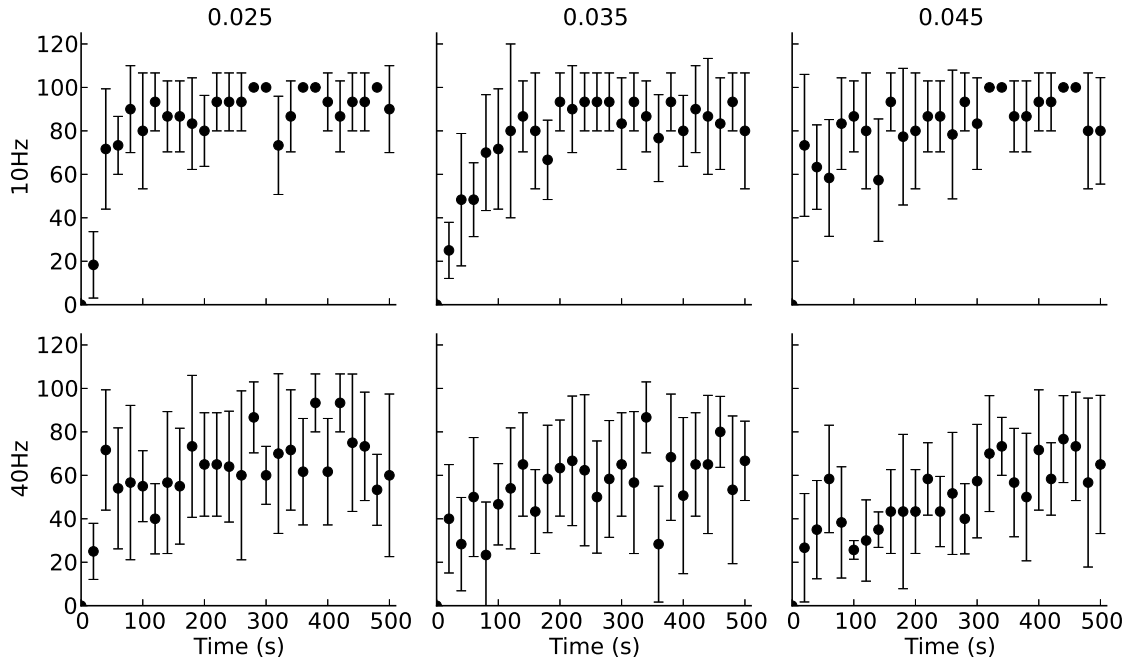



**Figure 8.25:** LI network pong performance for 300 ms reward. The y-axis is the accuracy of the network. The column titles correspond to the spatial width of the input stimulus. The row labels indicate the Gaussian peak value.

Figure 8.25 presents the results when the reward is 300 ms long. For the 10 Hz stimulus the network performs considerably better than the excitatory network results of Figure 8.16. The variability in the standard deviation is much lower and the overall performance is higher. However, when the peak input stimulus is higher the performance drops considerably. Although peak accuracy is slightly higher than the excitatory network, the standard deviations are larger in both cases.

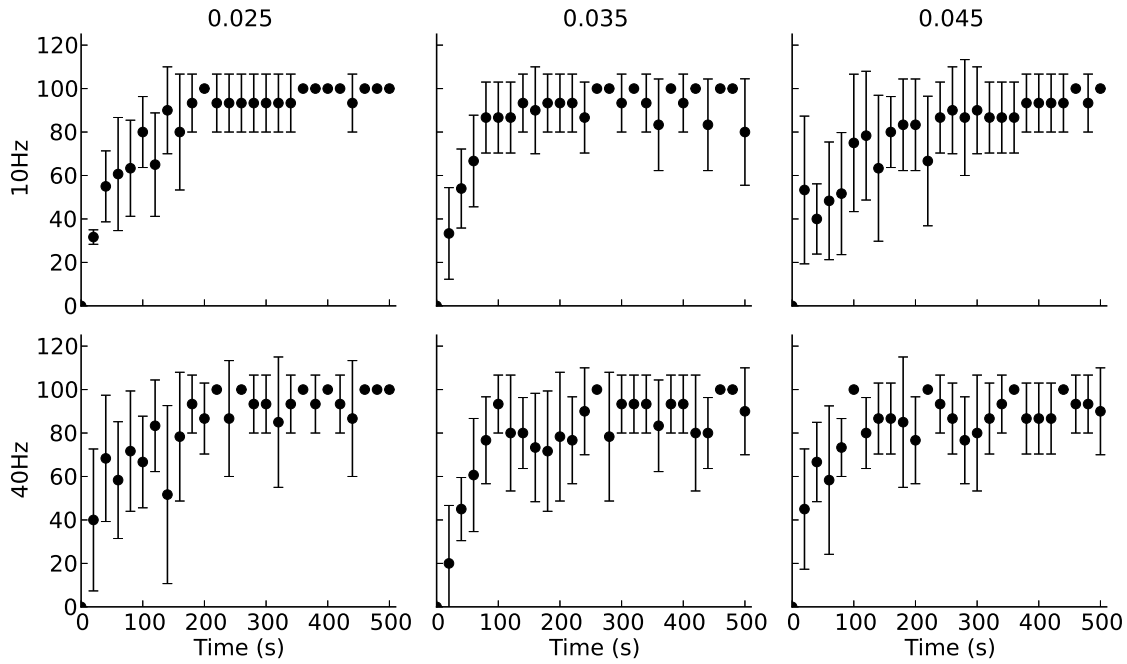



**Figure 8.26:** LI pong performance: 500 ms reward

When the reward time is increased to $500$ ms the overall performance throughout the parameter space is surprisingly consistent. The slopes in the accuracy curves are slightly different but all approach an accuracy of $100\%$ with relatively small standard deviation.

**Excitatory network vs. LI network**

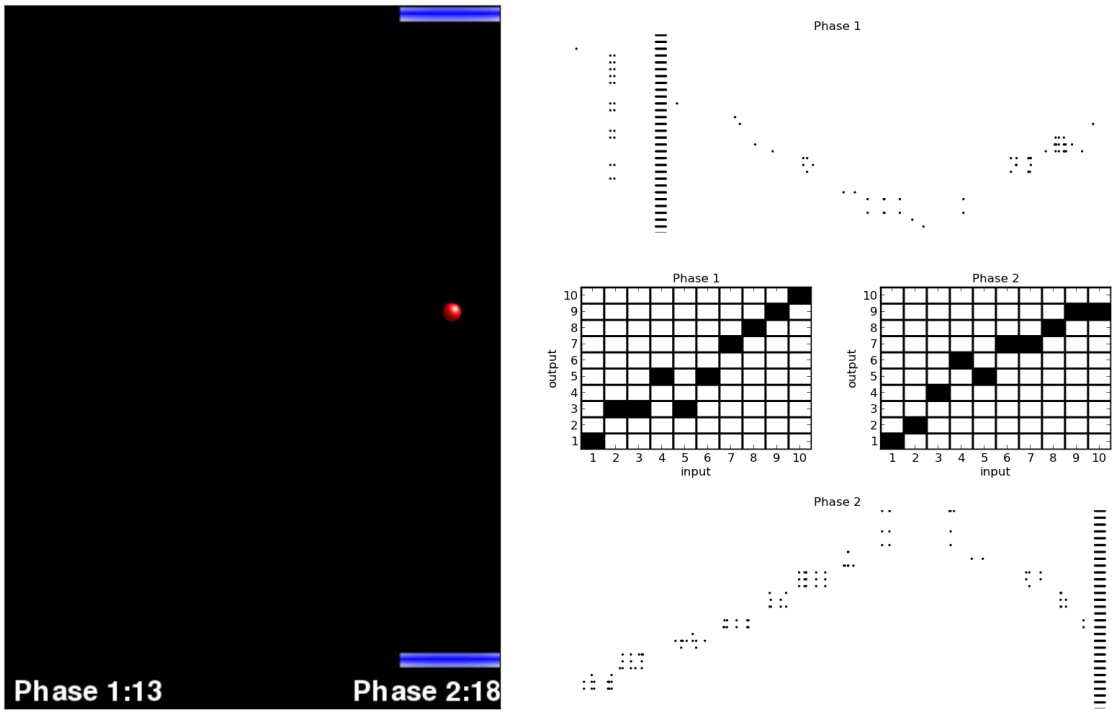



**Figure 8.27:** Excitatory network (Phase 1) vs. Lateral-inhibition network (Phase 2) in pong.

These results illustrate how the addition of inhibitory interneurons provide a mechanism for channels to actively suppress the surrounding channels. This creates a more stable configuration at the relatively low-cost of thirty extra neurons. As a toy example of this, the excitatory network is set against the LI network in pong, Figure 8.27. As would be expected, the LI network consistently beats the excitatory network. However, this paradigm does create an interesting test-bed for future network comparisons.

### 8.3.3 Basal ganglia direct pathway

**Basal activity**

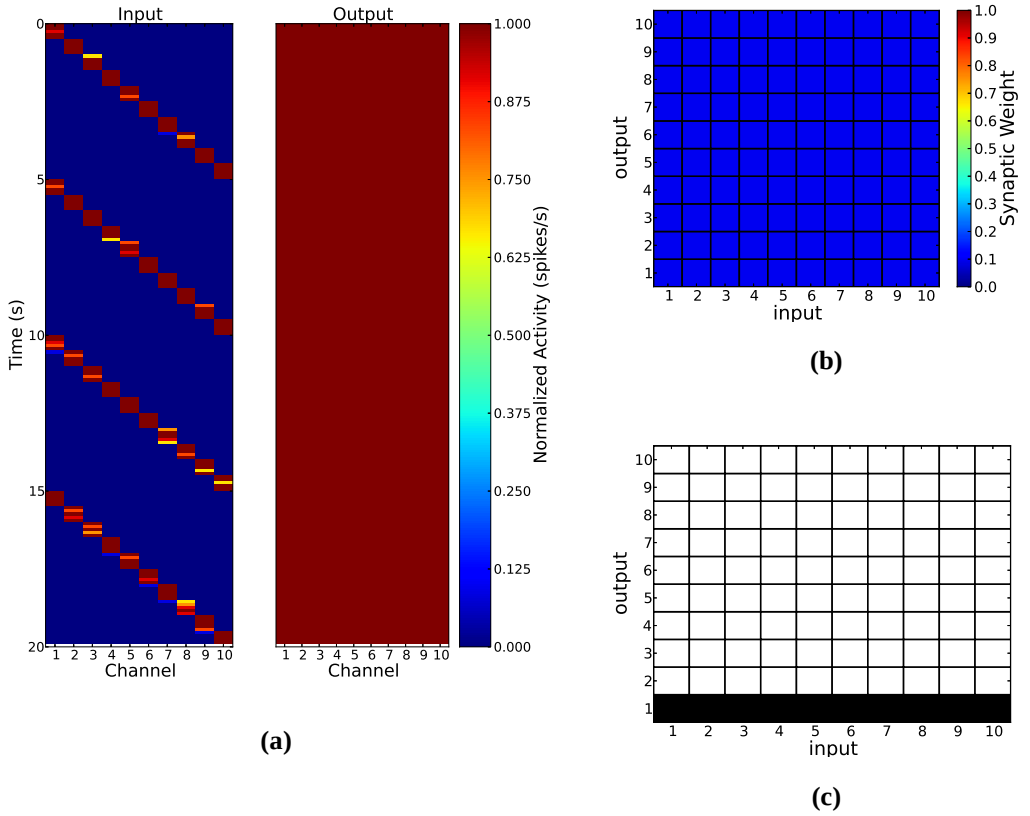



**Figure 8.28:** Basal activity of the BG Direct network. (a) (b) (c)

The basal output of the BG direct pathway is a tonic 30 Hz firing from the SNr inhibitory neurons. A channel is activated through disinhibition facilitated by the Str neurons. This basal activity is illustrated in Figure 8.28.

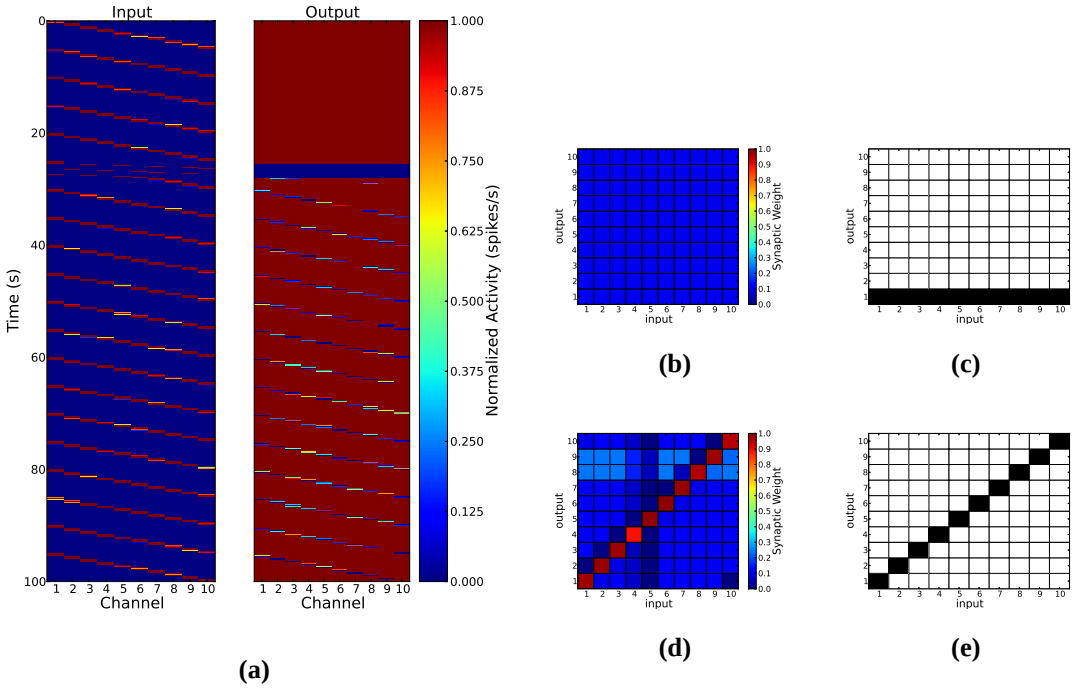



**Figure 8.29:** learning in the BG Direct network. (a) IO (b) Initial weight plots (c) Initial location of maximum average synaptic weight. (d) Average synaptic weights after learning. (e) Location maximum average synaptic weight after learning.

The ability of the BG direct pathway to learn a particular set of input/output pairs is similar to the other models, Figure 8.29. It should be noted that the network is capable of relearning but the time course is longer than in the Excitatory and LI networks.

## Pong environment

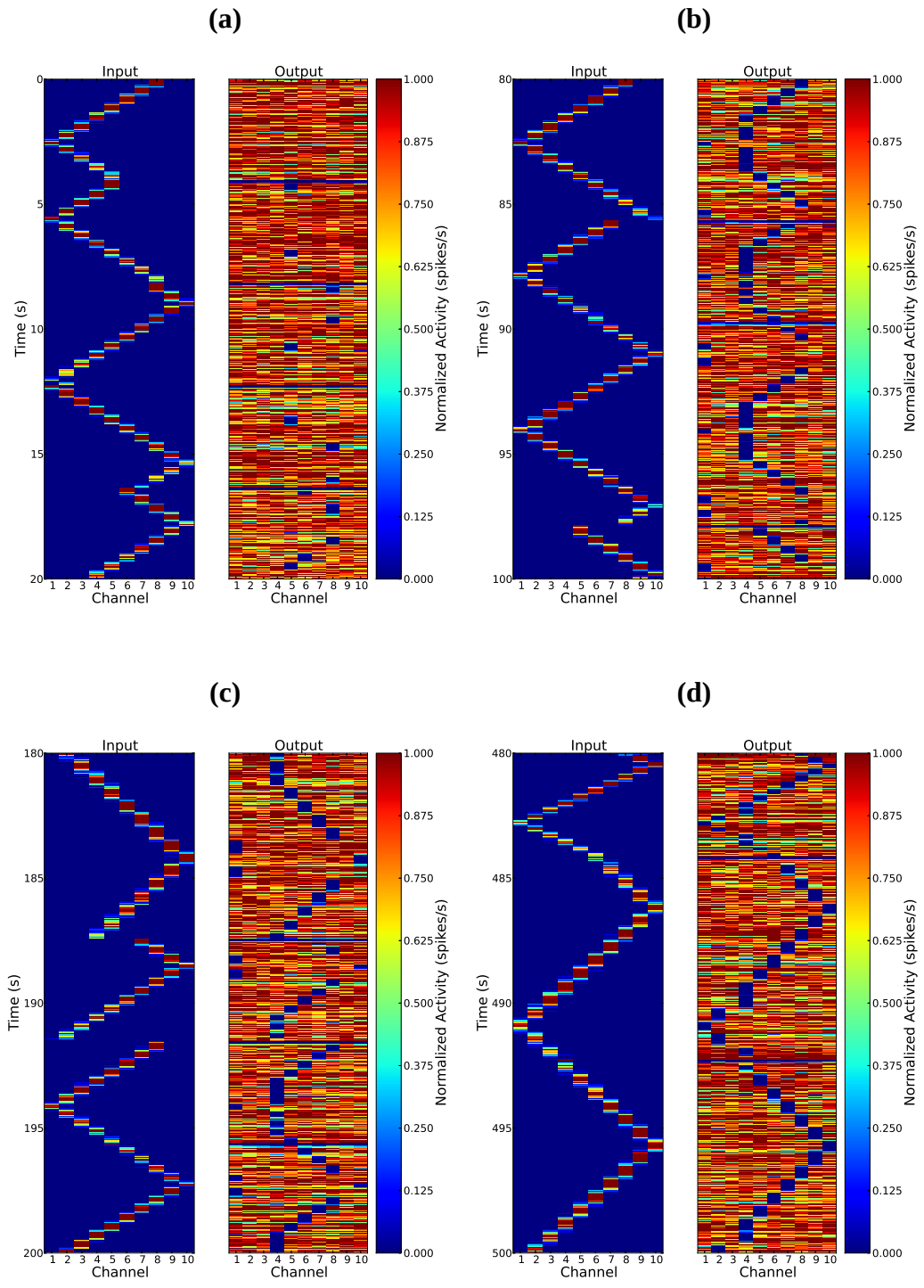



**Figure 8.30:** BG Direct network pong game play. The input signal corresponds to the position of the puck. The time course runs along the y-axis from top to bottom. (a) $0 - 20$ seconds. (b) $80 - 100$ seconds. (c) $180 - 200$ seconds. (d) $480 - 500$ seconds.

When immersed in the pong environment the BG direct network is capable of tracking the input stimulus, Figure 8.30.

**Pong performance**

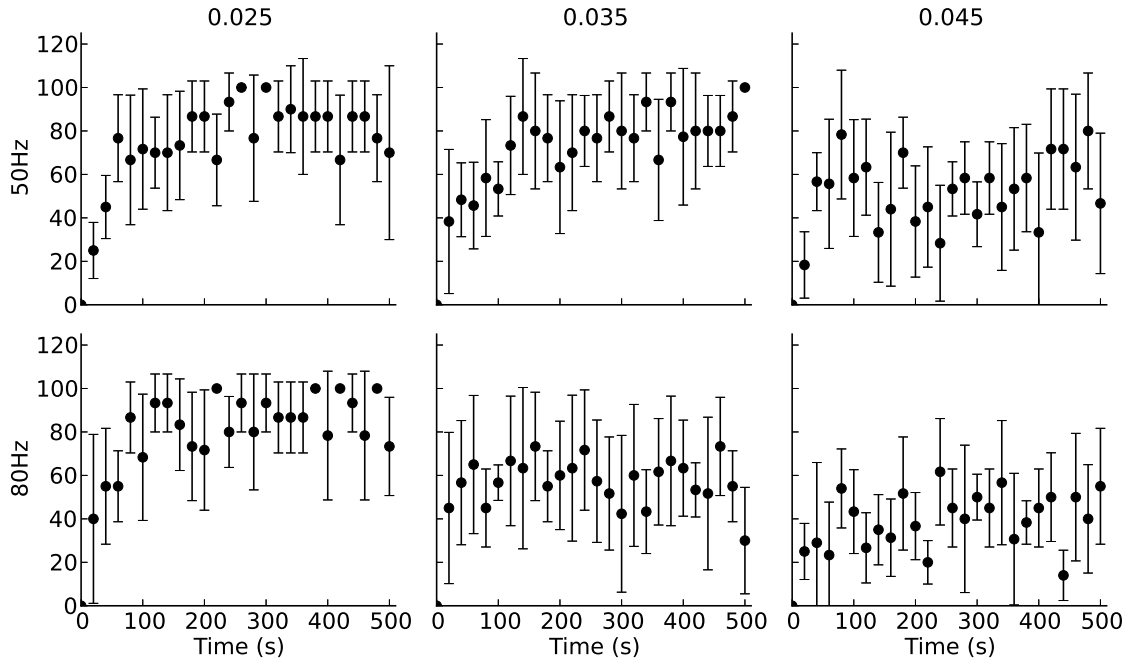



**Figure 8.31:** BG Direct pong performance for 300 ms reward. The y-axis is the accuracy of the network. The column titles correspond to the spatial width of the input stimulus. The row labels indicate the Gaussian peak value.

The peak value of the input stimulus for the BG direct network needs to be higher than the other networks to sufficiently activate the desired channel. The low rate is set to 50 Hz and the high is set to 80 Hz. With a 300 ms reward period the BG direct network performs well for small spatial widths, Figure 8.31. As the overlap is increased however, the performance degrades quickly.

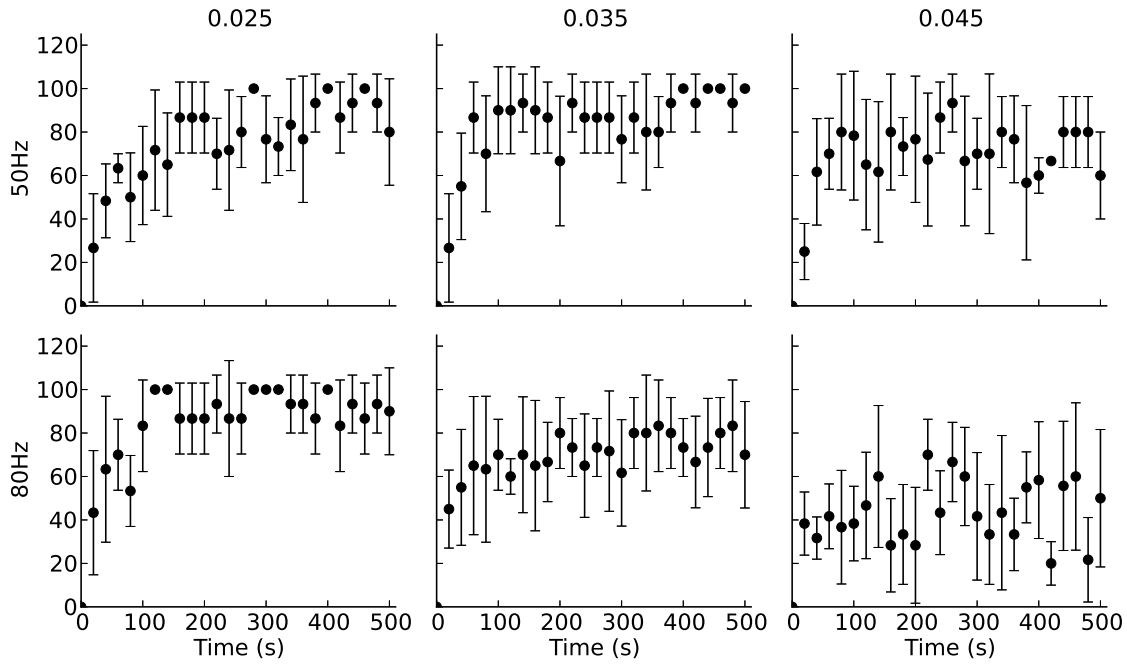



**Figure 8.32:** BG Direct pong performance: 500 ms reward

Increasing the reward period to $500$ ms improves the performance of the network for the first two spatial widths, Figure 8.32. When the spatial width is $0.045$ the performance is improved but it is still not as high as the LI network.

### 8.3.4 First-person shooter

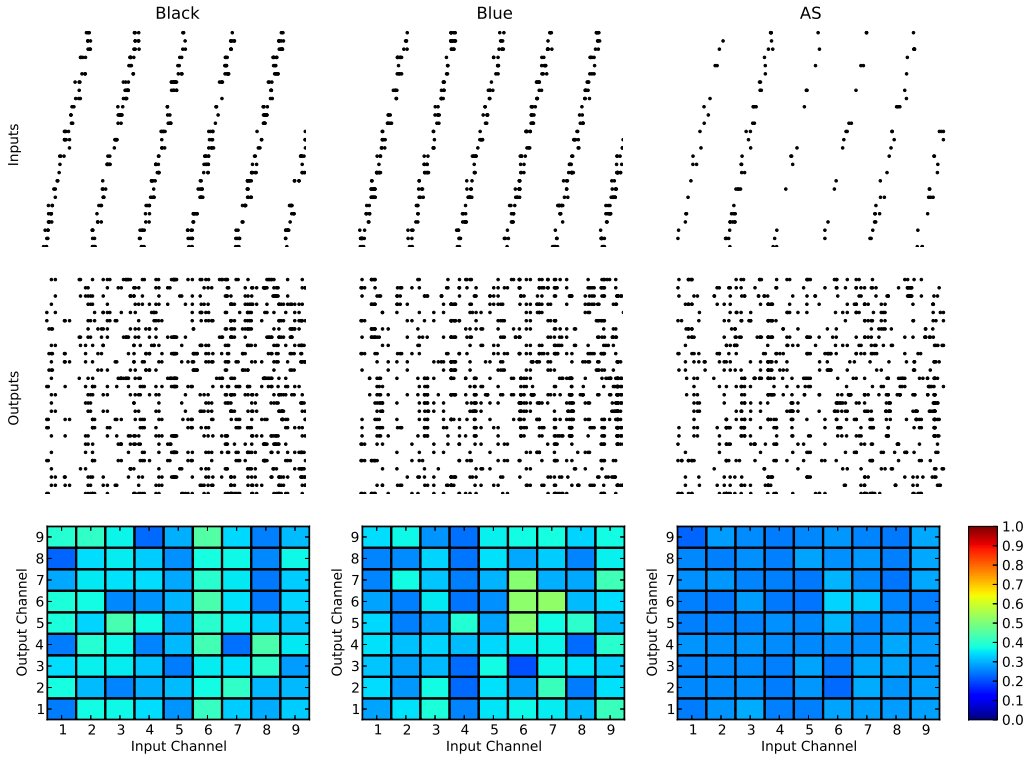



**Figure 8.33:** Basal activity of FPS network.

The combination of three LI networks allows for more complex decision making. The networks can learn to weight different classes of input information based on reward feedback. The basal activity of FPS network is presented in Figure 8.33. Each of the subnetworks has $9$ channels, with the Black and Blue subnetworks both feeding into the action selection (AS) subnetwork. The AS subnetwork also receives saliency information from the environment.

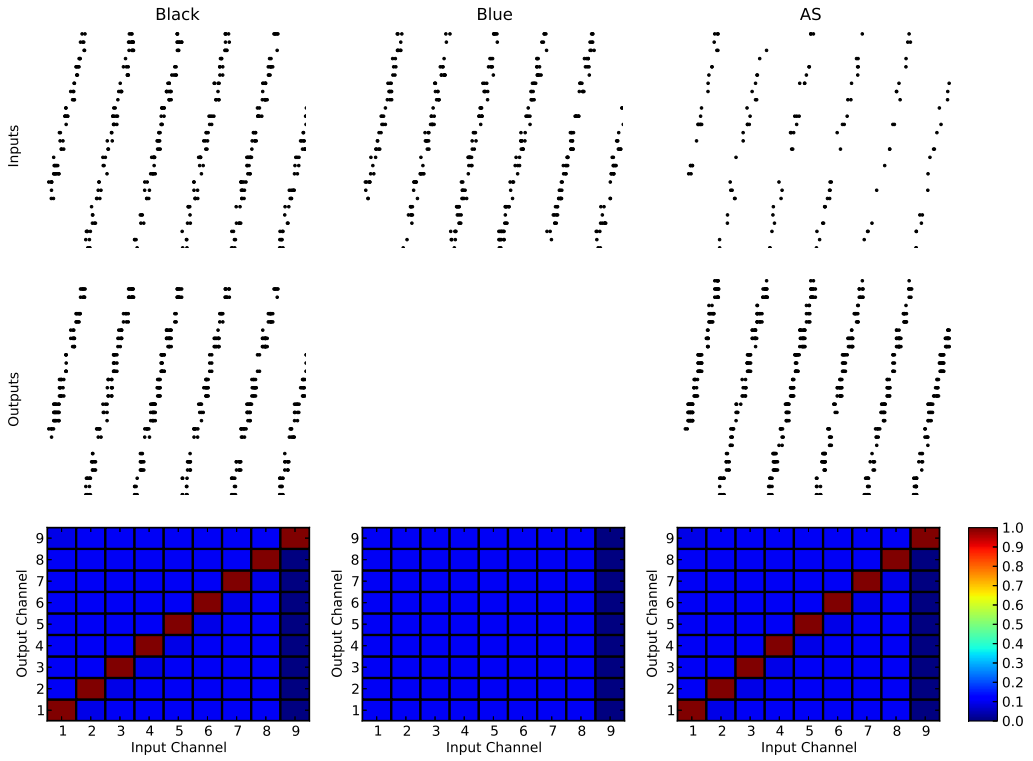



**Figure 8.34:** FPS network learning capabilities.

Using the same stereotyped reward mechanisms, the FPS network can be trained to perform more complex action selection tasks, Figure 8.34. In this case the Black and AS subnetworks have learned a one-to-one correlation, while the Blue subnetwork has been effectively disconnected. The result is that the saliency information alone is not enough to cause the AS network to cross the selection threshold. A complementary input is required from one of the other subnetworks, in this instance only a black game element can contribute, Figure 8.35. The resulting network learns to ignore the innocuous blue elements while focusing on the dangerous Black ones.

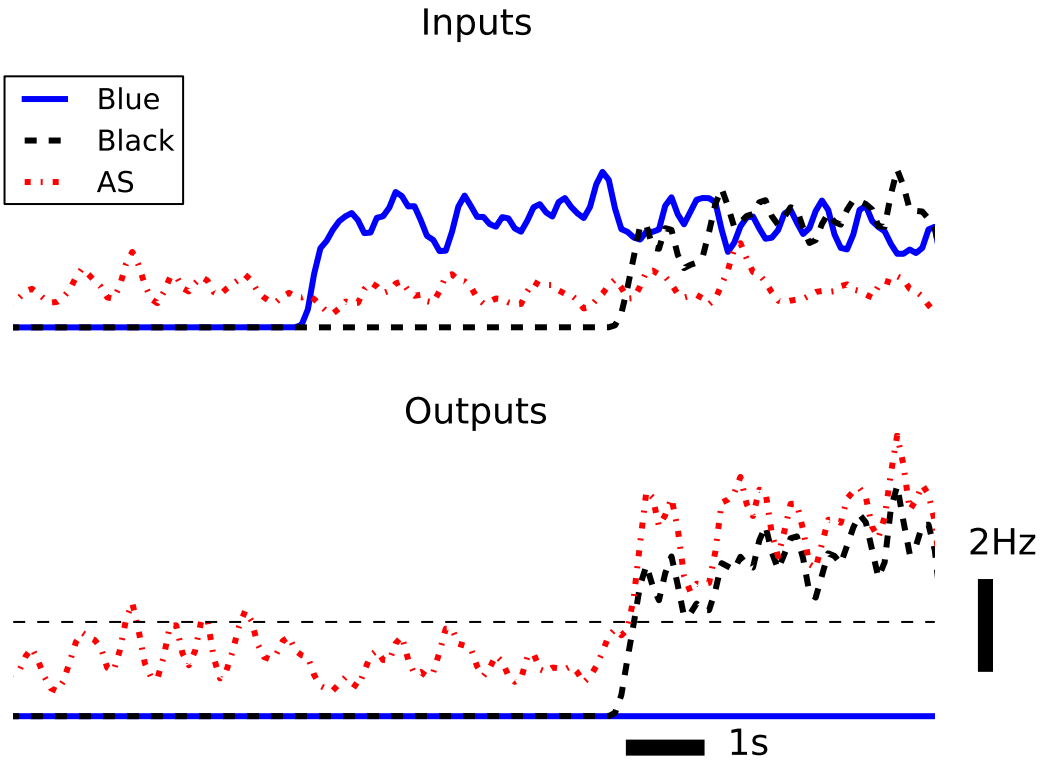



**Figure 8.35:** FPS single channel activity after training. The saliency input alone is not enough to push the AS subnetwork above the selection threshold (dashed gray line). The addition of a blue stimulus is ignored and thus does not contribute to the AS subnetwork activity. When a black element stimulus is added the activity of the AS subnetwork is driven passed the selection limit and that channel is selected.

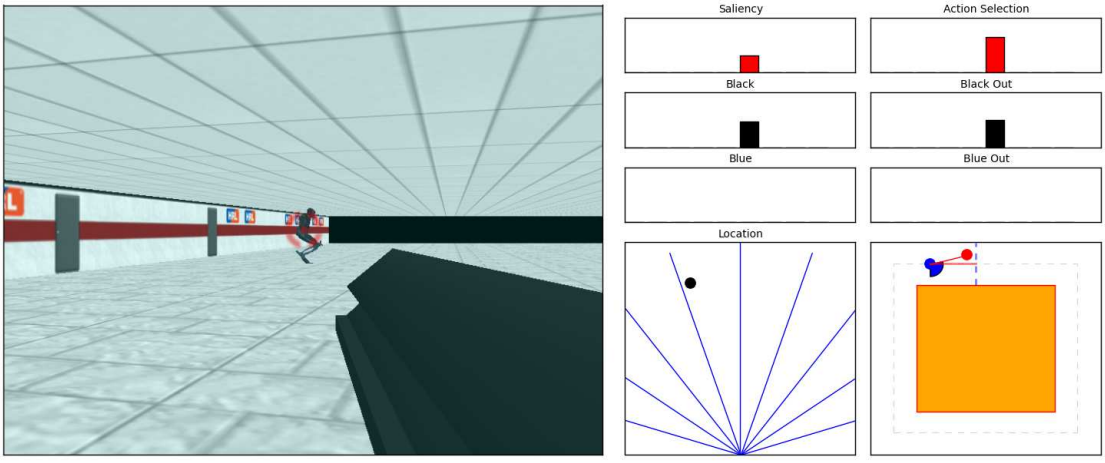



**Figure 8.36:** FPS Game Play. The network appropriately targets and shoots the enemy player.

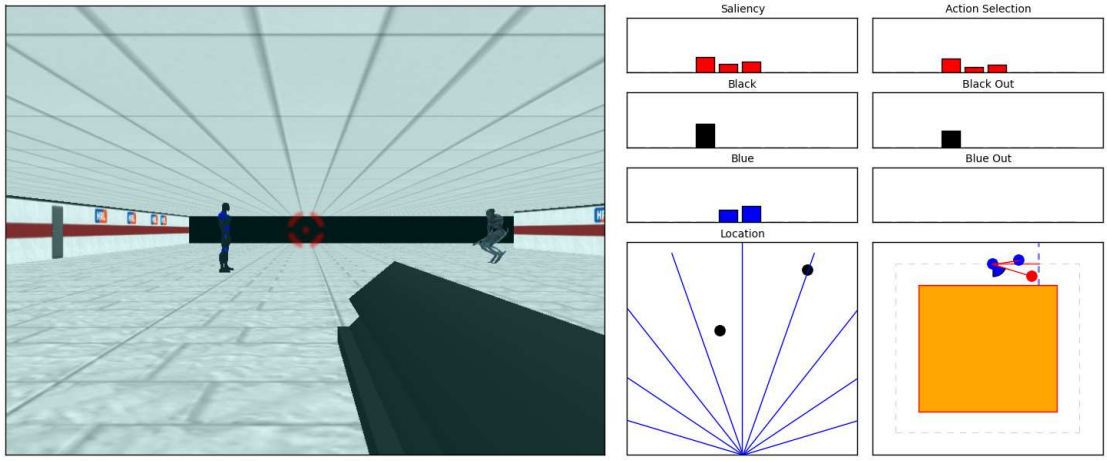



**Figure 8.37:** FPS Game Play. Although the blue character is more salient in the player's field of vision the network appropriately targets and shoots the enemy player.

When placed in the Neuralstein environment the network can move through the environment and target enemies when in view, Figure 8.36. In addition, when presented with both types of game elements, the network can appropriately select the black element, even when the blue one is closer to the player, Figure 8.37.

## 8.4 Discussion

### 8.4.1 Similar Work

Wiles et al. (2010) developed a spiking neural model to control a rat animate performing phototaxis. The network was constructed to perform the task similar to a Braitenberg vehicle. Burgsteiner (2006) created a liquid state machine using a recurrent network with fixed internal synapses and plastic output synapses that learned a similar task.

The model of Arena et al. (2009) consisted of three layers of Izhikevich neurons to control a virtual robot with several sensory modalities. The networks were constructed with an initial understanding of how to process low-level sensor input such as proximity

and contact sensors as well as visual cues. These were used to direct the robot through the environment. Simultaneously, the network learns to perform this navigation using a range-finding sensors. The inherent low-level sensors basically train the network on how to respond to the high-level sensors.

Florian (2006) evolved a fully recurrent spiking neural network to control a simple virtual agent to seek out, push and the release balls in its environment. An evolutionary algorithm was used to calculate the synaptic weights of the network to accomplish the task.

Barr et al. (2007) implemented a mode of the basal ganglia on a neural processor array. Although not directly demonstrated in the hardware presentation the original software neural model was capable of performing action selection. However, there are no inherent mechanisms for reinforcement-learning and the micro-channels of the basal ganglia were predefined by the network.

Merolla et al. (2011) presented a neuromorphic processor capable of playing a game of pong against a human opponent. This description was later extended by (Arthur et al., 2012). The network was constructed off-line and once programmed on the hardware remained static. In that, a neural network, consisting of $224$ neurons, that could also play a pong style game was created. The network was constructed off-line and was demonstrated on a neuromorphic processing core. The training of the network involved teaching the network to predict different patterns of motion by the puck. Rather than simply tracking it, like the networks here, the model would plan where the paddle must be placed. The resulting networks however, are specialized for that task and can not adapt to changing environments once embodied in hardware.

### 8.4.2 Playing games

#### Pong

The learning of channel associations is somewhat arbitrary in the examples presented here. The correlation between input and output populations can in fact be engineered to have more complex relationships than a simple pair. As illustrated by the FPS network results, other combinations can be created as well as mechanisms for more intricate information processing as well.

The tracking of the puck in the pong networks is reactive, with movements made based on the current position in the game. In the future this concept will be extended to include predictive control of the paddle. A recurrent network capable of learning these kinds of associations could be included along side the reactive networks presented here to achieve this.

Initially all of the weights would be random. Through the feedback mechanisms demonstrated here the reactive networks can be trained to track the position of the puck. This learned behavior can then be used as an training signal to the predictive networks.

#### Neuralstein

First-person shooters have been extremely popular in Artificial Intelligence (AI) research. The complex interactions between the environment, game elements and multiple players, challenge non-player controllers in unique ways. This popularity has even led to competitions, such as the Botprize, where the goal is to create the most “human like” AI controller (BotPrize, 2012).

Due to the different strategies required to successfully play a modern FPS, traditional AI domains have dominated (van Hoorn et al., 2009; Schrum and Miikkulainen, 2010). It

is the complexity of the task that makes it attractive to embodied modeling. The approach taken here relies on abstracting some of that complexity away. As the networks become more capable other aspects of the FPS paradigm can be added.

### 8.4.3 Future work

These simple feed-forward networks are a satisfactory start to employing the SyNAPSE neuromorphic architecture in embodied modeling. Alone, they can be utilized as configurable controllers but their real potential lies in their use as building blocks in more complex control systems. We have already demonstrated how these can be connected together in a simple configuration but in the future these will be combined with more sophisticated networks. For example, recurrent networks can provide, through feedback, state information of the system. This basic form of short-term memory can process the temporal aspects of a system's inputs and allow for more intelligent processing.

Although the performance of the BG direct pathway is slightly lower than the LI network for the tasks presented here, it is still an extremely useful building-block for future models. Physiologically the mammalian basal ganglia achieves action-gating by removing its inhibitory influence on thalamocortical relay neurons. This allows information from higher-cortical areas to pass through the thalamus to other brain areas. This type of action-gating is replicated by the BG model presented here and can perform a similar function in larger neural models.

Finally, the feedback for these networks was dependent on conditioned input stimulus to the reward modulation populations. The games played the role of the critic. In the future, more sophisticated reward and punishment signals, such as those in Florian (2007) and Friedrich et al. (2011), will be implemented to find a generic reward critic and more efficient controllers.

# Chapter 9

# Models of The Basal Ganglia For Neuromorphic Hardware

The basal ganglia plays a central role in many associative, motor and limbic functions. In addition, it is an important clinical target in several neurological disorders. While its functional significance is a topic of ongoing research, our current understanding has facilitated the creation of computational models that have contributed novel theories, explored new functional anatomy and demonstrated results complimenting physiological experiments. The utility of these models however, extends beyond these applications. In particular in neuromorphic engineering, where the basal ganglia's functional role in computation is important for applications such as power efficient autonomous agents and model-based control therapies. The neurons used in existing computational models of the basal ganglia however, are not amenable for many low-power hardware implementations.

In this chapter we explore some of these networks using the simple Izhikevich hybrid neuron (Izhikevich, 2003). Capable of replicating many of the known dynamics of the basal ganglia nuclei, hybrid neurons are computationally efficient compared to con-

ductance based counterparts. Four published models, spanning single neuron and small networks of the basal ganglia are created using the simple hybrid neuron. These models successfully replicated the results of the original works, providing validation for the hybrid neuron in biologically faithful models of the basal ganglia and creating a foundation for future neuromorphic hardware implementations.

## 9.1 Introduction

Computational models of the BG have proved useful in many aspects of neuroscience; including developing novel theories of Parkinson's disease and deep brain stimulation (Rubin and Terman, 2004) or testing novel functional anatomy involved in action selection (Gurney et al., 2001). Given its prominent role in behavioral function as well as its clinical relevance, models of the BG are important to many different aspects of neuroscience application and research. One of particular importance to this work, is neuromorphic engineering.

Neuromorphic engineering is a bottom-up approach to neural modeling where the single neuron dynamics are implemented in hardware specific digital and analog circuits. The neurons are then connected to each other through different levels of communication fabric to create large neural simulations. These low-power application specific options offer not only a mechanism for simulating large-scale neural models but also a means of embodying them in mobile agents. First introduced by Mead (1989), modern manufacturing processes with higher yield and transistor density have resulted in a renaissance for neuromorphic engineering. This is evidenced by a number of projects such as FACETS/BrainScaleS (Schemmel et al., 2010), SpiNNaker (Furber et al., 2012), Neurogrid (Gao et al., 2012), and SyNAPSE (Merolla et al., 2011; Srinivasa and Cruz-Albrecht, 2012). Each of these have different methods of simulating and abstracting models of the nervous system. How-

ever, they share the common goal of creating large-scale neural models of the nervous system.

One possible application for these low-power neuromorphic processors is in neural control engineering. The work of Voss et al. (2004) demonstrated one of the first examples of combining dynamical control theory and electrophysiology, where the state of a reduced neuron model was estimated using an unscented Kalman filter. This helped establish the strategies for observing and controlling the highly non-linear dynamics of neural systems. Although that original application contained only a single neuron and merely estimated the missing model parameters, subsequent work has shown the robustness of this strategy in a number of control and estimation paradigms (Abarbanel et al., 2008; Ullah and Schiff, 2009, 2010; Schiff, 2010, 2012; Aprasoff and Donchin, 2012) .

Closed-loop control of deep brain stimulation (DBS) has already proven to be more effective in the treatment of late-stage Parkinson's disease then open-loop configurations (Rosin et al., 2011). Model-based control strategies can potentially provide further clinical benefit as well as improved power efficiency. A key component to power-efficiency lies in the neuromorphic hardware discussed above. In addition, the computational models of the basal ganglia capable of capturing the important dynamics of Parkinson's disease need to be further developed along with more hardware friendly versions of those networks.

The individual neurons that comprise the subcortical structures of the BG have distinct firing characteristics that are thought to be essential for its function. These firing patterns are too complex for simple neuron models such as the leaky-integrate-and-fire (LIF) (Dayan and Abbott, 2005) and the more complex conductance based models are difficult to implement in hardware (Rangan et al., 2010). In order to satisfy the firing requirements and facilitate the realization of these models in hardware we are proposing the use of the simple hybrid neuron of Izhikevich (2003).

Simultaneously lauded for its ability to replicate a multitude of neuronal dynamics (Izhikevich, 2007a) and criticized for its lack of stability (Touboul, 2009), the Izhikevich simple hybrid neuron appears to be an ideal candidate for large-scale biologically realistic models of the BG. However, its use in existing spiking models has been sparse and when employed, only the fast-spiking and regular-spiking modes are used. To justify its use in developing novel BG theories as well as its inclusion in future neuromorphic hardware, four published models of the BG are explored using the hybrid model. The focus is on faithfully replicating both the single neuron and overall network dynamics while staying as close to the originally published architectures as possible. An additional motivation is to create a foundation for models amenable to both hardware implementations as well as model-based control systems.

## 9.2 Materials and methods

### 9.2.1 Simple Izhikevich neuron

Hybrid neuron are characterized by a set of continuous non-linear spike functions and a discontinuous after-spike reset. These are derived from dynamical system theory and are capable of replicating the firing activity of many cortical neurons (Izhikevich, 2007a). The model is expressed by the simple membrane voltage equation,

$$\frac{dV}{dt} = 0.04V^2 + 5V + 140 - u + I, \tag{9.1}$$

a recovery variable,

$$\frac{du}{dt} = a(bV - u), \tag{9.2}$$

and the spike reset equations

$$\text{if V} \geq 30\text{, then} \begin{cases} V \leftarrow c \\ u \leftarrow u + d. \end{cases}$$

The current $I$ represents the sum of the total synaptic and externally applied currents. The synaptic influence on the cell is defined by

$$I_{syn} = g_i^{syn} \cdot (E_{syn} - V). \tag{9.3}$$

Where $g$ is the synaptic conductance and $E_{syn}$ is the reversal potential of the synapse. After the arrival of a spike the synaptic currents are decayed based on

$$\tau_{syn} \frac{dg_i^{syn}}{dt} = -g_i^{syn} + \sum W_{ji} \delta \left(t - t_j\right). \tag{9.4}$$

In all of the simulations presented here a Euler integration method is used with time step $\tau = 1ms$. Often such a large time step can have undesired effects on the slower recovery variable. To avoid instability in the model the precise time the membrane voltage crosses the peak of $30$ mV is calculated using the linear interpolation formula from Izhikevich (2010):

$$t_{peak} = t + \frac{30 - V(t)}{V(t + \tau) - V(t)}. \tag{9.5}$$

The recovery variable $U$ updated using $(t_{peak} - t)$ instead of $\tau$. This increases the computational cost when a spike occurs but maintains an appropriate update of $U$ that is particularly important in some of the bursting neuron types found in the basal ganglia.

### 9.2.2 Integrate-and-fire-or-burst model

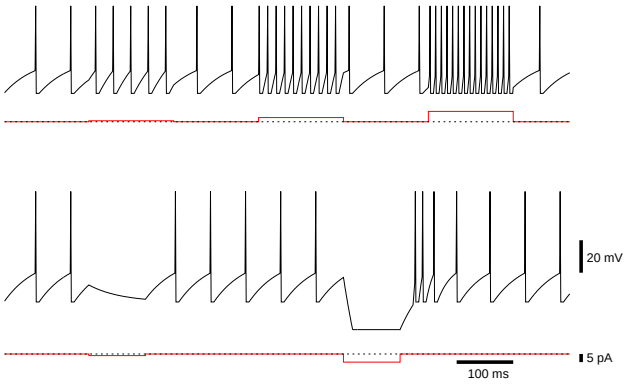



**Figure 9.1:** Thalamocortical neuron response using IFB model.

The integrate-and-fire-or-burst (IFB) model neuron is a simple model capable of capturing the burst responses of thalamacortical neurons. The model is defined by the differential equation

$$C_m \frac{dV}{dt} = I - I_{leak} - I_T, \tag{9.6}$$

and the leak current

$$I_{leak} = g_{leak} \left( V - E_{leak} \right), \tag{9.7}$$

and the T-type calcium channel current

$$I_T = g_T \cdot h_T \cdot H\left(V - E_h\right) \cdot \left(V - E_{Ca^{2+}}\right). \tag{9.8}$$

$H$ is the Heaviside function and $h_T$ is the inactivation variable defined by

$$h_T = \begin{cases} \frac{h_T}{\tau_h^-} & (V > V_h) \\ \frac{(1-h_T)}{\tau_h^+} & (V \leq V_h) \end{cases}$$

When the membrane voltage goes above $V_p$ an action potential is fired and the neuron is reset to $V_r$. The model is then placed in a refractory period for $5$ ms.

The inactivation variable, $h_T$, controls the burst response of the mode. When the membrane voltage is hyperpolarized $h_T$ deinactivates with timescale $\tau_h^+$. When the neuron is depolarized, $h_T$ is immediately activated due to the Heaviside function and begins to inactivate with timescale $\tau_h^-$. This allows $h_T$ to essentially build during periods of sustained hyperpolarization and when depolarized again facilitate bursts of action-potentials. An example thalamocortical neuron model is presented in Figure 9.1.

### 9.2.3 Modeling basal ganglia nuclei

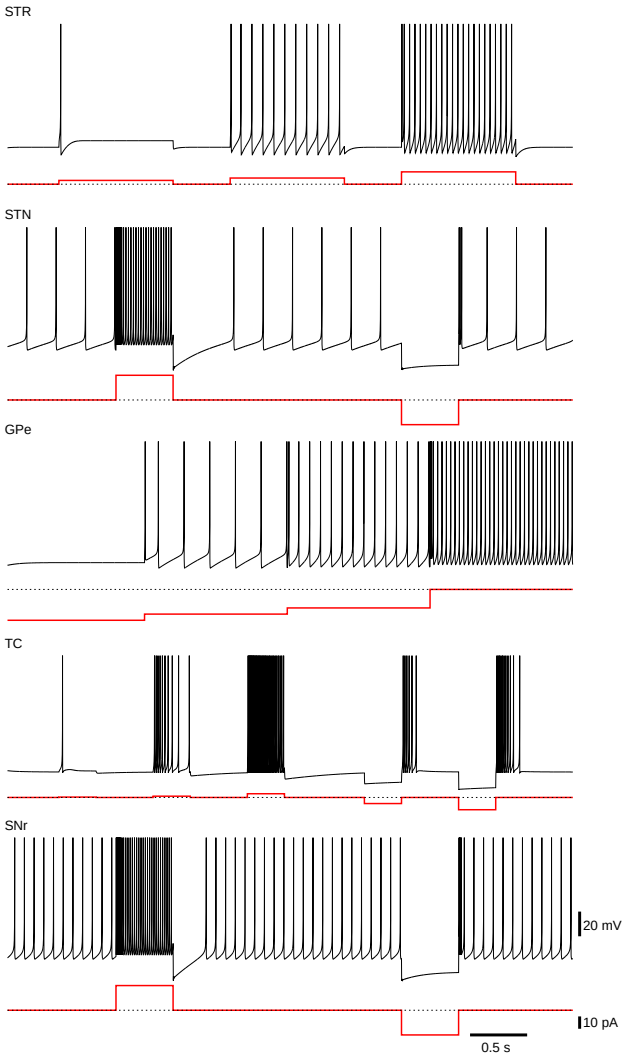



**Figure 9.2:** Basal Ganglia neurons. Many of the firing characteristics inherent to neurons of the BG nuclei are captured by the simple hybrid model. The model parameters used to achieve these patterns are STR: ($a = 0.02$,$b = 0.2$,$c = -65.0$,$d = 8.0$), STN: ($a = 0.005$,$b = 0.265$,$c = -65.0$,$d = 2.0$), GPe: ($a = 0.005$,$b = 0.585$,$c = -65.0$,$d = 4.0$), TC: ($a = 0.002$,$b = 0.25$,$c = -65.0$,$d = 0.05$), SNr: ($a = 0.005$,$b = 0.32$,$c = -65.0$,$d = 2.0$). Note that the GPi response is not shown here but has similar firing characteristics to the GPe neurons only with a higher basal level of firing.

As described in Chapter 7 there are six nuclei in historical models of the BG that can be separated into multiple pathways. Here, the dual pathway model is used. These dual pathways play an important role in the action selection models presented. Using the Humphries

et al. (2006) terminology the direct pathway acts as the “Selection” mechanism and the indirect pathway as the “Control” mechanism. Below we describe the fire patterns for the nuclei modeled in this chapter.

**Striatum**

For the models described here we selected parameters that matched the the medium-size spiny neurons of the striatum. The neurons in Figure 9.2 respond to increases in depolarizing inputs with an increase in firing rate. Notably absent from the single cell response is the presence of bistable behavior. Found in vitro, the neurons will fall into, the “up-state” and the “down-state.” These contribute to the responses to deploarizing currents and in particular, the long-latency spike-discharges. The simple hybrid model used here is not capable of replicating those dynamics, although there are versions of it that can (Izhikevich, 2007a). Despite this, the model is still suitable as the bistablity was determined to be unnecessary for the networks replicated here (Humphries et al., 2006).

**Globus Pallidus External**

The GPi neuron model presented in Figure 9.2 are intrinsically active and respond to hyperpolarizations with a decrease in tonic firing. Unlike the models of Rubin and Terman (2004) the simple hybrid model is unable to replicate the transition from tonic firing to bursting in response to sufficient hyperpolarizations. This however, did not appear to be a necessary property to replicate the dynamics of the model.

**Globus Pallidus Internal**

The firing patterns of the GPi neurons employed here match those of the GPe neurons presented above. However, they have a higher level of basal activity (Rubin and Terman,

2004).

**Subthalamic Nucleus**

The neurons of the STN model used here have spontaneous activity of a around $5 - 10$ Hz. Physiologically this is due to voltage activated $Na^+$ channels. In addition, when a depolarizing current is applied the STN model responds with a high-frequency tonic firing with a quiescent period after sustained depolarization. The model will fire rebound bursts in response to hyperpolarizations that are sufficient in time and magnitude, Figure 9.2. Missing from this model are the spontaneous bursts in the absence of inputs as well as plateau deploarizations as observed experimentally. The simple hybrid model is incapable of including all of the STN cell dynamics. However, the firing properties of the model neuron is sufficient to encompass all of those included in the original models.

**Substania Nigra pars reticulata**

The neuron model firing patterns are similar to those of the STN but with higher level of basal activity (see Figure 9.2).

**Thalamus**

The model parameters were selected to have firing patterns that match those of Sherman (2001). They do not fire spontaneously but when in the tonic mode show an increase in firing rate in response to larger depolarizing currents. In the burst mode when subjected to sustained hyperpolarizing input the model neurons respond with periods of bursting that is dependent on the strength and duration of the applied current (see Figure 9.2).

### 9.2.4 A physiologically plausible model of action selection

Conceptually, action selection is the arbitration of competing signals and the role of the BG is to select the most appropriate one. The complex circuitry of the BG is active in gating information flow in the frontal cortex and the selection mechanism can affect simple action all the way up to behaviors and cognitive processing Cohen and Frank (2009). To explore that mechanism in a physiologically meaningful way, Humphries et al. (2006) connected populations of realistic "spiking neurons" configured using the functional anatomy of Gurney et al. (2001). The biological fidelity of the model was validated at the population level as well as single-unit recordings from networks replicating anesthetized or lesioned conditions. The ability of the model to replicate single neuron dynamics under normal conditions similar to Rubin and Terman (2004) or the simulations presented in Figure 9.2 was not presented but descriptions of the included dynamics were included in the appendices. This was the first network model created for this work.

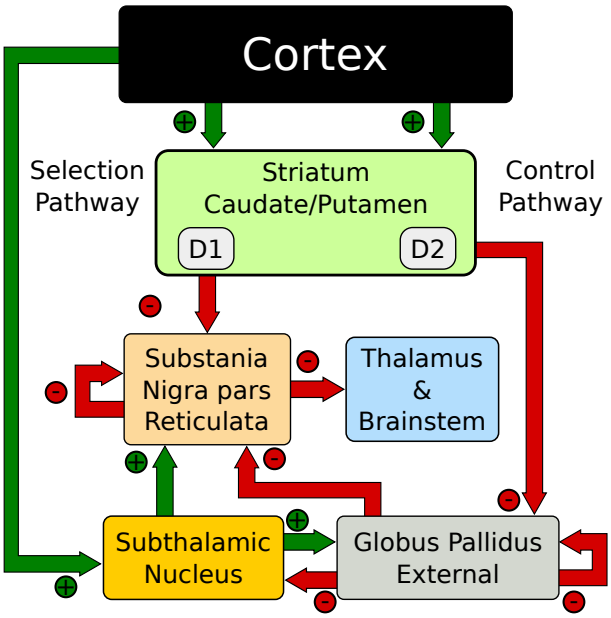



**Figure 9.3:** Action selection network model (Humphries et al., 2006).

The model exploited the concept of competing anatomical channels within the BG.

Three separate channels were constructed using the network of Figure 9.3. Each population consisted of 64 neurons with the parameters of Table 9.1A. Most connections of the model were focused projections where post-synaptic connections were randomly sampled within a channel using the probability $\rho_c = 0.25$. However, the diffuse projections listed in Table 9.1B, spanned all channels and the connection probability $\rho_c$ was divided among each of those. This is consistent with the more diffuse outputs from the STN (Haber, 2010). Cortical inputs to the striatum were simulated as Poisson random spikes.

**Table 9.1:** Parameters for the model of action selection.

**A. Neuron parameters**

| Neural Region | a | b | c | d | $\mathbf{I_{app}}$ (pA) |
|---|---|---|---|---|---|
| STR | 0.02 | 0.2 | −65.0 | 8.0 | 0.0 |
| SNr | 0.005 | 0.320 | −65.0 | 2.0 | 25.0 |
| STN | 0.005 | 0.265 | −65.0 | 2.0 | 20.0 |
| GPe | 0.005 | 0.585 | −65.0 | 4.0 | 5.0 |

**B. Connections**

| Source → Destination | Synaptic Conductance | Delay (ms) |
|---|---|---|
| Cortical Input → STR | 0.2 | 11 |
| Cortical Input → STN | 0.2 | 6 |
| Striatum D1 → SNr | 0.12 | 6 |
| Striatum D2 → GPe | 0.1 | 6 |
| GPe → STN | 0.025 | 6 |
| GPe → GPe | 0.025 | 2 |
| GPe → SNr | 0.015 | 6 |
| SNr → SNr | 0.015 | 6 |
| STN → SNr Focused | 0.075 | 2 |
| STN → SNr Diffuse | 0.35 | 2 |
| STN → GPe Focused | 0.075 | 2 |
| STN → GPe Diffuse | 0.35 | 2 |

**C. Synaptic parameters**

| Parameter | Value |
|---|---|
| $\tau_{ge}$ | 5 (ms) |
| $\tau_{gi}$ | 100 (ms) |
| $E_{exc}$ | 0 (mV) |
| $E_{inh}$ | −80 (mV) |
| $V_{rest}$ | 0 (mV) |

### 9.2.5 The parkinsonian BG and deep brain stimulation

The modeling study of Rubin and Terman (2004) presented the first explanation for the mechanism of action of deep brain stimulation. In Parkinson's disease there is a marked loss of dopaminergic cells in the Substania Nigra pars compacta. The reduction in tonic and phasic dopamine onto the BG nuclei results in, among many other phenomena, a rhythmic synchronization of the major output nuclei of the BG. Within the computational model this resulted in a decrease in the ability of thalamocortical neurons to respond to depolarizing cortical inputs. It was hypothesized that this loss of relay fidelity is one of the underlying causes for many clinical Parkinsonian symptoms.

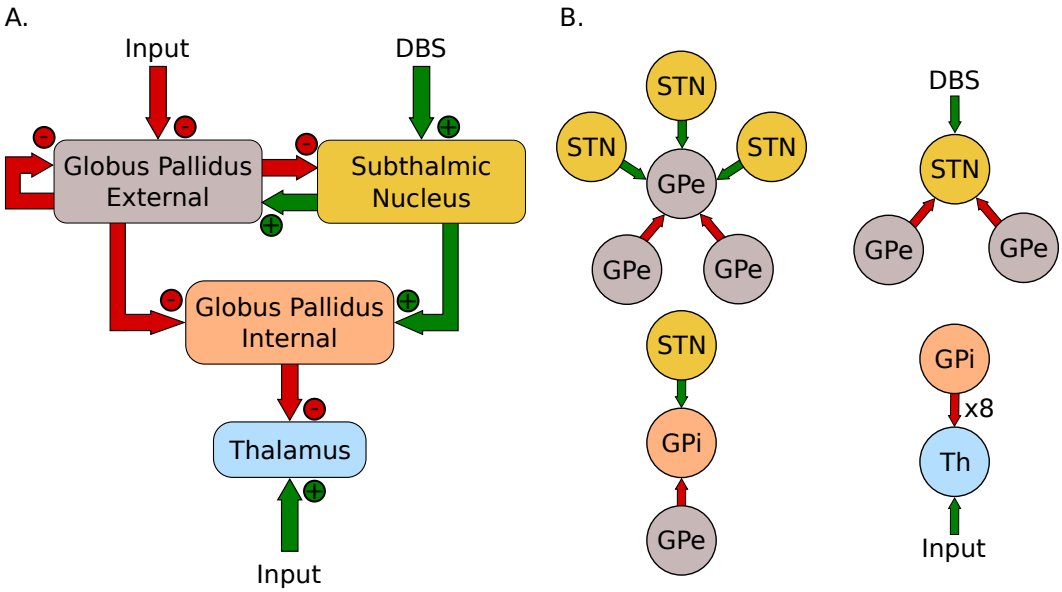



**Figure 9.4:** (A) Network layout of Rubin and Terman (2004). (B) Individual neuron connections.

The network of Rubin and Terman (2004) (RT Model), illustrated in Figure 9.4A, consists of four populations: the GPe, STN, GPi and thalamus. With the exception of the thalamus, that contains 2 neurons, each population has 16 neurons. Unlike the action selection model presented above, the RT model maintained consistent network connectivity that was exactly the same used by Rubin and Terman (2004). Figure 9.4B illustrates the

connectivity patterns for individual neurons of the network.

**Table 9.2:** Model parameters for Parkinson's disease model.

**A. Neuron parameters**

| **Neural Region** | **a** | **b** | **c** | **d** | $\mathbf{I_{app}}$ (pA) |
|---|---|---|---|---|---|
| GPe | 0.005 | 0.585 | −65.0 | 4.0 | 5.0 |
| GPi | 0.005 | 1.2 | −65.0 | 4.0 | 7.0 |
| STN | 0.005 | 0.265 | −65.0 | 2.0 | 15.0 |
| TC | 0.002 | 0.25 | −65.0 | 0.05 | 0.0 |

**B. Connections**

| | **Synaptic Conductance Range** | |
|---|---|---|
| **Source → Destination** | **Low** | **High** |
| GPe → GPe | 0.1 | 0.2 |
| GPe → STN | 0.1 | 0.2 |
| GPe → GPi | 0.3 | 0.4 |
| STN → GPe | 0.2 | 0.3 |
| STN → GPi | 0.5 | 0.6 |
| GPi → TC | 0.02 | 0.0225 |

**C. Input Parameters**

| **Parameter** | **Value** |
|---|---|
| $i_{SM}$ | 30 (pA) |
| $\delta_{SM}$ | 3 (ms) |
| $\rho_{SM}$ | 25 (ms) |
| $i_{DBS}$ | 130 (pA) |
| $\delta_{DBS}$ | 1 (ms) |
| $\rho_{DBS}$ | 8 (ms) |

The parameters used for the neurons of the RT model are listed in Table 9.2A. The synaptic conductances were randomly selected from a normal distribution with the ranges specified in Table 9.2B. There are two sources of depolarizing input current used in this

model. Both follow an equation of the form

$$I = i_{\chi} \cdot H\left(sin\frac{2\pi t}{\rho_{\chi}}\right) \cdot \left(1 - H\left(sin\frac{2\pi(t+\delta_{\chi})}{\rho_{\chi}}\right)\right), \tag{9.9}$$

where $H$ is the Heaviside function and $\chi \in \{SM, DBS\}$, $SM$ and $DBS$, are somatomotor and deep brain stimulation respectively. The values used for each of these currents is presented in Table 9.2C. The synaptic delay, imposed by the simulator, was $2$ ms.

The role of the STN and GPe in this model is to create the patterns of activity within the GPi that are observed experimentally. As discussed above, the GPi is the major output nuclei and is responsible here for appropriately controlling the activity of the thalamic neurons. The role of the thalamus in this case is simplified into a relay station; responsible for appropriately relaying depolarizing signals from somatomotor inputs.

Under the normal mode of operation the nuclei of the BG produce irregular firing patterns and the thalamus is capable of relaying somatomotor information reliably. In the Parkinsonian state the GPe and STN nuclei have more regular synchronized firing rates and the thalamic relay fidelity is greatly diminished. Similar to the original work the Parkinsonian mode is accomplished by reducing $GPe \rightarrow GPe$ to $0$ as well as reducing the current $I_{app}$ to $-19$. This follows the procedure of Rubin and Terman (2004) and is based on the activity patterns of Terman et al. (2002). Finally, the application of DBS to the STN is used to restore relay capabilities while in the Parkinsonian state.

To quantitatively evaluate the performance of the model in each of the three states, an error index measure was introduced in Rubin and Terman (2004). This is defined as

$$EI = \frac{m+b}{t} \tag{9.10}$$

where $m$, representing misses, is the number of somatomotor signals that were not relayed, $b$, bad, is the number of responses that result in multiple spikes and $t$ is the total number of stimulus inputs. The error index evaluation was completed by running $20$ simulations of the model in each of the modes described above. Each run resulted in different results due to the randomly selected connection weights described in Table 9.2B. The error index was calculated for each of the TC cells and a box and whisker plot were created to compare with Rubin and Terman (2004).

### 9.2.6 Restoring action selection in the Parkinsonian basal ganglia

In addition to exploring different sites of DBS application, the work of Pirini et al. (2009) divided the RT model into 2 distinct control channels and added a striatal current into the GPi, representing the direct pathway discussed above. The model was capable of demonstrating simple two channel action selection by way of disinhibition. The successful switching between channels was lost under Parkinsonian conditions but could be restored by the application of DBS into the STN.

The exact two channel network from Pirini et al. (2009) was constructed here. The network layout and individual neuron connections matched those of Figure 9.4 and the parameters of Table 9.2 were used with some modifications to handle the change in network configuration as well as the stochastic input pattern. The somatomotor input into the TC cells was reduced to $24.5$ pA and the duration, $\delta_{SM}$ was changed to $4$ ms. In addition, the pulse times were randomly selected from an exponential distribution with a mean of $15$ Hz. The DBS current was reduced to $50$ pA and the period $\rho_{DBS}$ was reduced to $7$ ms to more closely match the original work.

The action selection mechanism signaled by the striatum is modeled as a current input into the GPi nucleus. Under normal conditions the current values are $I_{On} = 0$ pA and

$I_{Off} = 27$ pA. For the Parkinsonian condition the values are set to $I_{On} = 0$ pA and $I_{Off} = 22$ pA to represent the loss of striatal inputs into the GPi. Each state lasts $2$ s before switching.

### 9.2.7 Thalamic relay fidelity between the BG and thalmus

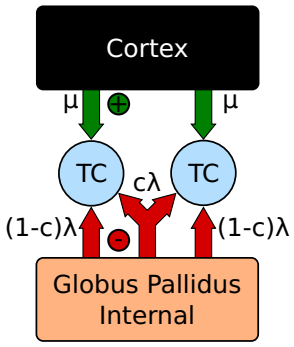



**Figure 9.5:** Correlation network configuration (Reitsma et al., 2011).

Correlated firing in neuronal ensembles is important in both understanding information encoding and in interpreting functional anatomy (Cohen and Kohn, 2011). Correlated activity in many brain regions has been linked to stimulus decoding and discrimination, attention, and motor behavior (de la Rocha et al., 2007). In addition, highly correlated firing has been associated with pathological conditions. In the basal ganglia in particular, correlated activity of globus pallidus internal (GPi) neurons is associated with Parkinson's disease or pharmacological agents causing Parkinsonian like conditions (Reitsma et al., 2011).

Reitsma et al. (2011) explored the implications the temporal relationships that emerge from the GPi have on the relay fidelity of the thalamic neurons they innervate. In the Parkinsonian BG the firing patterns become increasingly oscillatory with pronounced bursting. This synchronous fire rate can have deleterious effects on the functionality of the BG. The effect of those patterns of activity have on thalamocortical relay fidelity was explored

through correlation analysis of a computational model. Exploring single cell responses to correlated inputs is important in understanding how population level effects translate down to single cells. In addition, this helps in the elucidation of the important properties of correlations (Cohen and Kohn, 2011).

One conclusion of that work was that the integrate-and-fire-or-burst (IFB) neuron model demonstrated similar firing patterns and correlation transfer to that of a conductance-based model. This not only strengthened the overall conclusions of the study but also motivated the authors to suggest the IFB model as a suitable replacement for the conductance-based model in correlation studies. Here, we explore if a similar result can be accomplished with the hybrid neuron.

The IFB model achieves the bursting dynamics of TC cells through the inclusion of a T-type calcium channel. When the membrane voltage is hyperpolarized the inactivation gate of the channel begins to deinactivate. When the membrane voltage is depolarized the channel remains activated until the gate is reinactivated. Unlike the IFB model, the hybrid neuron used here does not have an explicit bursting mechanism. Instead the recovery variable is used to put the neuron with the bursting regime of the phase-portrait (Izhikevich, 2007a). Reitsma et al. (2011) demonstrated that although the T-current is required to replicate the physiological spike patterns, it is not needed to demonstrate transfer of correlations. However, our goal here was to replicate both the physiological spike patterns of the thalamocortical neurons as well as the correlation transfer.

The model consists of two spiking thalamocortical (TC) neurons subjected to inhibitory input from an engineered GPi signal as well as an excitatory input representing cortical innervations. This is illustrated in Figure 9.5. Each TC neuron receives independently generated 20Hz Poisson random excitatory inputs. The GPi spike trains are generated by inhomogeneous Poisson rate functions defined as $\lambda(t)$, with a fraction, $c$, of spikes

overlapping between each TC neuron. For values of $c > 0$ a single spike train with rate $\lambda(t)/c$ is constructed. Each cell then samples from this spike train with probability $c$. For $c = 0$ two Poisson random spike trains are generated using a common rate function $\lambda(t)$.

The model and corresponding analysis was computed using the numerical programming language Octave (Eaton et al., 2008). Table 9.3 presents the parameters used in the model. The simple hybrid neuron of Equation 9.1 is used however, to increase the stability of the simulations under the increased synaptic activity of the GPi the hybrid solution method from Izhikevich (2010) was used along with the spike peak detection presented in Equation 9.5. The hybrid numerical method treats the synaptic influence implicitly resulting in a linear dependence on the future value of the membrane voltage. The equation is

$$V(t+1) = \frac{V(t) + 0.04V^2 + 5V + 140 - u + g(t)E(t) + I}{1 + g(t)}. \tag{9.11}$$

Where the total conductance is

$$g(t) = \sum g_i(t) \tag{9.12}$$

and the total reversal potential is

$$E(t) = \sum \frac{g_i(t) \cdot E_i}{g(t)}. \tag{9.13}$$

**Table 9.3:** Parameters for the model of correlation transfer.

| Parameter | Value |
|---|---|
| $a$ | $0.002$ |
| $b$ | $0.25$ |
| $c$ | $-65.0$ |
| $d$ | $0.05$ |
| $g_e$ | $0.12$ |
| $\tau_e$ | $6.0$ (ms) |
| $V_e$ | $0.0$ (mV) |
| $g_i$ | $0.09$ |
| $\tau_i$ | $15.0$ (ms) |
| $V_i$ | $-85.0$ (mV) |

### IFB model

The IFB neuron model was also developed for comparison with the original work. Using Equation 9.6 and the parameters of Table 9.4 the procedure outlined above was repeated.

**Table 9.4:** Parameters for the model of correlation transfer.

| Parameter | Value | |
|---|---|---|
| $C_m$ | 2. | $\mu F/cm^2$ |
| $I_{app}$ | 0.89 | $nA/cm^2$ |
| $g_{leak}$ | 0.035 | $mS/cm^2$ |
| $g_T$ | 0.07 | $\mu F/cm^2$ |
| $g_i$ | 0.024 | $\mu F/cm^2$ |
| $g_e$ | 0.06 | $\mu F/cm^2$ |
| $E_{leak}$ | $-65.0$ | $mV$ |
| $E_{Ca}$ | 120.0 | $mV$ |
| $E_e$ | 0. | $mV$ |
| $E_i$ | $-85.$ | $mV$ |
| $V_h$ | $-70.$ | $mV$ |
| $\tau_h^+$ | 100. | $ms$ |
| $\tau_h^-$ | 20. | $ms$ |
| $V_r$ | $-68.$ | $mV$ |
| $V_p$ | $-50.$ | $mV$ |
| $\tau_e$ | 4. | $ms$ |
| $\tau_i$ | 15. | $ms$ |

### Input patterns

Consistent with the original work, four different GPi input patterns are constructed to emulate normal and Parkinsonian conditions observed experimentally. Samples of the rate functions are illustrated in Figure 9.6A. The normal input is a constant $70$ Hz Poisson random spike train. Similar to the original work the first Parkinsonian pattern, labeled oscillatory, is constructed as a sum of 21 sine waves. The individual sine waves have frequencies ranging from $5$ Hz to $15$ Hz with step changes of $0.5$ Hz between them. These are weighted by a Gaussian distribution with a mean of $10$ Hz and a variance of $1.5$ Hz; resulting in the $10$ Hz component dominating the rate function. The phase of the sine waves are then randomly shifted and summed together. The resulting function is then amplified by $50$ Hz and shifted up by $150$ Hz. Any negative values are railed to zero. Although

constructed differently than those described in Reitsma (2010); Reitsma et al. (2011), the resulting function qualitatively matches the samples presented there. In addition, the resulting functions exhibited a distinct peak at $10$ Hz, see Figure 9.6A below, similar to the original work.

The third pattern, labeled Bursty, consists of a basal level of firing at $70$ Hz interrupted by random bursts stepping to $470$ Hz. The duration of each burst is selected from a Gaussian distribution with a mean of $30$ ms and a variance of $10$ ms. The time between bursts is selected from a Poisson distribution with a mean of $70$ ms. The final GPi pattern, labeled Oscillatory Bursty, is constructed similar to the bursty case however, the inter-burst-interval is selected from a Gaussian distribution with a mean of $30$ ms and a variance of $10$ ms. This results in more periodic bursts.

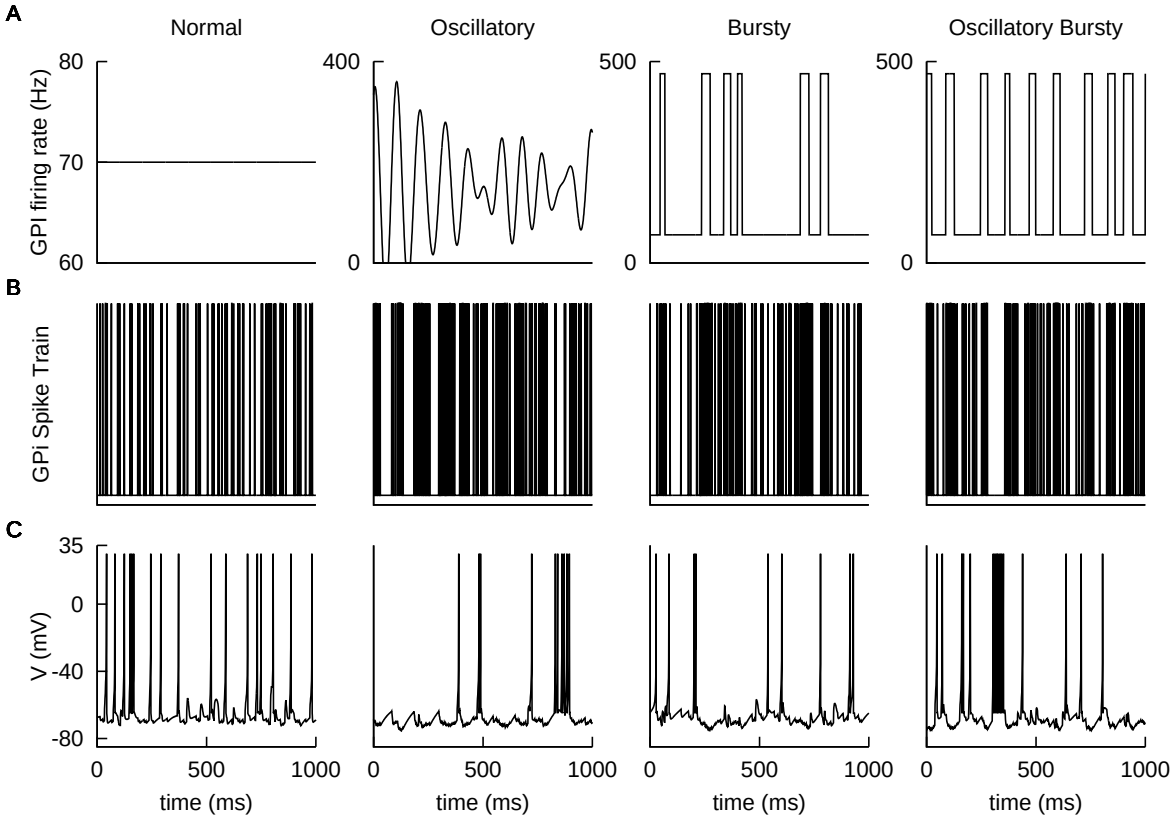



**Figure 9.6:** Example GPi spike patterns and TC cell responses for each of the four modes for the Izhikevich model. (A) Example input rate functions. Resulting GPi spike trains, (B), and TC Cell responses, (C).

These rate functions are then used to generate Poisson random spike trains. For the Izhikevich model examples of these spike trains are presented in Figure 9.6B with the corresponding TC neuron response in Figure 9.6C. For the IFB model examples of the spike trains are presented in Figure 9.7B with the corresponding response of the inactivation variable $h_T$ in Figure 9.7C and the TC neuron response in Figure 9.7D. These patterns were selected by the authors to replicate firing patterns and overall spike rates found in the GPi under parkinsonian conditions (Reitsma et al., 2011).

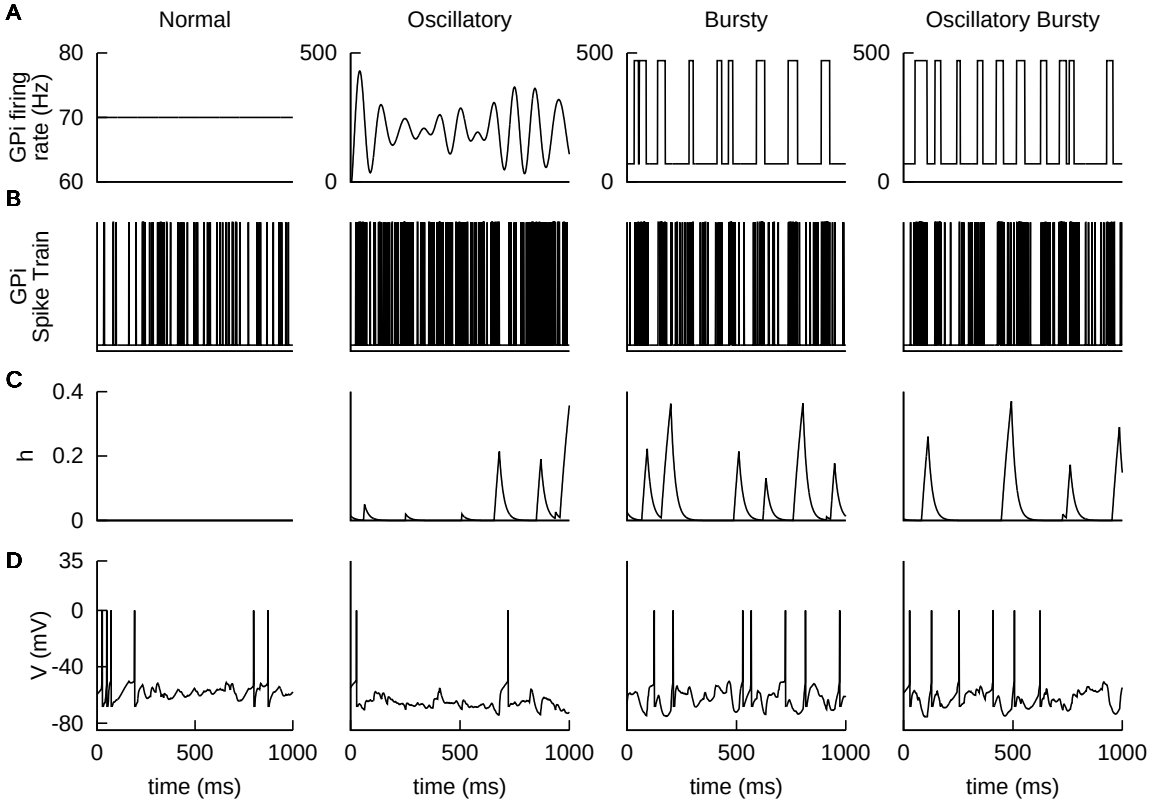



**Figure 9.7:** Example GPi spike patterns and TC cell responses for each of the four modes for the IFB model. (A) Example input rate functions. (B) Resulting GPi spike trains, (C) Response of $h_T$. (D) TC Cell responses.

### TC model spike response

Both interspike interval (ISI) distributions and power spectra were computed on the model TC cells for comparisons with the original work. The power spectra was computed for

the TC model spike response as well as the corresponding GPi and cortical inputs using the point process multi-taper spectrum analysis from the Chronux software package (Bokil et al., 2010).

**Correlation calculations**

The measure of correlation is calculated using the Pearson's correlation coefficient. This is a spike count measurement that compares the number of spikes that occur over a window of length $T$ defined as

$$\rho(t) = \frac{cov(n_1(T), n_2(T))}{[var(n_1(T)) \cdot var(n_2(T))]^{1/2}}. \tag{9.14}$$

Where $cov$ is the covariance, $var$ is the variance and $n_1(T)$ and $n_2(T)$ are spike counts at window $T$.

The correlation coefficient is used to calculate the correlation susceptibility. This is used to quantify the degree to which correlations are transfered through the model. This is found using the equation

$$\rho_{out}(T) = S(T)\rho_{in}(T) - k. \tag{9.15}$$

Where $\rho_{in}$ and $\rho_{out}$ are the GPi input correlation coefficient and the TC output correlation coefficient respectively.

To demonstrate how sensitive the TC neurons were to correlated input the correlation coefficients were calculated for $c = 0, 0.25, 0.5, 0.75, 1.0$ and similar to Reitsma et al. (2011) a sample bootstrap method was used to generate confidence intervals on the analysis. For each value of $c$, 30 simulations of were run for $100$ s each. This resulted in $150$ pairs

of correlation coefficients. A straight line was then fit between the values of $\rho_{in}$ and $\rho_{out}$ to find the correlation susceptibility $S$ based on the slope of the line. This was completed over a range of window sizes $T$. The $150$ pairs were then sampled with replacement to generate a new set correlation coefficients and $S$ values. This resampling was completed 1000 times to generate $98\%$ confidence bands for each value of $T$.

### 9.2.8 HRLSim

With the exception of the correlation study, all of the models were simulated using the HRLSim neural simulator package (Minkovich et al., 2012). HRLSim is the first distributed GPGPU spiking neural simulation environment. It currently supports two different point neuron implementations, the Leaky Integrate-and-Fire (LIF) model and the simple hybrid Izhikevich model. With an emphasis on high-performance, HRLSim was developed to support the modeling efforts of the SyNAPSE project and its team members. It has also proven extremely useful in general neural simulation studies (Srinivasa and Cho, 2012; O'Brien and Srinivasa, 2013).

## 9.3 Results

### 9.3.1 Action selection

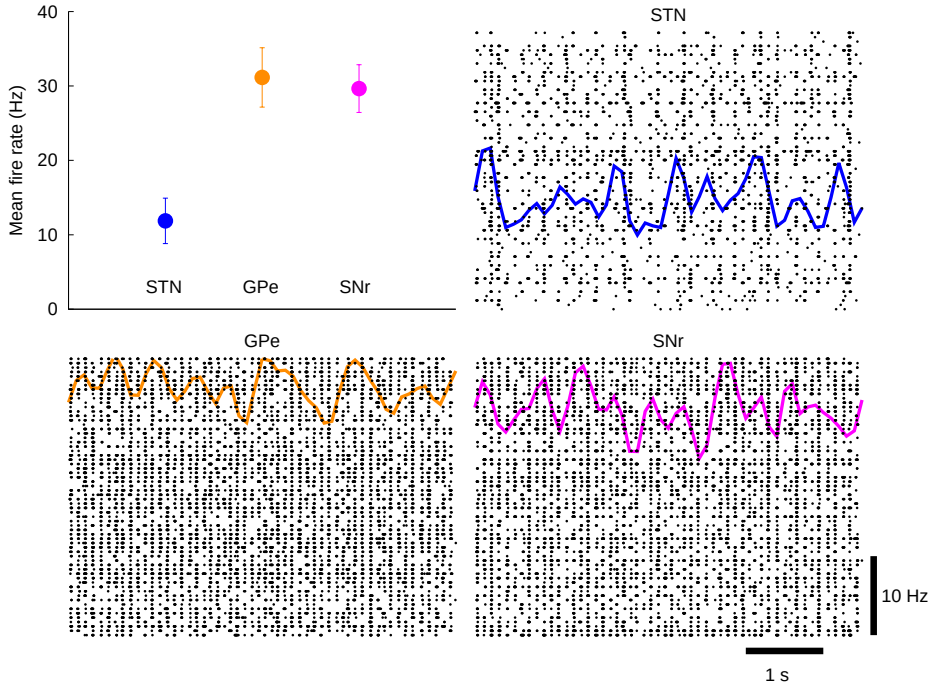



**Figure 9.8:** Basal activity of the model of action selection. Upper left: the mean rates for the STN, GPe and SNr qualitatively match the simulated and experimental results of Humphries et al. (2006). Remaining plots: the spike rasters for each of the nuclei are overlaid with the corresponding spike-count firing rates.

The action selection model of Figure 9.3 was first tuned to match the original model of Humphries et al. (2006). Using the model-as-animal strategy $15$ simulations were completed with different randomly connected networks. From each of those simulations $3$ cell indexes were selected from a normal distribution and the overall activity rate of the last $9$ seconds of simulation were computed for those neurons. The mean rates and $95\%$ confidence intervals were then computed to ensure the activity was in the ranges as Humphries et al. (2006). This is presented in Figure 9.8. In addition, the spike rasters and binned spike count rate functions are included. The overall mean firing rate results are in good

agreement with the original work as well as the experimental results referenced there.

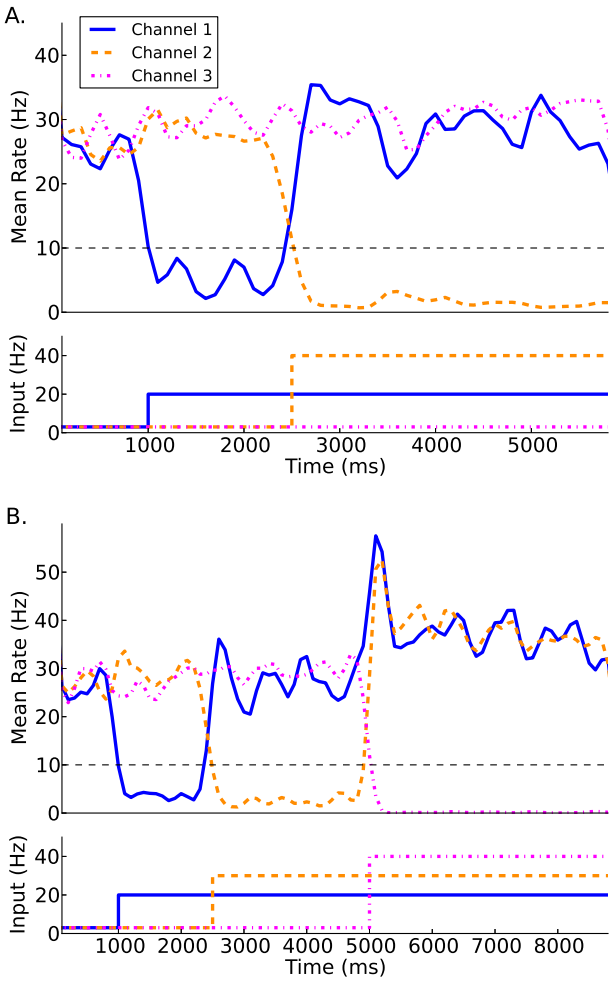



**Figure 9.9:** Action selection performance. The model is capable of appropriate selecting the most salient input between two competing channels (A) as well as three competing channels (B).

Using the protocol of Humphries et al. (2006) the ability of the models to appropriately select the most salient input was first simulated using two of the three channels. Figure 9.9A illustrates the two channel action selection results. Initially the network is at its basal level of activity with a 3 Hz Poisson input. At 1 second the input for channel 1 is increased to 20 Hz, causing, through disinhibition, the selection of that channel. At 2.5 seconds a 40 Hz cortical input is injected into Channel 2. The activity of channel 1 is pushed up to its

basal level of activity and the channel 2 output is inhibited causing it to be selected. This selection mechanism is more decisive then the one presented in Humphries et al. (2006). In the original work the previously selected channel had an increase in activity that was only slightly above the selection limit. To build on this result we tested the selection capabilities of all three channels, something that was not part of the original work. The results of this are presented in Figure 9.9B as well as in Figure 9.10 where the spike rasters of the model nuclei are plotted with the overlaid spike count rate functions. This is an encouraging result and suggests that the functional anatomy of the original work can be extended to more than three channels.

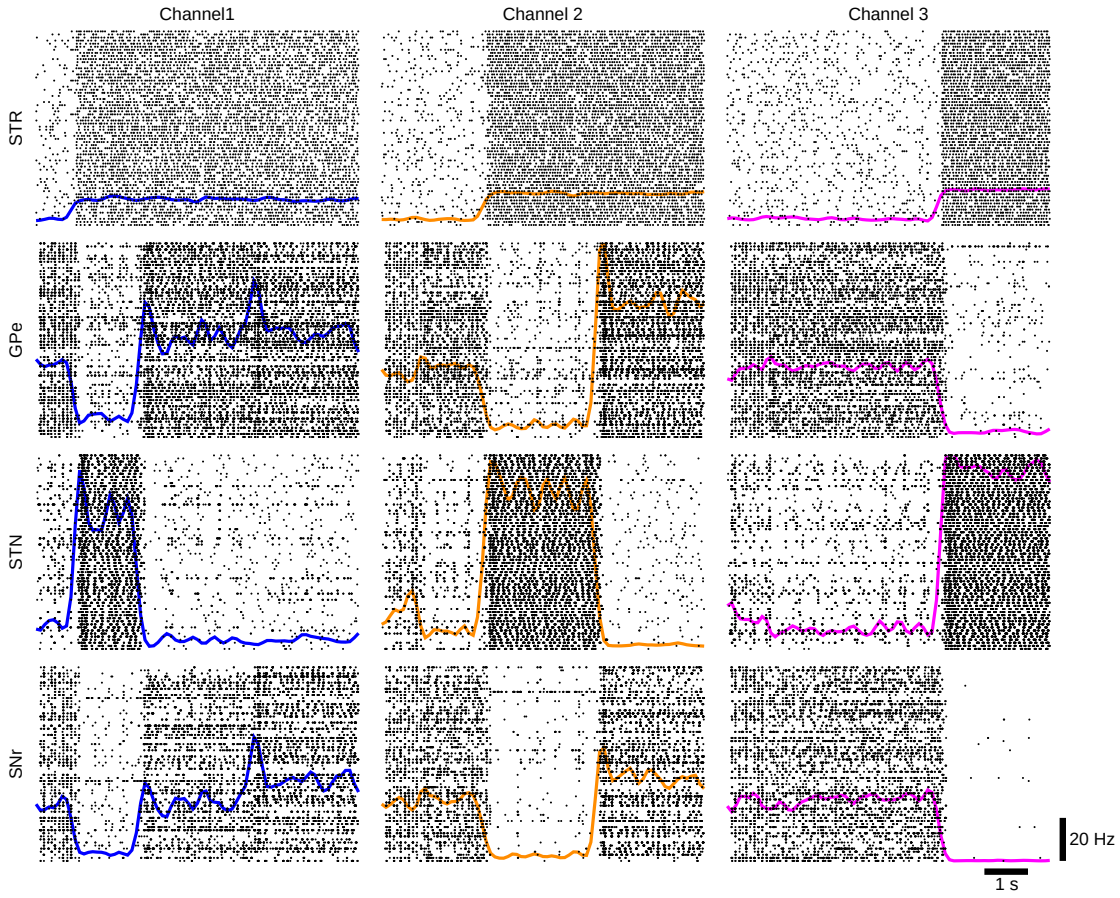



**Figure 9.10:** Network response to competing inputs; spike rasters of the major nuclei of the BG action selection with the spike count rates overlaid.

### 9.3.2 The Parkinsonian BG and deep brain stimulation

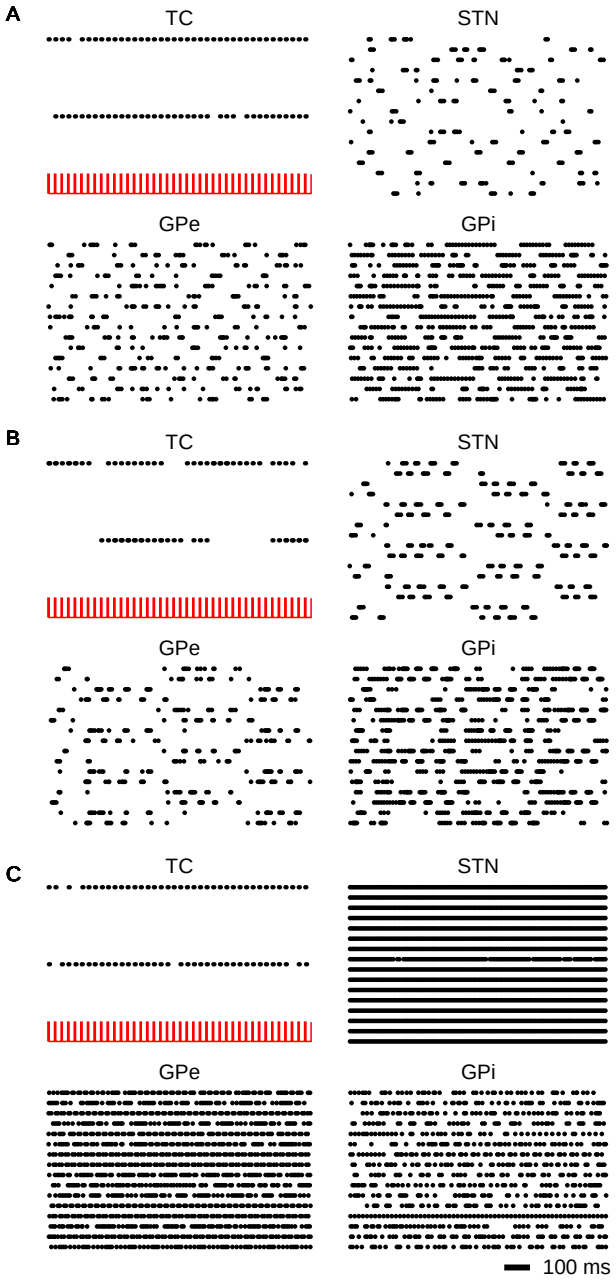



**Figure 9.11:** Simulated recovery of TC relay fidelity. (A) Under normal BG activity the thalamus is capable of relaying somatomotor inputs. (B) Under parkinsonian conditions the BG nuclei fall into oscillatory firing patterns TC relay capabilities are greatly diminished. (C) Application of DBS to the STN restores lost TC relay fidelity.

In the normal mode the BG nuclei have irregular firing patterns with interspike interval coefficients of variation $\geq 1.0$. With this irregular pattern of activity the thalamus is capable of reliably transmitting the somatomotor signals (see Figure 9.11A).

In Parkinson's disease the firing pattern of the BG neurons have been reported to have regular synchronous firing patterns (Walters and Bergstrom, 2010). In Figure 9.11B it can be seen that the BG nuclei begin to fire synchronously. The neurons of the STN separate into two distinct populations with different phases of bursting. The periods of bursting oscillate around $4$ Hz which is consistent with synchronous oscillations observed in the parkinsonian BG (Walters and Bergstrom, 2010). This synchronous activity results a marked loss of thalamic relay. As noted by Rubin and Terman (2004) the GPi activity is affected by the periods of bursting in the GPe, where the GPi would otherwise fire tonically.

The application of DBS to the STN results in an increased firing rate and a disruption of the synchronous oscillations of the BG nuclei. This disruption in the oscillatory activity is sufficient to restore the relay fidelity of the thalamus (see Figure 9.11C).

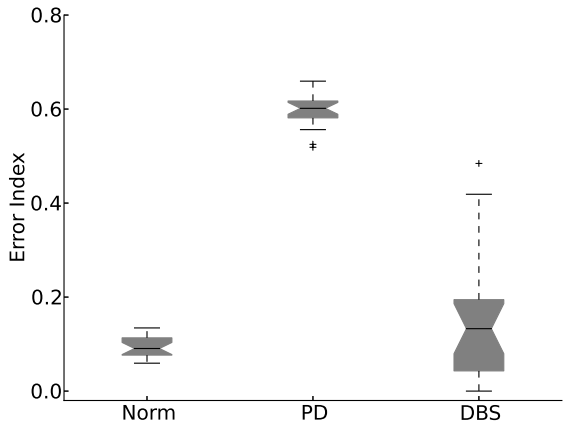



**Figure 9.12:** Error index statisitcs. Allowing the network connection weights to randomly change over 20 simulations results in the Normal and DBS modes operating with less errors then the PD mode.

The results of Figure 9.11 are quantified in Figure 9.12. Here the normal and DBS

modes of the model result in EI medians that are comparable. Although the spreads are somewhat dissimilar neither overlaps with the much higher values measured in the Parkinsonian state.

### 9.3.3 Restoring action selection in the Parkinsonian basal ganglia


**Figure 9.13:** Parkinsonian fire patterns result in a loss of accurate selection capabilities.

The modified RT network of Pirini et al. (2009) puts the theoretical concepts of the previous sections into a dynamical model of action selection. The results of this experiment are shown in Figure 9.13. Once again the loss of faithful relay can be alleviated with the application of DBS to the STN.

### 9.3.4 BG correlation transfer

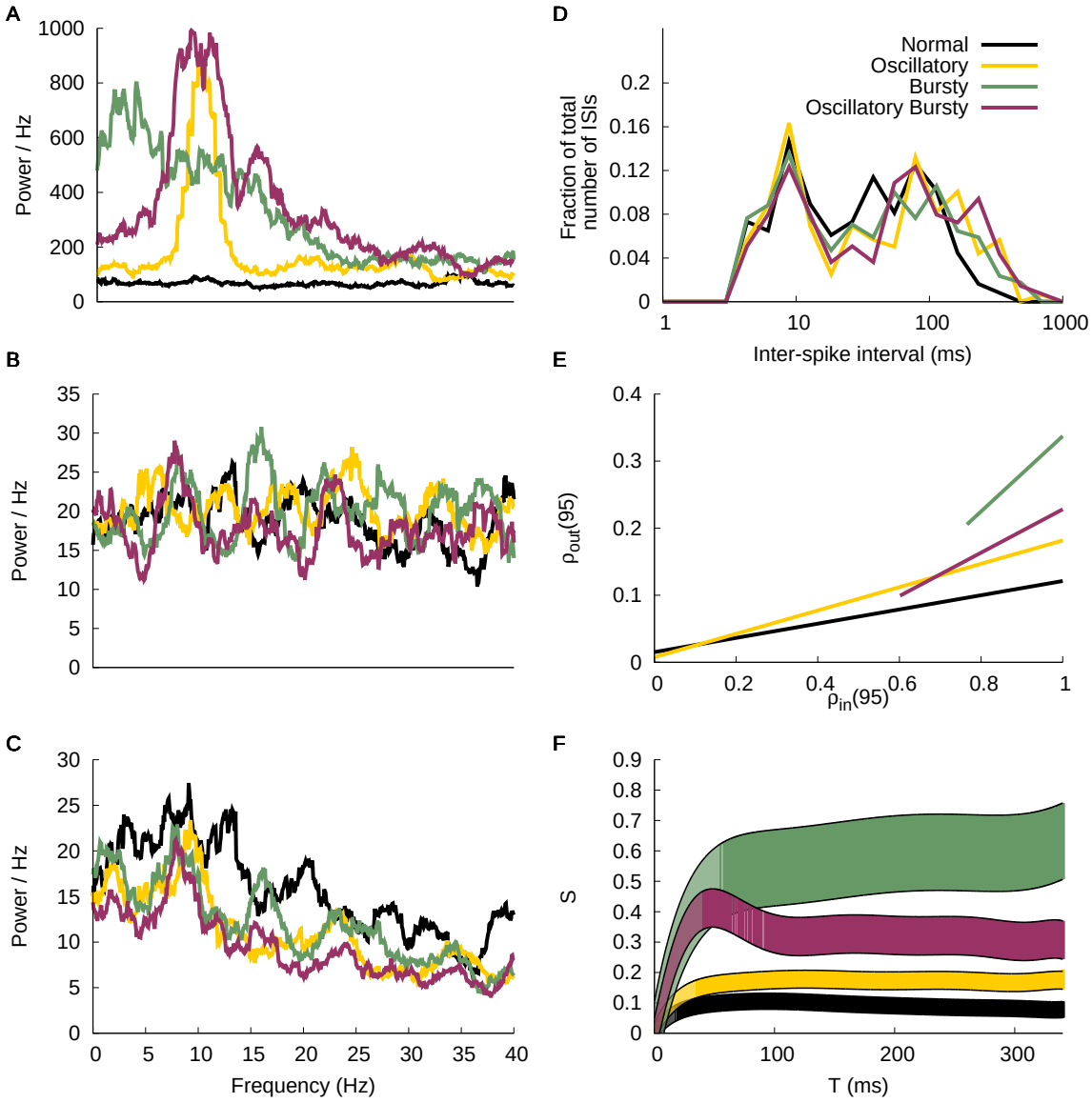



**Figure 9.14:** Correlation analysis. Spectral power of the GPi input patterns (A), excitatory input (B) and corresponding TC cell response (C). (D) ISI profile of the TC Cells. (E) Correlation susceptibility for $T = 95ms$ (F) Susceptibility $S$ of the TC cells based on the analysis window $T$.

#### Firing patterns

Validating the generated GPi input spike trains was completed by the spectral power analysis presented in Figure 9.14A. As in Reitsma et al. (2011) the Oscillatory and Oscillatory

Bursty patterns have clear spectral peaks at $10$ Hz, while the Normal and Bursty cases have no obvious peak. As expected the cortical inputs lack a peak in the frequency range of interest (see Figure 9.14B).

The parameters for the model were selected based on the TC cells firing patterns and spectral analysis. Although the Normal and Bursty spectral powers do peak around $10$ Hz there are oscillations present in both (see Figure 9.14C). This is contrasted with the cleaner peaks of the Oscillatory and Oscillatory Bursty cases. This is however, consistent with the spectral analysis reported by Reitsma et al. (2011). The discrepancies are likely due to analysis parameters and the way GPi inputs were generated, as discussed below.

There is a clear bimodality to the interspike interval histogram of Figure 9.14D, which is consistent with the original work. However the first peak, at $10$ ms, is lower than the $30$ ms peak described by Reitsma et al. (2011). This may be a product of that model using a refractory period of $5$ ms, possibly resulting in slower bursts. It may also be a product of the way the dynamical correlate of the T-current is produced in the hybrid model. This would cause the inputs to recruit the bursting regime of the model in a different or perhaps less efficient way than the IFB or conductance based models used in Reitsma et al. (2011).

Despite the slight differences the firing patterns of the hybrid model in this network are still in general agreement with Reitsma et al. (2011). In that work it was stated that without the T-current these firing characteristics were lost. Suggesting that the mechanism for bursting in the hybrid model is sufficient to reveal these characteristics. In addition, the power spectrum peaks suggest a frequency selectivity that is observed in experimental paradigms and was suggested to be dependent on the T-current (Reitsma et al., 2011).

**Correlation susceptibility**

The general susceptibility analysis, Figure 9.14F, qualitatively matches the results of Reitsma et al. (2011) however, the magnitude of the steady-state values are consistently lower than the original work. Similar results were found for our implementation of the IFB model, Figure 9.15, suggesting that the discrepancy in the magnitude of the susceptibility may arise due to differences in the way the input signals are generated. The correlation coefficients for $T = 95$ ms, Figure 9.14E, when fit with a linear curve illustrates the different slopes produced by the four input patterns tested. The Bursty and Oscillatory Bursty cases here have input correlation coefficients that are always greater than zero, even when $c = 0$. This is a product of generating the spike trains using a common time-dependent rate function.

In the work of Reitsma et al. (2011) the susceptibility values reach an asymptote around $T = 200$ ms. Here for the Normal and Oscillatory cases that plateau is reached much earlier, around $T = 100$ ms. The implications of this are unclear but they do not appear to affect the conclusion that the bursty inputs cause an increase in correlation susceptibility. In addition, this further supports the conclusion that the correlation results of Reitsma et al. (2011) are independent of model details. That combined with the fire pattern results above, helps validate the use of the simple hybrid model in correlation studies.

## IFB Response

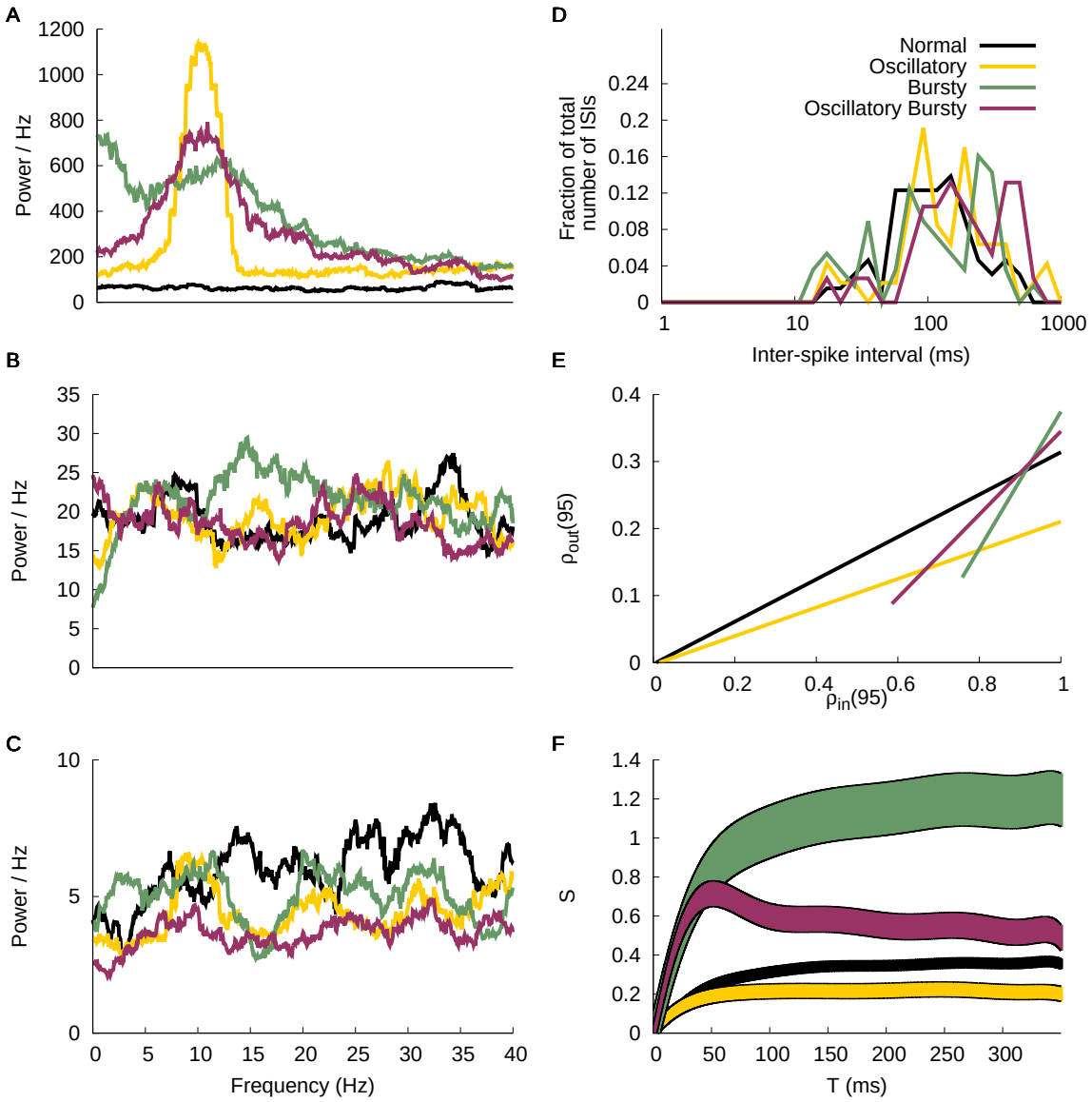



**Figure 9.15:** IFB model correlation analysis. Spectral power of the GPi input patterns (A), excitatory input (B) and corresponding TC cell response (C). (D) ISI profile of the TC Cells. (E) Correlation susceptibility for $T = 95ms$ (F) Susceptibility $S$ of the TC cells based on the analysis window $T$.

Using the same parameters of (Reitsma et al., 2011) for the IFB neuron model and the input patterns presented here, we were unable to recreate the same spectral and ISI patterns as the original paper Figure 9.15B and C. It is unclear where the difference arises from and

communications with the original authors did not reveal a cause. Despite these different responses, the correlation susceptibility results, Figure 9.14 E and F, were extremely close to the Izhikevich model results as well as the original work .

## 9.4 Discussion

### 9.4.1 Previous BG models utilizing the hybrid neuron

There are number of spiking neural models of the basal ganglia. However, only a small subset of those make use of the simple hybrid model and none use parameters that produce dynamics faithful to the nuclei they are modeling. For example the model presented by Igarashi et al. (2011) utilized the hybrid model for neurons only in the striatum. However, they use the expanded form of the hybrid equation. The other BG nuclei are modeled using a conductance-based integrate-and-fire model. Modolo et al. (2007) incorporated the simple Izhikevich neuron into a population based model but the neuron parameters were selected to achieve single neuron dynamics within the tonic regime described by Izhikevich (2003). Latteri et al. (2011) explored the synchronization characteristics of a population of coupled neurons. A single type of Izhikevich model was used with the simulations matching both experimental results and model results based on the Morris Lecar neuron model. Although this model has been shown to phenomenologically replicate the spiking dynamics of most neuron types, there have been no spiking models of the basal ganglia that attempt to use the hybrid model parameters that produce single neuron dynamics of the BG nuclei.

### 9.4.2 Physiological model of action selection

An interesting result of this work that was absent from Humphries et al. (2006) was the use of a neuron model that could replicate experimental dynamics at both the single neuron and population levels. Even with the added current source the LIF neuron employed by Humphries et al. (2006) is unable to completely replicate the complex fire patterns presented in Figure 9.2. It was argued that the most relevant dynamics are included and given that the model of Humphries et al. (2006) was able to replicate experimental results, it can be argued that the individual neuron dynamics may not be necessary. However, as illustrated by the results in Figure 9.9, the model presented here was able to not only selected the most salient input but also drive the activity of the previously activated channel clearly away from the selection limit. The selection results presented by Humphries et al. (2006) as well as by independent testing of the model (not presented) demonstrated a sufficient but modest increase in the activity of the previously selected channel. The increased activity of our model is large enough to push the previous channel back to its basal level of firing; reducing the possibility of selecting undesired or multiple channels. The mechanism for the improved selection capabilities is unclear and remains a focus of future studies. In addition, in the future this model will be extended to include a larger number of channels to determine how feasible it is to scale beyond the three presented here.

The original rate based model of Gurney et al. (2001) was converted into the spiking domain by Stewart et al. (2010) using LIF neurons and the Neural Engineering Framework (Eliasmith and Anderson, 2003). It was then expanded to include both action selection and reward learning (Stewart et al., 2012). The combination of action-selection and reinforcement-learning is another aspect of this model we plan to explore.

### 9.4.3 The Parkinsonian BG

Rubin and Terman (2004) offered the first explanation for the paradoxical therapeutic effects of DBS in a Parkinsonian BG. The simplicity of the model combined with the comprehensive mathematical analysis has made it a seminal work in BG modeling. The data driven extension of this work presented by Guo et al. (2008) further supported these results and linked its theories to experimental recordings. A similar extension was performed by Meijer et al. (2011) where the relay fidelity of a single TC neuron in response to different DBS parameters was explored. Similarly, Dorval et al. (2010) used the RT model to compliment human subject experiments exploring the regularity of DBS inputs.

The majority of these studies supported the theory that oscillatory inputs into the thalamus from the GPi negatively affect relay fidelity of the thalamus. In addition, constant inputs from the GPi, arising from DBS application, result in more effective relay in the thalamus (Rubin et al., 2012).

There have been a number of studies that have extended the RT model to explore the therapeutic effects of different DBS locations, protocols and strategies (Hahn and McIntyre, 2010; Guo and Rubin, 2011; Agarwal and Sarma, 2012), as well as closed loop configurations (Feng et al., 2007) and medicated states (Frank, 2005). Similarly, the inverse relationship between frequency and stimulus amplitude in clinically effective DBS has been explored with the RT Model (Cagnan et al., 2009). In addition, the effects of Parkinsonian symptoms and reduced levels of dopamine on action selection have been researched with the RT model (Leblois et al., 2006; Pirini et al., 2009).

## Future Work

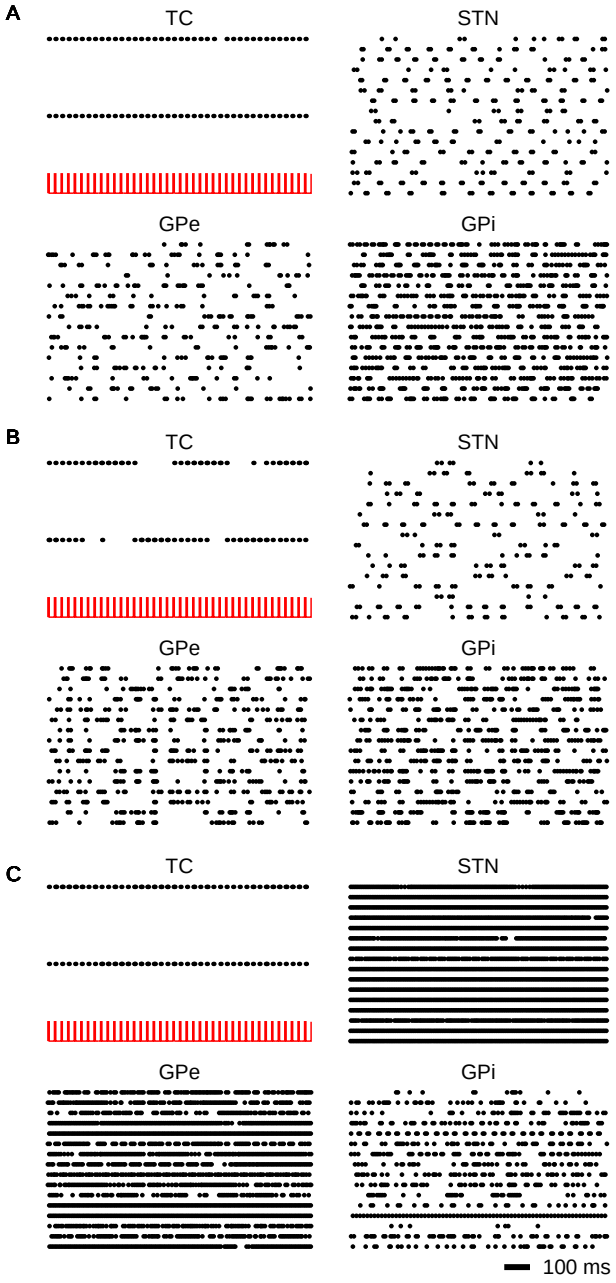



**Figure 9.16:** Simulated recovery of TC relay fidelity with 6Hz Parkinsonian activity. (A) Under normal BG activity the thalamus is capable of relaying somatomotor inputs. (B) Under Parkinsonian conditions the BG nuclei fall into oscillatory firing patterns TC relay capabilities are greatly diminished. (C) Application of DBS to the STN restores lost TC relay fidelity.

Preliminary searches of the parameter space have revealed that this model can replicate a number of different states that match experimental results without modification to the network structure. Figure 9.16 demonstrates a model with oscillations around $6$ Hz. The error analysis, Figure 9.17, reveals similar results to the $4$ Hz model presented here. The concept that the parameter choice can move this model into different pathological conditions is novel in basal ganglia modeling. We plan to explore this further in future studies. The results of which could in fact merge previously divergent theories of the nature of Parkinson's disease or possibly reveal serious deficiencies in these network models.

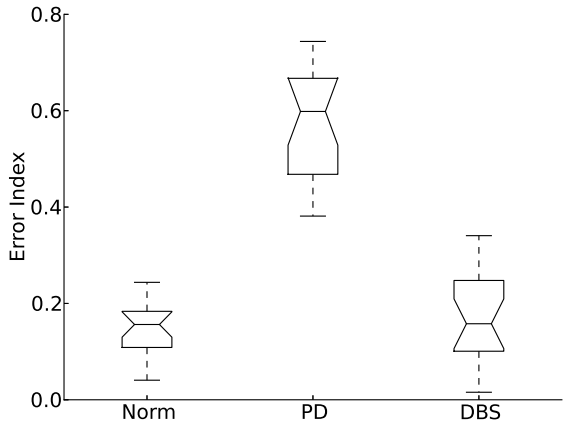



**Figure 9.17:** Error index statisitcs for 6Hz model. Allowing the network connection weights to randomly change over 20 simulations results in the Normal and DBS modes operating with less errors then the PD mode.

### 9.4.4 Thalamic relay fidelity between the BG and thalamus

The correlation study of Reitsma et al. (2011) highlighted that a number of point neuron models were capable of demonstrating how the pattern of firing in the GPi could affect correlation transfer in the thalamus. With this aspect of the study a similar result, that firing patterns observed in the Parkinsonian BG result in increased correlation susceptibility of the thalamus was found. This could provide an explanation for some of the pathological

hallmarks of Parkinson's disease.

Although it was shown that the T-current, required for TC neuron bursting, is responsible for the spike pattern of the model, it does not appear to have an effect on the correlation transfer (Reitsma et al., 2011). Here however, we were able to demonstrate both similar spiking patterns as well as similar correlation susceptibility as the models with higher biological fidelity. These results open up a number of future studies employing the hybrid model. This includes a frequency space analysis of the correlation transfer as well as a more thorough mathematical analysis of the relationship between GPi inhibition and spike correlation.

### 9.4.5 BG models in neuromorphic hardware

The complexity of the neuron models explored in the original studies require a level population specificity that is undesirable in generic hardware implementations. Although the LIF neurons of Humphries et al. (2006) are ideal for neuromorphic hardware the gated synaptic currents as well as the piecewise calcium currents would require circuitry specific to a nuclei type and would greatly diminish the utility of a hardware system.

The motivations for embedding BG models in hardware systems go beyond the obvious applications to intelligent agents and neurorobotics. It has been shown that the model based control concepts introduced in Section 9.1 have a number of clinical and practical applications (Schiff, 2012). In addition to the control system computations are the numerical calculations required for simulating the model aspect of the observer. Combining the control system with neuromorphic hardware, perhaps in a system on chip, would greatly reduce the power consumption and provide a solution appropriate for portable realization. As emphasized in Schiff (2010), even if the results of closing the loop are a significant reduction in battery life the model-based paradigm would be beneficial. Ideally, extended

battery life will be accompanied by clinical improvements and studies cited here support the presence of both in closed-loop strategies.

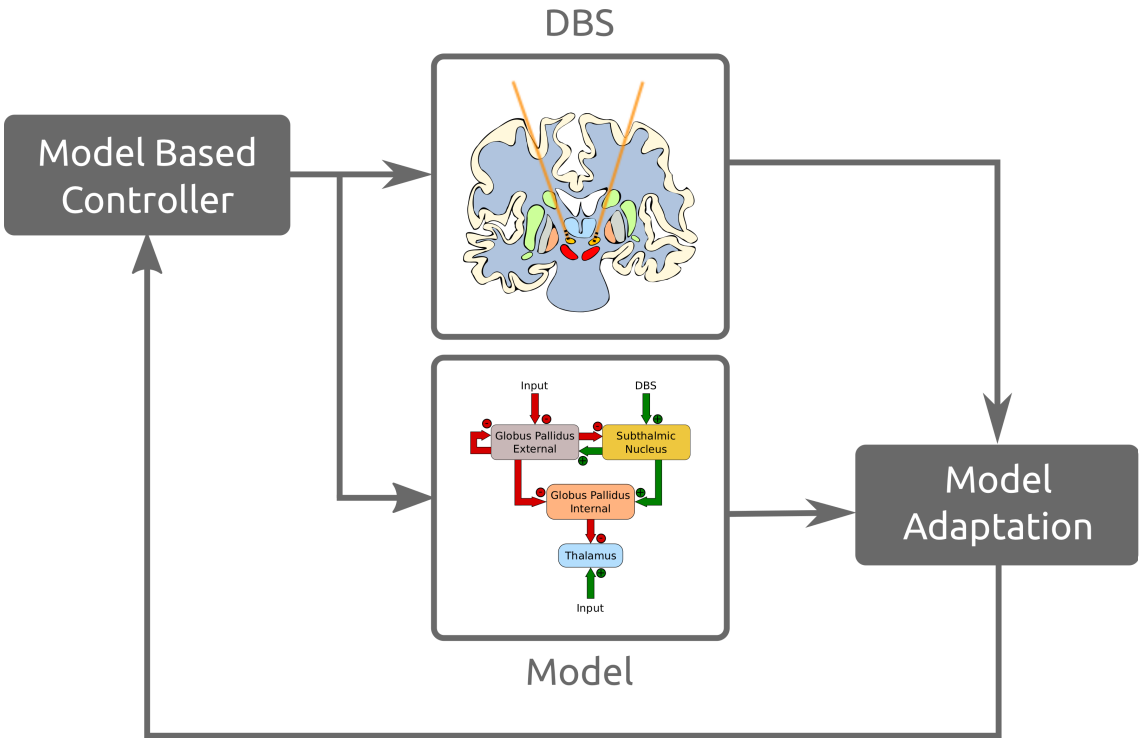



**Figure 9.18:** Simplified example of how these models fit in a model based control DBS paradigm.

Model based or model predictor control systems work as state estimators where the dynamics of the model are used to predict the state of the current system. That prediction is then corrected with new measurements. These allows us to incorporate the predictions of the system's state as well as sensor estimates with the real sensor information to get a better estimate of the actual state. Figure 9.18 is a simplified overview of how these models would fit into such a control system. This is a brief example of how these models and neuromorphic hardware fit in with model based control strategies, for a more extensive review see Schiff (2012).

To quantify the energy efficiency of neuromorphic hardware, power estimates were computed for the models presented here. The details of how these were computed are

included in Appendix 1. The comparison with three commercial off-the-shelf components, Figure 9.19, illustrates the tremendous energy advantage neuromorphic hardware has over the current hardware.

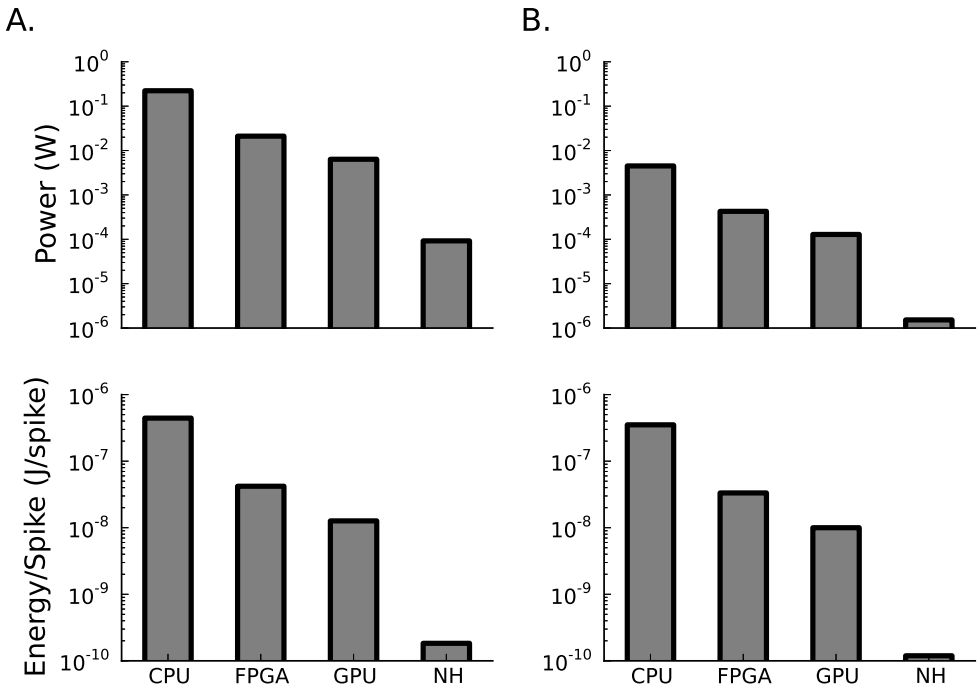



**Figure 9.19:** Power estimate for a model on a particular hardware, top. Energy per spike for the model, bottom. (A) Estimates for Action selection model of Section 9.2.4. (B) Estimates for Parkinsonian model of Section 9.2.5. The hardware used for comparison was standard central processing unit (CPU), field programmable gate arrays (FPGA),graphical processing unit (GPU) and sample neuromorphic hardware (NH). The neuromorphic hardware is consistently lower in both and power and and energy per spike for both neural models. The details for this are included in Appendix B.

There are a number of issues, however, beyond implementation difficulties that need to be resolved before model-based control strategies will prove useful. The level of realism required in the neuron model is still unclear at this point. Schiff (2010) was able to demonstrate model-based control of DBS using the simple neuron implementation of Rubin and Terman (2004). Although computationally cheaper than the full conductance based models this still suffers from the problems discussed above. A logical next step in this work

will be to show that the simple hybrid neuron can also be effective in model-based control strategies of DBS.

These strategies may also prove efficacious in brain computer interfaces (BCI). Rather than contributing to the dynamic changes in brain dynamics, BCI applications would be used in estimating state and decoding measurements. This is a concept that, although promising, has proven difficult to achieve (Schiff, 2012). Low-power realizations of these systems, as suggested here, offer a cost-effective option as BCI theories mature.

Finally, the most important point on the study of neural control engineering is that often the best model is not the most physiological one, but the one that best reduces error (Schiff, 2012). This is important because focusing too much on model adequacy may take away from the more important task of producing better therapies. An important question that will need to be answered in this case is, how detailed does a BG network model need to be in order to prove effective in estimating pathological conditions? The next step in this work is to begin developing strategies based on the these models and the control theoretic approaches of Voss et al. (2004), Schiff (2010), and Schiff (2012).

Ultimately, until models are capable of predicting therapeutic outcomes, either through realistic biological results or through a dimensionality reduced interpretations, the pathological BG models will remain just a compliment to physiological experiments.

### 9.4.6 Conclusions

The models utilizing the simple hybrid neuron presented here may offer a mechanism for revealing mathematical details of BG function and dysfunction that are hidden by the complexity of other models. An immediate extension that highlights that concept is in the parameter exploration of the RT model. The computational efficiency of the network presented in Section 9.2.5 has allowed us to begin exploring the parameter space using a

commodity computing cluster. Sweeps can be completed in hours as opposed to months of computing it would take to explore the original RT Model. We hope to present details of this in future publications.

Using the simple hybrid neuron in such small networks and deriving biologically significant meaning from them can be unreliable. Care must be taken when interpreting the results in the context of both pathological conditions as well as clinical therapies. The traditional niche for the simple model has really been in large-scale modeling. The more biologically realistic conductance based neuron models are generally recommended for single and small-scale network studies (Izhikevich, 2007a). In addition to those presented above, one of the primary motivations for using the simple model lie in the intention to construct large-scale models of the BG. This work presents the foundations for those future studies and the results demonstrate that the hybrid model is capable of capturing many of the relevant BG responses and dynamics. These studies are meant to compliment experimental research as well as the more detailed modeling efforts.

# Chapter 10

# Discussion and Future Directions

Future extensions of this work will focus on exploring how the anatomy that has been historically ignored in basal ganglia models can influence the reward related firing leading to dynamic models of action-selection. This includes the context specific firing of the pedunculopontine tegmental nucleus onto the dopaminergic systems of the basal ganglia as well as a new theories of the functional anatomy of the basal ganglia. The combination of theories that will be integrated are outlined below.

## 10.1 The role of dopamine as a neuromodulator

At the D1 receptors dopamine plays a contrast enhancer role, suppressing activity in less active cells while enhancing activity in already active cells (Cohen and Frank, 2009). However, at D2 type receptors dopamine has an inhibitory effect. The SNc has a tonic level of activity that creates a constant release of dopamine on the striatum. That release is phasically enhanced by unexpected reward and reduced when an expected reward does not arrive (Schultz et al., 1997). One theory for how these dynamic changes in dopamine

release can affect reward learning rely on the parallel pathways of the traditional model (Cohen and Frank, 2009). Considering this, during an unexpected reward the phasic increase in dopamine will enhance the response of active neurons in the “Go” pathway, thus encouraging long-term potentiation through Hebbian synaptic plasticity. While the less active neurons of the D1 “Go” pathway will be further depressed by long-term depression mechanisms. Similarly, during drops in dopaminergic neuron activity the neurons of the “No-Go” pathway will have a reduction in dopamine modulated inhibition at D2 receptors. This increase in activity promotes long-term potentiation.

There are a number of inconsistencies in the current understanding of dopamine’s role in RL. Two in particularly are relevant here. (i) The stereotyped response and latency of phasic dopamine release has an on-rate and duration that is faster than it takes to both attend to and appreciate a rewarding stimulus (Redgrave et al., 2010). This implies that the dopaminergic response will have finished before the rewarding stimulus can be acknowledged. (ii) Dopamine neurons also produce a phasic response to unexpected stimuli that, although novel, have no reinforcement consequences (Redgrave et al., 2010). In addition, the afferent information from the superior colliculus may be better suited for assessing the agency of a stimulus as well as the novelty of the actions that resulted in an unpredicted stimulus, rather than a processed representation of it(Redgrave and Gurney, 2006). Given that sensory information, and the level of processing the superior colliculus is capable of providing to the the midbrain dopaminergic neurons, it is more likely that they will be notified that an event has occurred as opposed to the details of that event. This theory implies that the BG is involved in predicting the error in sensory information and whether or not the agent was responsible for a discrepancy in that prediction. The concept of “sensory prediction error” places less burden on the sensory processing areas than the “reward prediction error” hypothesis.

## 10.2 New functional anatomy

The early models of the BG organized the flow of information in closed parallel paths. The BG was considered a pass-through structure where information was processed and returned to the signaling cortical area. Recent studies have shown that the BG can no longer be thought of this way (Obeso and Lanciego, 2011). This is supported by the acceptance of the hyper-direct pathway and the subcortical innervations described below. In addition, recent experimental evidence has shown that the connections from STN to GPi may not have the sharpened projections that are a hallmark of the traditional BG model. Rather then the GPe transmitting its influence by way of the STN it is now believed that information is translated directly to the GPi through more focused efferent connections. In addition, the STN influence on the GPi is now believed to act as a global modulator due its more diffuse projections to GPi (Cohen and Frank, 2009). The dopaminergic afferents into the other structures of the BG also runs counter to the traditional BG model (Obeso and Lanciego, 2011).

## 10.3 Subcortical connections

There are large number of subcortical connections with the BG that have historically been neglected in neurocomputational models. Decerabrate rats, where the entire brain behind the superior colleculus is removed, still exhibit simple action selection; such as performing coordinated feeding movements Humphries et al. (2007). This action selection capability in the absence of the major BG afferents suggests that there are subcortical correlates to simple action selection. In addition, the role of other subcortical influences has been demonstrated empirically Winn et al. (2010); Norton et al. (2011)

### 10.3.1 Pedunculopontine tegmental nucleus

The pedunculopontine tegmental nucleus (PPN) is a sub-cortical structure often associated with orienting movement(Gerfen and Bolam, 2010). It has connections to both the brain-stem and the basal ganglia (Winn et al., 2010). The reciprocal connections to the BG are made to the striatum by way of the different thalamic nuclei and the dopaminergic nuclei of the VTA and SNC, as well as directly to the BG nuclei that include, SNr, STN, GPe and GPi. The PPN has been shown to encode reward stimulus in context-dependent manner (Norton et al., 2011). However, unlike the VTA and SNc the PPN does not appear to encode the prediction of a reward but rather its consumption (Norton et al., 2011). In addition, the duration of PPN reward response can be as long as 1000ms compared to the 70-100ms bursts observed in the SNc and VTA (Norton et al., 2011). The response of PPN neurons also appears to be dependent on the context transmitted by sensory systems (Norton et al., 2011).

### 10.3.2 Lateral habenula

The lateral habenula (LHb) is an epithalamic structure that is associated with the negative control of motivation (Bromberg-Martin and Hikosaka, 2011; Lavezzi and Zahm, 2011). Appropriately, the LHb shows a decrease in basal levels of firing in response to rewards and reward-predicting cues and increases firing in response to noxious or omitted stimulus (Bromberg-Martin and Hikosaka, 2011; Lavezzi and Zahm, 2011). Neurons in the LHb has been shown to respond to the prediction of reward-information sensory cures as well as errors in primary rewards themselves (Bromberg-Martin and Hikosaka, 2011). In addition, it has been shown that LHb neurons would increase activity when it was learning that a reward was being withheld (Bromberg-Martin and Hikosaka, 2011). It has long

been theorized that the inhibitory projection neurons of the LHb supress the activity of the dopaminergic neurons of the midbrain (Lavezzi and Zahm, 2011). However, there are a number of timing and response characteristics that run counter to that theory. In addition, the majority of LHb neurons that do project to the midbrain are glutamatergic (Lavezzi and Zahm, 2011). It wasn't until the discovery of the mesopontine restromedial tegmental nucleus that these observations had an explanation.

### 10.3.3 Mesopontine restromedial tegmental nucleus

The mesopontine restromedial tegmental nucleus (RMTg) is newly discovered structure located behind the VTA. It is densely packed with GABAergic inhibitory neurons that receive dense glutamatergic connections from the LHb. The major output projections of the RMTg are to the VTA and SNc. In addition, there are robust connections to the PPN, dorsal raphe nucleus, pontine and medullary reticular formation (Lavezzi and Zahm, 2011). The current theory is that the RMTg acts as a functional inhibitory relay for the LHb, among other areas. This is support by investigations demonstrating that the RMTg responds to adverse stimuli but does not respond to rewarding stimuli (Lavezzi and Zahm, 2011).

### 10.3.4 The agency hypothesis in reward learning

The idea that the BG is involved with reinforcement learning (RL) is most often attributed to the seminal work of Schultz *et al.* (1997)(Schultz et al., 1997). In that work the phasic changes in tonic firing of dopaminergic neurons in response to unexpected, noxious or expected but absent stimuli, was observed in the superior colliculus. This has become an accepted basis of RL in neuroscience (Redgrave et al., 2010). However, Redgrave and colleagues argue that there is enough contradictory evidence to support a reevaluation of

that idea(Redgrave and Gurney, 2006; Redgrave et al., 2008).

The agency hypothesis, discussed above, proposes that the BG circuitry is more suited to determining if the agent is the likely cause of the unpredicted event. Then through neural plasticity the appropriate novel actions can be reinforced (Redgrave et al., 2010). In this theory there are three afferent signals to the striatum. (i) Sensory information is projected to both the SNc as well as the striatum by way of the thalamus. Given the timing of these two signals they should converge on the striatum. (ii) Contextual information is sent from the higher cortical structures to communicate the current state of the agent. (iii) There is a motor-copy that is sent via fibers that connect to motor and pro-motor areas in the medulla and spinal cord.

This hypothesis fits with the well known timing of dopamine release in the BG. However, it fails to provide a satisfactory answer for where the drop in dopamine activity initiated during harmful or noxious stimuli. Part of this future work is to explore the possible role of the subcortical structures described above to see if they can fill this theoretical gap in the agency hypothesis.

## 10.4 Bringing sensory information together

In the traditional model of the basal ganglia sensory feedback is communicated through the SNc only. However, this can only account for phasic responses to unexpected rewarding stimuli. By introducing the sub-cortical structures reviewed above this aim is attempting to elucidate the neural mechanisms behind those unexplained responses. Given the reliance of the agency hypothesis on sensory information and the empirical evidence that the sub-cortical structures described above all participate in the relaying of sensory information, the overall performance of the RL/AS system should be improved. This evidence can be

summarized as:

- The LHb responds to errors in the prediction of reward-information in sensory cues Bromberg-Martin and Hikosaka (2011).
- RMTg relays the error information from sensory cues processed by the LHb (Lavezzi and Zahm, 2011).
- The reward correlates of the PPN appear to be sensory based (Norton et al., 2011).

The usefulness of this is supported by the reliance the agency hypothesis has on sensory feedback. The combination of the proposed functional anatomy, as well as subcortical connections, with novel theories of reward-learning is unique to this project. The computational complexity of the agency hypothesis is not only consistent with current understanding of RL but also facilitates using this model in an embodied agent.

## 10.5 The Integration of action-selection and reinforcement-learning

These concepts will be incrementally merged with the models presented in this paper. First the model of Section 9.2.4 will be modified to emphasize the new functional anatomy described above. The more prominent roles of the GPe and STN should improve the overall stability and extensibility of the model.

The simple models from Chapter 8 will then be extended to explore the agency hypothesis. Although these models ignore the full structure of the BG the integration of sensory, motor and contextual information can be included. Such simple models will make the task of exploring, tuning and validating the agency hypothesis more tractable.

The subcortical afferents of the LHb, RMTg and PPN will integrated into both of these

modeling tasks. This will provide increased stability as well as a neural correlate for punishment signals; improving overall task performance.

# Bibliography

Abarbanel, H. D. I., D. R. Creveling, and J. M. Jeanne (2008, Jan). Estimation of parameters in nonlinear systems using balanced synchronization. *Phys. Rev. E 77*, 016208.

Agarwal, R. and S. Sarma (2012). The effects of dbs patterns on basal ganglia activity and thalamic relay. *Journal of Computational Neuroscience 33*, 151–167.

Aprasoff, J. and O. Donchin (2012). Correlations in state space can cause sub-optimal adaptation of optimal feedback control models. *Journal of Computational Neuroscience 32*, 297–307.

Arena, P., L. Fortuna, M. Frasca, and L. Patane (2009, feb.). Learning anticipation via spiking networks: Application to navigation control. *Neural Networks, IEEE Transactions on 20*(2), 202 –216.

Arent, A. and T. Costa (2012). Artemis entity system framework. `http://www.gamadu.com/artemis/`. Accessed: 8/15/2012.

Arthur, J., P. Merolla, F. Akopyan, R. Alvarez, A. Cassidy, S. Chandra, S. Esser, N. Imam, W. Risk, D. Rubin, R. Manohar, and D. Modha (2012, june). Building block of a programmable neuromorphic substrate: A digital neurosynaptic core. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pp. 1 –8.

Atallah, H. E., M. J. Frank, and R. C. O'Reilly (2004). Hippocampus, cortex, and basal ganglia: Insights from computational models of complementary learning systems. *Neurobiology of Learning and Memory 82*(3), 253 – 267.

Barr, D., P. Dudek, J. Chambers, and K. Gurney (2007, aug.). Implementation of multilayer leaky integrator networks on a cellular processor array. In *Neural Networks, 2007. IJCNN 2007. International Joint Conference on*, pp. 1560 –1565.

Bergman, H., A. Zaidel, B. Rosin, M. Slovik, M. Rivlin-Etzion, S. Moshel, and Z. Israel (2010). Pathological synchrony of basal ganglia-cortical networks in the systemic mptp primate model of parkinson's disease. In H. Steiner and K. Y. Tseng (Eds.), *Handbook of Basal Ganglia Structure and Function*, Volume 20 of *Handbook of Behavioral Neuroscience*, pp. 653 – 658. Elsevier.

Bezard, E., G. Porras, J. Blesa, and J. A. Obeso (2010). Compensatory mechanisms in experimental and human parkinsonism: Potential for new therapies. In H. Steiner and K. Y. Tseng (Eds.), *Handbook of Basal Ganglia Structure and Function*, Volume 20 of *Handbook of Behavioral Neuroscience*, pp. 641 – 652. Elsevier.

Bilas, S. (2007). A data-driven game object system. `http://scottbilas.com/files/2002/gdc%5Fsan%5Fjose/game_objects_slides.pdf`. Accessed: 8/15/2012.

Boahen, K. A. (2000). Point-to-Point Connectivity Between Neuromorphic Chips Using Address-Events. *IEEE Transactions on Circuits and Systems II 47*(5), 416–34.

Bokil, H., P. Andrews, J. E. Kulkarni, S. Mehta, and P. P. Mitra (2010). Chronux: A platform for analyzing neural signals. *Journal of Neuroscience Methods 192*(1), 146 – 151.

Bolam, J., J. Hanley, P. Booth, and M. Bevan (2000, MAY). Synaptic organisation of the basal ganglia. *JOURNAL OF ANATOMY 196*(Part 4), 527–542.

BotPrize (2012). The 2K BotPrize. `http://botprize.org/index.html`. Accessed: 10/16/2012.

Bower, J. and D. Beeman (1998). *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System* (2 ed.). New York: Springer-Verlag.

Bower, J., D. Beeman, and M. Hucka (2002). *The Handbook of Brain Theory and Neural Networks*, Chapter GENESIS Simulation System, pp. 475–478. Cambridge, MA: The MIT Press.

Brette, R., M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. Bower, M. Diesmann, A. Morrison, P. Goodman, F. Harris, M. Zirpe, T. Natschlager, D. Pecevski, B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Vieville, E. Muller, A. Davison, S. El Boustani, and A. Destexhe (2007). Simulation of networks of spiking neurons: a review of tools and strategies. *Journal of computational neuroscience 23*(3), 349–398.

Bromberg-Martin, E. S. and O. Hikosaka (2011). Lateral habenula neurons signal errors in the prediction of reward information. *Nat Neurosci 14*(9), 1209 – 1216.

Brown, J. W., D. Bullock, and S. Grossberg (2004). How laminar frontal cortex and basal ganglia circuits interact to control planned and reactive saccades. *Neural Networks 17*(4), 471 – 510.

Burgsteiner, H. (2006). Imitation learning with spiking neural networks and real-world devices. *Engineering Applications of Artificial Intelligence 19*(7), 741 – 752.

Burkitt, N. (2006, June). A review of the integrate-and-fire neuron model: I. homogeneous synaptic input. *Biol. Cybern. 95*, 1–19.

Cagnan, H., H. G. E. Meijer, S. A. Van Gils, M. Krupa, T. Heida, M. Rudolph, W. J. Wadman, and H. C. F. Martens (2009). Frequency-selectivity of a thalamocortical relay neuron during parkinsons disease and deep brain stimulation: a computational study. *European Journal of Neuroscience 30*(7), 1306–1317.

Chakravarthy, V., D. Joseph, and R. Bapi (2010). What do the basal ganglia do? a modeling perspective. *Biological Cybernetics 103*, 237–253.

Chang, S.-Y., Y. M. Shon, F. Agnesi, and K. Lee (2009, sept.). Microthalamotomy effect during deep brain stimulation: Potential involvement of adenosine and glutamate efflux. In *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE*, pp. 3294 –3297.

Chorley, P. and A. K. Seth (2011). Dopamine-signalled reward predictions generated by competitive excitation and inhibition in a spiking neural network model. *Frontiers in Computational Neuroscience 5*(0).

Cohen, M. (2008). Neurocomputational mechanisms of reinforcement-guided learning in humans A review. *Cognitive, Affective, & Behavioral Neuroscience 8*, 113–125.

Cohen, M. R. and A. Kohn (2011, Jul). Measuring and interpreting neuronal correlations. *Nat Neurosci 14*(7), 811–819.

Cohen, M. X. and M. J. Frank (2009). Neurocomputational models of basal ganglia function in learning, memory and choice. *Behavioural Brain Research 199*(1), 141 – 156.

Crook, S., P. Gleeson, F. Howell, J. Svitak, and R. Silver (2007). Morphml: level 1 of the neuroml standards for neuronal morphology data and model specification. *Neuroinformatics 5*(2), 96–104.

Cruz-Albrecht, J., M. Yung, and N. Srinivasa (2012, june). Energy-efficient neuron, synapse and stdp integrated circuits. *Biomedical Circuits and Systems, IEEE Transactions on 6*(3), 246 –256.

DARPA (2012). Synapse broad agency announcement (baa). Available: `https://www.fbo.gov/spg/ODA/DARPA/CMO/BAA08-28/listing.html`.

Davison, A. P., D. Brderle, J. M. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger (2009). Pynn: a common interface for neuronal network simulators. *Frontiers in Neuroinformatics 2*(11).

Dayan, P. and L. F. Abbott (2005). *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press.

de Camargo, R. Y., L. Rozante, and S. W. Song (2011). A multi-gpu algorithm for large-scale neuronal networks. *Concurrency and Computation: Practice and Experience 23*(6), 556–572.

de la Rocha, J., B. Doiron, E. Shea-Brown, K. Josic, and A. Reyes (2007, Aug). Correlation between neural spike trains increases with firing rate. *Nature 448*(7155), 802–806.

Demirkol, A. and S. Ozoguz (2011, june). A low power vlsi implementation of the izhikevich neuron model. In *New Circuits and Systems Conference (NEWCAS), 2011 IEEE 9th International*, pp. 169 –172.

Diesmann, M. and M. Gewaltig (2002). Nest: An environment for neural systems simulations. *Forschung und wisschenschaftliches Rechnen, Beiträge zum Heinz-Billing-Preis 58*(43-70).

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

Djurfeldt, M. and A. Lansner (2007). Workshop report: 1st incf workshop on large-scale modeling of the nervous system. *Available from Nature Precedings*.

Dorval, A. D., A. M. Kuncel, M. J. Birdno, D. A. Turner, and W. M. Grill (2010). Deep brain stimulation alleviates parkinsonian bradykinesia by regularizing pallidal activity. *Journal of Neurophysiology 104*(2), 911–921.

Duan, B., W. Wang, X. Li, C. Zhang, P. Zhang, and N. Sun (2011, dec.). Floating-point mixed-radix fft core generation for fpga and comparison with gpu and cpu. In *Field-Programmable Technology (FPT), 2011 International Conference on*, pp. 1 –6.

Eaton, J. W., D. Bateman, and S. Hauberg (2008). *GNU Octave Manual Version 3*. Network Theory Limited.

Eliasmith, C. and C. H. Anderson (2003). *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. Cambridge, MA: MIT Press.

Eppler, J., H. Plesser, A. Morrison, M. Diesmann, and M.-O. Gewaltig (2007). Multithreaded and distributed simulation of large biological neuronal networks. In F. Cappello, T. Herault, and J. Dongarra (Eds.), *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Volume 4757 of *Lecture Notes in Computer Science*, pp. 391–392. Springer Berlin / Heidelberg.

Fahn, S., D. Oakes, I. Shoulson, K. Kieburtz, A. Rudolph, A. Lang, C. Olanow, C. Tanner, and K. Marek (2004). Levodopa and the progression of parkinson's disease. *New England Journal of Medicine 351*(24), 2498–2508.

Fall, C., E. Marlan, J. Wagner, and J. Tyson (2002). *Computational Cell Biology*. IAM.

Fan, Z., F. Qiu, A. Kaufman, and S. Yoakum-Stover (2004, nov.). Gpu cluster for high performance computing. In *Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 Conference*, pp. 47.

Feng, X.-J., E. Shea-Brown, B. Greenwald, R. Kosut, and H. Rabitz (2007). Optimal deep brain stimulation of the subthalamic nucleusa computational study. *Journal of Computational Neuroscience 23*, 265–282.

Fidjeland, A. and M. Shanahan (2010, july). Accelerated simulation of spiking neural networks using gpus. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pp. 1 –8.

Florian, R. (2006). Spiking neural controllers for pushing objects around. In S. Nolfi, G. Baldassarre, R. Calabretta, J. Hallam, D. Marocco, J.-A. Meyer, O. Miglino, and D. Parisi (Eds.), *From Animals to Animats 9*, Volume 4095 of *Lecture Notes in Computer Science*, pp. 570–581. Springer Berlin Heidelberg.

Florian, R. V. (2007, 6). Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Computation 19*(6), 1468–1502.

Frank, M. J. (2005). Dynamic dopamine modulation in the basal ganglia: A neurocomputational account of cognitive deficits in medicated and nonmedicated parkinsonism. *Journal of Cognitive Neuroscience 17*(1), 51–72.

Friedrich, J., R. Urbanczik, and W. Senn (2011, 06). Spatio-temporal credit assignment in neuronal population learning. *PLoS Comput Biol 7*(6).

Furber, S., D. Lester, L. Plana, J. Garside, E. Painkras, S. Temple, and A. Brown (2012). Overview of the spinnaker system architecture. *Computers, IEEE Transactions on PP*(99), 1.

Gao, P., B. V. Benjamin, and K. Boahen (2012). Dynamical system guided mapping of quantitative neuronal models onto neuromorphic hardware. *Circuits and Systems I: Regular Papers, IEEE Transactions on PP*(99), 1.

Gerfen, C. R. and J. P. Bolam (2010). The neuroanatomical organization of the basal ganglia. In H. Steiner and K. Y. Tseng (Eds.), *Handbook of Basal Ganglia Structure and Function*, Volume 20 of *Handbook of Behavioral Neuroscience*, pp. 3 – 28. Elsevier.

Girard, B., N. Tabareau, Q. Pham, A. Berthoz, and J.-J. Slotine (2008). Where neuroscience and dynamic system theory meet autonomous robotics: A contracting basal ganglia model for action selection. *Neural Networks 21*(4), 628 – 641. Robotics and Neuroscience.

Group, K. O. W. et al. (2008). The opencl specification. *A. Munshi, Ed*.

Guo, Y. and J. E. Rubin (2011). Multi-site stimulation of subthalamic nucleus diminishes thalamocortical relay errors in a biophysical network model. *Neural Networks 24*(6), 602 – 616.

Guo, Y., J. E. Rubin, C. C. McIntyre, J. L. Vitek, and D. Terman (March 2008). Thalamocortical relay fidelity varies across subthalamic nucleus deep brain stimulation protocols in a data-driven computational model. *Journal of Neurophysiology 99*(3), 1477–1492.

Gurney, K., T. J. Prescott, and P. Redgrave (2001). A computational model of action selection in the basal ganglia. ii. analysis and simulation of behaviour. *Biological Cybernetics 84*(6), 411.

Haber, S. N. (2010). Integrative networks across basal ganglia circuits. In H. Steiner and K. Y. Tseng (Eds.), *Handbook of Basal Ganglia Structure and Function*, Volume 20 of *Handbook of Behavioral Neuroscience*, pp. 409 – 427. Elsevier.

Hahn, P. and C. McIntyre (2010). Modeling shifts in the rate and pattern of subthalamopallidal network activity during deep brain stimulation. *Journal of Computational Neuroscience 28*, 425–441.

Han, B. and T. M. Taha (2010). Acceleration of spiking neural network based pattern recognition on nvidia graphics processors. *Appl. Opt. 49*(10), B83–B91.

Hille, B. (2001, July). *Ion Channels of Excitable Membranes* (3 ed.). Sinauer Associates.

Hines, M. and N. Carnevale (2007). Translating network models to parallel hardware in neuron. *J. Neurosci. Methods 169*, 425–455.

Hines, M., S. Kumar, and F. Schurmann (2011). Comparison of neuronal spike exchange methods on a blue gene/p supercomputer. *Frontiers in Computational Neuroscience 5*(0).

Hines, M. L. and N. T. Carnavale (1997). The neuron simulation environment. *Neural Computation 9*(6), 1179–1209.

Holden, R. (1986). The contagiousness of aircraft hijacking. *American Journal of Sociology*, 874–904.

Humphries, M., K. Gurney, and T. Prescott (2007). Is there a brainstem substrate for action selection? *Philosophical Transactions of the Royal Society B: Biological Sciences 362*(1485), 1627–1639.

Humphries, M. D., R. D. Stewart, and K. N. Gurney (2006). A physiologically plausible model of action selection and oscillatory activity in the basal ganglia. *The Journal of Neuroscience 26*(50), 12921–12942.

id Software (2012). Wolfenstein 3D. `http://www.idsoftware.com/games/wolfenstein/wolf3d`. Accessed: 11/2/2012.

Igarashi, J., O. Shouno, T. Fukai, and H. Tsujino (2011). Real-time simulation of a spiking neural network model of the basal ganglia circuitry using general purpose computing on graphics processing units. *Neural Networks 24*(9), 950 – 960.

Izhikevich, E. (2003). Simple model of spiking neurons. *IEEE Transactions On Neural Networks 14*(6), 1569–1572.

Izhikevich, E. M. (2007a). *Dynamical Systems in Neuroscience*. Cambridge, MA: The MIT Press.

Izhikevich, E. M. (2007b). Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cerebral Cortex 17*(10), 2443–2452.

Izhikevich, E. M. (2010). Hybrid spiking models. *Philosophical Transactions of the Royal Society A: Mathematical,Physical and Engineering Sciences 368*(1930), 5061–5070.

Keener, J. and J. Sneyd (2008). *Mathematical Physiology: I: Cellular Physiology*, Volume 1. Springer.

Khan, M., D. Lester, L. Plana, A. Rast, X. Jin, E. Painkras, and S. Furber (2008). Spinnaker: mapping neural networks onto a massively-parallel chip multiprocessor. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pp. 2849–2856. IEEE.

Koch, C. and I. Segev (Eds.) (1998). *Methods in Neuronal Modeling*. Cambridge: The MIT Press.

Krichmar, J. (2008). Neurorobotics. `http://www.scholarpedia.org/article/Neurorobotics`.

Krichmar, J. L. and G. M. Edelman (2005, jan). Brain-based devices for the study of nervous systems and the development of intelligent machines. *Artif. Life 11*(1-2), 63–78.

Krishnan, R., S. Ratnadurai, D. Subramanian, V. Chakravarthy, and M. Rengaswamy (2011). Modeling the role of basal ganglia in saccade generation: Is the indirect pathway the explorer? *Neural Networks 24*(8), 801 – 813.

Latteri, A., P. Arena, and P. Mazzone (2011). Characterizing deep brain stimulation effects in computationally efficient neural network models. *Nonlinear Biomedical Physics 5*(1), 2.

Lavezzi, H. N. and D. S. Zahm (2011). The mesopontine rostromedial tegmental nucleus: An integrative modulator of the reward system. *Basal Ganglia 1*(4), 191 – 200.

Leblois, A., T. Boraud, W. Meissner, H. Bergman, and D. Hansel (2006). Competition between feedback loops underlies normal and pathological dynamics in the basal ganglia. *The Journal of Neuroscience 26*(13), 3567–3583.

Markram, H., J. Lbke, M. Frotscher, and B. Sakmann (1997). Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps. *Science 275*(5297), 213–215.

Markram, H., Y. Wang, and M. Tsodyks (1998). Differential signaling via the same axon of neocortical pyramidal neurons. *Proceedings of the National Academy of Sciences 95*(9), 5323–5328.

Mead, C. (1989). *Analog VLSI and neural systems*. Reading: Addison-Wesley.

Meijer, H. G. E., M. Krupa, H. Cagnan, M. A. J. Lourens, T. Heida, H. C. F. Martens, L. J. Bour, and S. A. van Gils (2011). From parkinsonian thalamic activity to restoring thalamic relay using deep brain stimulation: new insights from computational modeling. *Journal of Neural Engineering 8*(6), 066005.

Merolla, P., J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. Modha (2011, sept.). A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45nm. In *Custom Integrated Circuits Conference (CICC), 2011 IEEE*, pp. 1 –4.

Michel, O. (2004). Webots: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems 1*(1), 39–42.

Migliore, M., C. Cannia, W. Lytton, H. Markram, and M. Hines (2006). Parallel network simulations with neuron. *Journal of Computational Neuroscience 21*, 119–129.

Minkovich, K., N. Srinivasa, J. Cruz-Albrecht, Y. Cho, and A. Nogin (2012, june). Programming time-multiplexed reconfigurable hardware using a scalable neuromorphic compiler. *Neural Networks and Learning Systems, IEEE Transactions on 23*(6), 889 –901.

Minkovich, K., C. M. Thibeault, M. J. O'Brien, A. Nogin, Y. Cho, and N. Srinivasa (2012). HRLSim: High-performance GPGPU based spiking neural simulator for GPGPU clusters. *In Submission to Neural Networks and Learning Systems, IEEE Transactions on*.

Modolo, J., E. Mosekilde, and A. Beuter (2007). New insights offered by a computational model of deep brain stimulation. *Journal of Physiology-Paris 101*(1-3), 56 – 63. Neuro-Computation: From Sensorimotor Integration to Computational Frameworks.

Montgomery, E. (2012). The epistemology of deep brain stimulation and neuronal pathophysiology. *Frontiers in Integrative Neuroscience 6*(78).

Morrison, A., C. Mehring, T. Geisel, A. Aertsen, and M. Diesmann (2005, August). Advancing the boundaries of high-connectivity network simulation with distributed computing. *Neural Computation 17*(8), 1776–1801.

Nageswaran, J. M., N. Dutt, J. L. Krichmar, A. Nicolau, and A. V. Veidenbaum (2009). A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors. *Neural Networks 22*(5-6), 791 – 800. Advances in Neural Networks Research: IJCNN2009, 2009 International Joint Conference on Neural Networks.

Navaridas, J., M. Luján, J. Miguel-Alonso, L. A. Plana, and S. Furber (2009). Understanding the interconnection network of spinnaker. In *Proceedings of the 23rd international conference on Supercomputing*, ICS '09, pp. 286–295. ACM.

Nere, A., A. Hashmi, and M. Lipasti (2011, may). Profiling heterogeneous multi-gpu systems to accelerate cortically inspired learning algorithms. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pp. 906 –920.

Norris, G. (1996). *Boeing 777*. Zenith Press.

Norton, A. B. W., Y. S. Jo, E. W. Clark, C. A. Taylor, and S. J. Y. Mizumori (2011). Independent neural coding of reward and movement by pedunculopontine tegmental nucleus neurons in freely navigating rats. *European Journal of Neuroscience 33*(10), 1885–1896.

NVIDIA (2012). Whitepaper nvidia geforce gtx 680. `http://www.geforce.com/Active/en\_US/pdf/GeForce-GTX\-680-Whitepaper-FINAL.pdf`.

Obeso, J. A. and J. L. Lanciego (2011). Past, present and future of the pathophysiological model of the basal ganglia. *Frontiers in Neuroanatomy 5*(00039).

O'Brien, M. J. and N. Srinivasa (2013). A spiking neural model for stable reinforcement of synapses based on multiple distal rewards. *Neural Computation 25*, 123–156.

Oorschot, D. E. (2010). Cell types in the different nuclei of the basal ganglia. In H. Steiner and K. Y. Tseng (Eds.), *Handbook of Basal Ganglia Structure and Function*, Volume 20 of *Handbook of Behavioral Neuroscience*, pp. 63 – 74. Elsevier.

O'Reilly, R. C. (2006). Biologically based computational models of high-level cognition. *Science 314*(5796), 91–94.

Pascual, A., J. Modolo, and A. Beuter (2006). Is a computational model useful to understand the effect of deep brain stimulation in parkinson's disease? *Journal of Integrative Neuroscience 5*(4), 541 – 559.

Pecevski, D., T. Natschlger, and K. Schuch (2009). Pcsim: a parallel simulation environment for neural circuits fully integrated with python. *Frontiers in Neuroinformatics 3*(0).

Peck, C., J. Kozloski, A. Rao, and G. Cecchi (2003). Simulation infrastructure for modeling large scale neural systems. *Proceedings of the International Conference on Computer Science (ICCS) 2003*, 1127–1136.

Pirini, M., L. Rocchi, M. Sensi, and L. Chiari (2009). A computational modelling approach to investigate different targets in deep brain stimulation for parkinsons disease. *Journal of Computational Neuroscience 26*, 91–107.

Plesser, H., J. Eppler, A. Morrison, M. Diesmann, and M.-O. Gewaltig (2007). Efficient parallel simulation of large-scale neuronal networks on clusters of multiprocessor computers. In A.-M. Kermarrec, L. Bougé, and T. Priol (Eds.), *Euro-Par 2007 Parallel Processing*, Volume 4641 of *Lecture Notes in Computer Science*, pp. 672–681. Springer Berlin / Heidelberg. 10.1007/978-3-540-74466-5_71.

Purves, D., G. J. Augustine, D. Fitzpatrick, W. C. Hall, A.-S. LaMantia, J. O. McNamara, and L. E. White (2007). *Neuroscience, 4th edition*. New York, NY: Sinauer Associates.

Pyggel group (2012). Pyggel. `http://www.pygame.org/project-PYGGEL-968-.html`. Accessed: 10/22/2012.

Rangan, V., A. Ghosh, V. Aparin, and G. Cauwenberghs (2010). A subthreshold vlsi implementation of the izhikevich simple neuron model. In *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*, pp. 4164 –4167.

Redgrave, P., V. Coizet, and J. Reynolds (2010). Phasic dopamine signaling and basal ganglia function. In H. Steiner and K. Y. Tseng (Eds.), *Handbook of Basal Ganglia Structure and Function*, Volume 20 of *Handbook of Behavioral Neuroscience*, pp. 549 – 559. Elsevier.

Redgrave, P. and K. Gurney (2006). The short-latency dopamine signal: a role in discovering novel actions? *Nat Rev Neurosci 7*(12), 967 – 975.

Redgrave, P., K. Gurney, and J. Reynolds (2008). What is reinforced by phasic dopamine signals? *Brain Research Reviews 58*(2), 322 – 339.

Reitsma, P. (2010). The transfer of correlations from basal ganglia to thalamus in parkinson's disease. Master's thesis, University of Pittsburgh, Pittsburgh, PA.

Reitsma, P., B. Doiron, and J. E. Rubin (2011). Correlation transfer from basal ganglia to thalamus in parkinson's disease. *Frontiers in Computational Neuroscience 5*(0).

Richert, M., J. M. Nageswaran, N. Dutt, and J. L. Krichmar (2011). An efficient simulation environment for modeling large-scale cortical processing. *Frontiers in Neuroinformatics 5*(19).

Richmond, P., B. Lars, G. Michele, and V. E. Richmond (2011, 05). Democratic population decisions result in robust policy-gradient learning: A parametric study with GPU simulations. *PLoS ONE 6*(5).

Rosin, B., M. Slovik, R. Mitelman, M. Rivlin-Etzion, S. Haber, Z. Israel, E. Vaadia, and H. Bergman (2011). Closed-loop deep brain stimulation is superior in ameliorating parkinsonism. *Neuron 72*(2), 370 – 384.

Rubin, J. and D. Terman (2004). High frequency stimulation of the subthalamic nucleus eliminates pathological thalamic rhythmicity in a computational model. *Journal of Computational Neuroscience 16*(3), 211–235.

Rubin, J. E., C. C. McIntyre, R. S. Turner, and T. Wichmann (2012). Basal ganglia activity patterns in parkinsonism and computational modeling of their downstream effects. *European Journal of Neuroscience 36*(2), 2213–2228.

Schemmel, J., D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner (2010). A wafer-scale neuromorphic hardware system for large-scale neural modeling. *Proceedings of the 2010 IEEE International Symposium on Circuits and Systems (ISCAS10)*, 1947–1950.

Schiff, S. J. (2010). Towards model-based control of parkinson's disease. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 368*(1918), 2269–2308.

Schiff, S. J. (2012). *Neural Control Engineering The Emerging Intersection between Control Theory and Neuroscience*. The MIT Press.

Schrum, J. and R. Miikkulainen (2010). Evolving agent behavior in multiobjective domains using fitness-based shaping. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pp. 439–446. ACM.

Schultz, W., P. Dayan, and P. R. Montague (1997). A neural substrate of prediction and reward. *Science 275*(5306), 1593–1599.

Scorcioni, R. (2010, may). Gpgpu implementation of a synaptically optimized, anatomically accurate spiking network simulator. In *Biomedical Sciences and Engineering Conference (BSEC), 2010*, pp. 1 –3.

Scott, A. (2002). *Neuroscience A Mathematical Neuroscience*. New York, NY: Springer-Verlag.

SET Corporation (2012). Castle. `https://project.setcorp.com/castle/index.html`.

Sherman, S. (2001). Tonic and burst firing: dual modes of thalamocortical relay. *Trends in Neurosciences 24*(2), 122 – 126.

Sherman, S. M. and R. W. Guillery (2002). The role of the thalamus in the flow of information to the cortex. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences 357*(1428), 1695–1708.

Shinners, P. (2012). Pygame. `http://pygame.org/`. Accessed: 10/22/2012.

Shouno, O., J. Takeuchi, and H. Tsujino (2009). A spiking neuron model of the basal ganglia circuitry that can generate behavioral variability. In J. Bures, I. Kopin, B. McEwen, K. Pribram, J. Rosenblatt, and L. Weiskranz (Eds.), *The Basal Ganglia IX*, Volume 58 of *Advances in Behavioral Biology*, pp. 191–200. Springer New York.

Song, S., L. Abbott, et al. (2001). Cortical development and remapping through spike timing-dependent plasticity. *Neuron 32*(2), 339–350.

Song, S., K. D. Miller, and L. F. Abbott (2000). Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience* (9), 919–926.

Srinivasa, N. and Y. Cho (2012). Self-organizing spiking neural model for learning fault-tolerant spatio-motor transformations. *Neural Networks and Learning Systems, IEEE Transactions on PP*(99), 1.

Srinivasa, N. and J. Cruz-Albrecht (2012, jan.). Neuromorphic adaptive plastic scalable electronics: Analog learning systems. *Pulse, IEEE 3*(1), 51 –56.

Stewart, T., X. Choo, and C. Eliasmith (2010). Dynamic behaviour of a spiking model of action selection in the basal ganglia. In *10th International Conference on Cognitive Modeling*, pp. 235–240.

Stewart, T. C., T. Bekolay, and C. Eliasmith (2012). Learning to select actions with spiking neurons in the basal ganglia. *Frontiers in Neuroscience 6*(00002).

Sur, S., U. K. R. Bondhugula, A. Mamidala, H.-W. Jin, and D. K. Panda (2005). High performance rdma based all-to-all broadcast for infiniband clusters. In *Proceedings of International Conference On High Performance Computing (HIPC)*.

Sussillo, D., T. Toyoizumi, and W. Maass (June 2007). Self-tuning of neural circuits through short-term synaptic plasticity. *Journal of Neurophysiology 97*(6), 4079–4095.

Tan, C. O. and D. Bullock (2008). A local circuit model of learned striatal and dopamine cell responses under probabilistic schedules of reward. *The Journal of Neuroscience 28*(40), 10062–10074.

Terman, D., J. E. Rubin, A. C. Yew, and C. J. Wilson (2002). Activity patterns in a model for the subthalamopallidal network of the basal ganglia. *The Journal of Neuroscience 22*(7), 2963–2976.

Thibeault, C. M., R. Hoang, and F. C. Harris Jr. (2011, March). A novel multi-gpu neural simulator. In *ISCA's 3rd International Conference on Bioinformatics and Computational Biology (BICoB '11), New Orleans, Louisiana*.

Thibeault, C. M. and N. Srinivasa (2012). Physiologically inspired models of the basal ganglia for embedding in neuromorphic hardware: a modeling study. *In Submission to Journal of Neural Engineering*.

Tiesel, J.-P. and A. S. Maida (2009). Using parallel gpu architecture for simulation of planar i/f networks. *Neural Networks, IEEE - INNS - ENNS International Joint Conference on 0*, 3118–3123.

Touboul, J. (2009). Importance of the cutoff value in the quadratic adaptive integrate-and-fire model. *Neural Computation 21*(8), 2114 – 2122.

Ullah, G. and S. J. Schiff (2009, Apr). Tracking and control of neuronal hodgkin-huxley dynamics. *Phys. Rev. E 79*(4), 040901.

Ullah, G. and S. J. Schiff (2010, 05). Assimilating seizure dynamics. *PLoS Comput Biol 6*(5), e1000776.

van Hoorn, N., J. Togelius, and J. Schmidhuber (2009, sept.). Hierarchical controller learning in a first-person shooter. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pp. 294 –301.

Voss, H. U., J. Timmer, and J. Kurths (2004). Nonlinear dynamical system identification from uncertain and indirect measurements. *I. J. Bifurcation and Chaos*, 1905–1933.

Walters, J. R. and D. A. Bergstrom (2010). Synchronous activity in basal ganglia circuits. In H. Steiner and K. Y. Tseng (Eds.), *Handbook of Basal Ganglia Structure and Function*, Volume 20 of *Handbook of Behavioral Neuroscience*, pp. 429 – 443. Elsevier.

West, M. (2007, January). Evolve your hierarchy. http://cowboyprogramming.com/2007/01/05/evolve-your-heirachy/. Accessed: 8/15/2012.

Wiles, J., D. Ball, S. Heath, C. Nolan, and P. Stratton (2010). Spike-time robotics: A rapid response circuit for a robot that seeks temporally varying stimuli. *Australian Journal of Intelligent Information Processing Systems 11*(1).

Wilson, E. C., P. H. Goodman, and F. C. Harris Jr. (2001). Implementation of a biologically realistic parallel neocortical-neural network simulator. *Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing, March 12-14, 2001*, 1–11.

Winn, P., D. I. Wilson, and P. Redgrave (2010). Subcortical connections of the basal ganglia. In H. Steiner and K. Y. Tseng (Eds.), *Handbook of Basal Ganglia Structure and Function*, Volume 20 of *Handbook of Behavioral Neuroscience*, pp. 397 – 408. Elsevier.

Xilinx (2009). Vertex 5 family overview. http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf.

Yudanov, D., M. Shaaban, R. Melton, and L. Reznik (2010, july). Gpu-based simulation of spiking neural networks with real-time performance amp; high accuracy. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pp. 1 –8.

# Appendix A

# Communication Experiment Results

**Table A.1:** Communication Scheme Experiments: IB, 10Hz Activity.

| **Strong Scaling** | | | **Communication Scheme** | | |
|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **Blocking** | **Non-Blocking** | **Alltoall** |
| 4 | 2000000 | 1000 | 0.16 | 0.18 | 0.27 |
| 8 | 2000000 | 1000 | 0.21 | 0.22 | 0.31 |
| 16 | 2000000 | 1000 | 0.33 | 0.30 | 0.64 |
| 32 | 2000000 | 1000 | 0.52 | 0.47 | 1.27 |
| 48 | 2000000 | 1000 | 1.23 | 1.24 | 2.16 |
| 4 | 250000 | 10000 | 0.04 | 0.04 | 0.06 |
| 8 | 250000 | 10000 | 0.14 | 0.09 | 0.15 |
| 16 | 250000 | 10000 | 0.15 | 0.21 | 0.33 |
| 32 | 250000 | 10000 | 0.32 | 0.32 | 1.07 |
| 48 | 250000 | 10000 | 1.21 | 1.22 | 1.98 |

| **Weak Scaling** | | | **Communication Scheme** | | |
|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **Blocking** | **Non-Blocking** | **Alltoall** |
| 4 | 2000000 | 1000 | 0.16 | 0.18 | 0.19 |
| 8 | 4000000 | 1000 | 0.40 | 0.41 | 0.43 |
| 16 | 8000000 | 1000 | 1.06 | 1.16 | 1.22 |
| 32 | 16000000 | 1000 | 2.71 | 2.61 | 3.25 |
| 48 | 24000000 | 1000 | 5.40 | 5.37 | 5.53 |
| 4 | 250000 | 10000 | 0.04 | 0.03 | 0.05 |
| 8 | 500000 | 10000 | 0.13 | 0.15 | 0.17 |
| 16 | 1000000 | 10000 | 0.22 | 0.22 | 0.60 |
| 32 | 2000000 | 10000 | 0.46 | 0.50 | 1.26 |
| 48 | 3000000 | 10000 | 1.38 | 1.45 | 2.21 |

**Table A.2:** Bit-Packing Experiments: IB, 10Hz Activity.

| Strong Scaling | | | Pivot Point | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **0** | **1** | **2** | **3** | **10** | **20** |
| 4 | 2000000 | 1000 | 0.75 | 0.17 | 0.16 | 0.18 | 0.18 | 0.17 |
| 8 | 2000000 | 1000 | 0.87 | 0.21 | 0.23 | 0.24 | 0.23 | 0.22 |
| 16 | 2000000 | 1000 | 0.90 | 0.34 | 0.38 | 0.33 | 0.34 | 0.33 |
| 32 | 2000000 | 1000 | 1.32 | 0.55 | 0.49 | 0.52 | 0.46 | 0.50 |
| 48 | 2000000 | 1000 | 2.01 | 1.26 | 1.24 | 1.23 | 1.20 | 1.21 |
| 4 | 250000 | 10000 | 0.14 | 0.04 | 0.05 | 0.04 | 0.04 | 0.04 |
| 8 | 250000 | 10000 | 0.18 | 0.09 | 0.09 | 0.08 | 0.10 | 0.10 |
| 16 | 250000 | 10000 | 0.26 | 0.18 | 0.21 | 0.17 | 0.21 | 0.20 |
| 32 | 250000 | 10000 | 0.37 | 0.29 | 0.35 | 0.33 | 0.35 | 0.32 |
| 48 | 250000 | 10000 | 1.20 | 1.19 | 1.20 | 1.24 | 1.18 | 1.18 |

| Weak Scaling | | | Pivot Point | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **0** | **1** | **2** | **3** | **10** | **20** |
| 4 | 2000000 | 1000 | 0.77 | 0.17 | 0.18 | 0.19 | 0.17 | 0.17 |
| 8 | 4000000 | 1000 | 2.04 | 0.43 | 0.41 | 0.43 | 0.46 | 0.44 |
| 16 | 8000000 | 1000 | 4.32 | 1.01 | 1.11 | 1.00 | 1.07 | 1.00 |
| 32 | 16000000 | 1000 | 9.95 | 3.06 | 2.67 | 2.58 | 2.72 | 2.34 |
| 48 | 24000000 | 1000 | 21.56 | 5.28 | 5.21 | 5.45 | 5.36 | 5.37 |
| 4 | 250000 | 10000 | 0.13 | 0.04 | 0.06 | 0.03 | 0.04 | 0.04 |
| 8 | 500000 | 10000 | 0.27 | 0.11 | 0.11 | 0.12 | 0.13 | 0.07 |
| 16 | 1000000 | 10000 | 0.50 | 0.27 | 0.30 | 0.21 | 0.22 | 0.26 |
| 32 | 2000000 | 10000 | 1.03 | 0.50 | 0.54 | 0.48 | 0.51 | 0.53 |
| 48 | 3000000 | 10000 | 2.55 | 1.38 | 1.44 | 1.40 | 1.47 | 1.46 |

**Table A.3:** Communication Scheme Experiments: IB, 30Hz Activity.

| **Strong Scaling** | | | **Communication Scheme** | | |
|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **Blocking** | **Non-Blocking** | **Alltoall** |
| 4 | 2000000 | 1000 | 0.70 | 0.50 | 0.55 |
| 8 | 2000000 | 1000 | 0.61 | 0.62 | 0.60 |
| 16 | 2000000 | 1000 | 0.94 | 0.76 | 0.90 |
| 32 | 2000000 | 1000 | 0.87 | 1.15 | 1.47 |
| 48 | 2000000 | 1000 | 1.88 | 2.06 | 2.35 |
| 4 | 250000 | 10000 | 0.08 | 0.10 | 0.10 |
| 8 | 250000 | 10000 | 0.13 | 0.16 | 0.19 |
| 16 | 250000 | 10000 | 0.26 | 0.26 | 0.48 |
| 32 | 250000 | 10000 | 0.33 | 0.35 | 1.03 |
| 48 | 250000 | 10000 | 1.21 | 1.19 | 2.06 |

| **Weak Scaling** | | | **Communication Scheme** | | |
|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **Blocking** | **Non-Blocking** | **Alltoall** |
| 4 | 2000000 | 1000 | 0.68 | 0.51 | 0.56 |
| 8 | 4000000 | 1000 | 2.08 | 1.31 | 1.26 |
| 16 | 8000000 | 1000 | 5.15 | 3.15 | 3.11 |
| 32 | 16000000 | 1000 | 11.47 | 7.99 | 7.99 |
| 48 | 24000000 | 1000 | 20.92 | 14.57 | 17.77 |
| 4 | 250000 | 10000 | 0.08 | 0.08 | 0.10 |
| 8 | 500000 | 10000 | 0.21 | 0.20 | 0.24 |
| 16 | 1000000 | 10000 | 0.40 | 0.41 | 0.70 |
| 32 | 2000000 | 10000 | 0.81 | 0.87 | 1.43 |
| 48 | 3000000 | 10000 | 2.33 | 2.49 | 2.61 |

**Table A.4:** Bit-Packing Experiments: IB, 30Hz Activity.

| Strong Scaling | | | Pivot Point | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **0** | **1** | **2** | **3** | **10** | **20** |
| 4 | 2000000 | 1000 | 1.44 | 0.52 | 0.51 | 0.51 | 0.52 | 0.50 |
| 8 | 2000000 | 1000 | 1.40 | 0.65 | 0.63 | 0.63 | 0.62 | 0.68 |
| 16 | 2000000 | 1000 | 1.43 | 0.94 | 0.71 | 0.75 | 0.72 | 0.76 |
| 32 | 2000000 | 1000 | 1.62 | 1.21 | 1.07 | 1.08 | 1.01 | 0.85 |
| 48 | 2000000 | 1000 | 3.36 | 2.16 | 2.05 | 2.00 | 2.12 | 1.97 |
| 4 | 250000 | 10000 | 0.18 | 0.10 | 0.08 | 0.08 | 0.10 | 0.11 |
| 8 | 250000 | 10000 | 0.26 | 0.19 | 0.14 | 0.14 | 0.15 | 0.12 |
| 16 | 250000 | 10000 | 0.35 | 0.29 | 0.21 | 0.20 | 0.27 | 0.27 |
| 32 | 250000 | 10000 | 0.45 | 0.40 | 0.33 | 0.33 | 0.30 | 0.35 |
| 48 | 250000 | 10000 | 1.28 | 1.15 | 1.20 | 1.21 | 1.22 | 1.20 |

| Weak Scaling | | | Pivot Point | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **0** | **1** | **2** | **3** | **10** | **20** |
| 4 | 2000000 | 1000 | 1.37 | 0.53 | 0.53 | 0.53 | 0.50 | 0.53 |
| 8 | 4000000 | 1000 | 3.05 | 1.30 | 1.33 | 1.31 | 1.36 | 1.30 |
| 16 | 8000000 | 1000 | 7.13 | 3.19 | 3.44 | 3.21 | 3.17 | 3.28 |
| 32 | 16000000 | 1000 | 15.86 | 7.93 | 7.92 | 7.95 | 8.10 | 8.18 |
| 48 | 24000000 | 1000 | 59.47 | 15.09 | 14.39 | 14.93 | 14.48 | 14.21 |
| 4 | 250000 | 10000 | 0.21 | 0.11 | 0.09 | 0.07 | 0.08 | 0.12 |
| 8 | 500000 | 10000 | 0.41 | 0.25 | 0.20 | 0.21 | 0.21 | 0.18 |
| 16 | 1000000 | 10000 | 0.77 | 0.58 | 0.41 | 0.48 | 0.47 | 0.43 |
| 32 | 2000000 | 10000 | 1.57 | 1.15 | 0.83 | 0.82 | 0.81 | 0.86 |
| 48 | 3000000 | 10000 | 3.34 | 2.75 | 2.74 | 2.68 | 2.76 | 2.53 |

**Table A.5:** Communication Scheme Experiments: IB, 50Hz Activity.

| Strong Scaling | | | Communication Scheme | | |
|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **Blocking** | **Non-Blocking** | **Alltoall** |
| 4 | 2000000 | 1000 | 1.13 | 0.83 | 0.84 |
| 8 | 2000000 | 1000 | 1.69 | 1.05 | 1.05 |
| 16 | 2000000 | 1000 | 1.36 | 1.25 | 1.16 |
| 32 | 2000000 | 1000 | 1.67 | 1.54 | 1.85 |
| 48 | 2000000 | 1000 | 2.75 | 2.71 | 2.87 |
| 4 | 250000 | 10000 | 0.12 | 0.12 | 0.11 |
| 8 | 250000 | 10000 | 0.16 | 0.15 | 0.30 |
| 16 | 250000 | 10000 | 0.25 | 0.27 | 0.55 |
| 32 | 250000 | 10000 | 0.35 | 0.39 | 1.16 |
| 48 | 250000 | 10000 | 1.17 | 1.17 | 2.05 |

| Weak Scaling | | | Communication Scheme | | |
|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **Blocking** | **Non-Blocking** | **Alltoall** |
| 4 | 2000000 | 1000 | 1.07 | 0.82 | 0.81 |
| 8 | 4000000 | 1000 | 3.06 | 2.19 | 2.12 |
| 16 | 8000000 | 1000 | 7.37 | 5.25 | 4.63 |
| 32 | 16000000 | 1000 | 17.14 | 12.66 | 17.22 |
| 48 | 24000000 | 1000 | 32.65 | 22.54 | 30.39 |
| 4 | 250000 | 10000 | 0.10 | 0.13 | 0.11 |
| 8 | 500000 | 10000 | 0.25 | 0.28 | 0.31 |
| 16 | 1000000 | 10000 | 0.56 | 0.57 | 0.79 |
| 32 | 2000000 | 10000 | 1.36 | 1.35 | 1.76 |
| 48 | 3000000 | 10000 | 3.85 | 3.90 | 3.53 |

**Table A.6:** Bit-Packing Experiments: IB, 50Hz Activity.

| Strong Scaling | | | Pivot Point | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **0** | **1** | **2** | **3** | **10** | **20** |
| 4 | 2000000 | 1000 | 2.08 | 2.12 | 0.87 | 0.82 | 0.85 | 0.85 |
| 8 | 2000000 | 1000 | 2.08 | 2.10 | 1.06 | 1.08 | 1.04 | 1.06 |
| 16 | 2000000 | 1000 | 2.17 | 2.13 | 1.26 | 1.36 | 1.29 | 1.47 |
| 32 | 2000000 | 1000 | 2.18 | 2.25 | 1.55 | 1.54 | 1.53 | 1.58 |
| 48 | 2000000 | 1000 | 5.62 | 5.57 | 2.77 | 2.76 | 2.95 | 2.87 |
| 4 | 250000 | 10000 | 0.28 | 0.27 | 0.11 | 0.10 | 0.15 | 0.12 |
| 8 | 250000 | 10000 | 0.34 | 0.33 | 0.18 | 0.14 | 0.17 | 0.15 |
| 16 | 250000 | 10000 | 0.43 | 0.41 | 0.26 | 0.24 | 0.27 | 0.27 |
| 32 | 250000 | 10000 | 0.49 | 0.54 | 0.37 | 0.36 | 0.35 | 0.38 |
| 48 | 250000 | 10000 | 1.30 | 1.30 | 1.18 | 1.21 | 1.20 | 1.17 |

| Weak Scaling | | | Pivot Point | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **0** | **1** | **2** | **3** | **10** | **20** |
| 4 | 2000000 | 1000 | 2.07 | 2.06 | 0.82 | 0.96 | 0.82 | 0.83 |
| 8 | 4000000 | 1000 | 4.47 | 4.50 | 2.22 | 2.19 | 2.18 | 2.31 |
| 16 | 8000000 | 1000 | 10.48 | 9.76 | 6.71 | 5.29 | 5.34 | 5.33 |
| 32 | 16000000 | 1000 | 23.85 | 24.00 | 12.84 | 12.64 | 14.61 | 15.13 |
| 48 | 24000000 | 1000 | 96.84 | 96.90 | 22.84 | 22.52 | 23.15 | 21.48 |
| 4 | 250000 | 10000 | 0.27 | 0.27 | 0.12 | 0.12 | 0.11 | 0.13 |
| 8 | 500000 | 10000 | 0.57 | 0.56 | 0.26 | 0.25 | 0.30 | 0.25 |
| 16 | 1000000 | 10000 | 1.08 | 1.06 | 0.59 | 0.66 | 0.57 | 0.67 |
| 32 | 2000000 | 10000 | 2.22 | 2.16 | 1.72 | 1.26 | 1.31 | 1.37 |
| 48 | 3000000 | 10000 | 4.31 | 4.46 | 3.92 | 3.93 | 3.95 | 3.91 |

**Table A.7:** Communication Scheme Experiments: IB, 80Hz Activity.

| **Strong Scaling** | | | **Communication Scheme** | | |
|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **Blocking** | **Non-Blocking** | **Alltoall** |
| 4 | 2000000 | 1000 | 1.81 | 1.51 | 1.30 |
| 8 | 2000000 | 1000 | 2.54 | 1.77 | 1.69 |
| 16 | 2000000 | 1000 | 3.58 | 2.08 | 2.23 |
| 32 | 2000000 | 1000 | 2.64 | 2.54 | 2.84 |
| 48 | 2000000 | 1000 | 4.05 | 4.05 | 4.21 |
| 4 | 250000 | 10000 | 0.17 | 0.17 | 0.21 |
| 8 | 250000 | 10000 | 0.21 | 0.24 | 0.33 |
| 16 | 250000 | 10000 | 0.31 | 0.31 | 0.55 |
| 32 | 250000 | 10000 | 0.48 | 0.47 | 1.24 |
| 48 | 250000 | 10000 | 1.20 | 1.26 | 2.00 |

| **Weak Scaling** | | | **Communication Scheme** | | |
|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **Blocking** | **Non-Blocking** | **Alltoall** |
| 4 | 2000000 | 1000 | 1.78 | 1.52 | 1.32 |
| 8 | 4000000 | 1000 | 5.74 | 3.76 | 3.15 |
| 16 | 8000000 | 1000 | 15.48 | 8.81 | 10.08 |
| 32 | 16000000 | 1000 | 36.77 | 20.11 | 23.19 |
| 48 | 24000000 | 1000 | 54.08 | 35.16 | 47.50 |
| 4 | 250000 | 10000 | 0.16 | 0.17 | 0.18 |
| 8 | 500000 | 10000 | 0.37 | 0.40 | 0.44 |
| 16 | 1000000 | 10000 | 1.18 | 1.11 | 1.00 |
| 32 | 2000000 | 10000 | 2.70 | 3.05 | 3.03 |
| 48 | 3000000 | 10000 | 5.58 | 5.34 | 5.57 |

**Table A.8:** Bit-Packing Experiments: IB, 80Hz Activity.

| Strong Scaling | | | Pivot Point | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **0** | **1** | **2** | **3** | **10** | **20** |
| 4 | 2000000 | 1000 | 3.33 | 3.30 | 3.36 | 1.43 | 1.45 | 1.45 |
| 8 | 2000000 | 1000 | 3.37 | 3.38 | 3.35 | 1.81 | 1.78 | 1.80 |
| 16 | 2000000 | 1000 | 3.39 | 3.47 | 3.41 | 1.99 | 2.02 | 2.07 |
| 32 | 2000000 | 1000 | 3.63 | 3.40 | 3.49 | 2.62 | 2.78 | 2.70 |
| 48 | 2000000 | 1000 | 10.02 | 10.00 | 9.98 | 3.88 | 3.95 | 3.91 |
| 4 | 250000 | 10000 | 0.37 | 0.37 | 0.37 | 0.17 | 0.17 | 0.18 |
| 8 | 250000 | 10000 | 0.47 | 0.55 | 0.53 | 0.22 | 0.24 | 0.23 |
| 16 | 250000 | 10000 | 0.65 | 0.61 | 0.63 | 0.32 | 0.33 | 0.33 |
| 32 | 250000 | 10000 | 0.62 | 0.63 | 0.70 | 0.46 | 0.48 | 0.46 |
| 48 | 250000 | 10000 | 1.39 | 1.40 | 1.37 | 1.27 | 1.27 | 1.24 |

| Weak Scaling | | | Pivot Point | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **0** | **1** | **2** | **3** | **10** | **20** |
| 4 | 2000000 | 1000 | 3.29 | 3.35 | 3.28 | 1.45 | 1.43 | 1.46 |
| 8 | 4000000 | 1000 | 7.14 | 7.13 | 7.28 | 3.76 | 3.88 | 4.15 |
| 16 | 8000000 | 1000 | 15.31 | 15.36 | 15.40 | 8.77 | 8.85 | 9.00 |
| 32 | 16000000 | 1000 | 40.52 | 40.11 | 40.01 | 20.68 | 19.75 | 20.47 |
| 48 | 24000000 | 1000 | 163.72 | 163.65 | 163.69 | 33.63 | 32.90 | 35.69 |
| 4 | 250000 | 10000 | 0.36 | 0.37 | 0.37 | 0.16 | 0.20 | 0.18 |
| 8 | 500000 | 10000 | 0.84 | 0.84 | 0.84 | 0.39 | 0.40 | 0.39 |
| 16 | 1000000 | 10000 | 1.70 | 1.73 | 1.69 | 1.09 | 1.05 | 1.02 |
| 32 | 2000000 | 10000 | 3.55 | 3.51 | 3.57 | 2.45 | 2.73 | 2.58 |
| 48 | 3000000 | 10000 | 5.96 | 5.73 | 5.59 | 5.48 | 5.65 | 5.48 |

**Table A.9:** Communication Scheme Experiments: Ethernet, 10Hz Activity.

| Strong Scaling | | | Communication Scheme | | |
|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **Blocking** | **Non-Blocking** | **Alltoall** |
| 4 | 2000000 | 1000 | 1.84 | 1.84 | 3.61 |
| 8 | 2000000 | 1000 | 5.32 | 5.43 | 5.09 |
| 16 | 2000000 | 1000 | 5.67 | 5.69 | 9.47 |
| 32 | 2000000 | 1000 | 6.65 | 6.33 | 16.74 |
| 48 | 2000000 | 1000 | 7.34 | 7.64 | 24.66 |
| 4 | 250000 | 10000 | 0.41 | 0.40 | 0.54 |
| 8 | 250000 | 10000 | 3.38 | 3.40 | 3.56 |
| 16 | 250000 | 10000 | 3.47 | 3.45 | 6.86 |
| 32 | 250000 | 10000 | 3.74 | 3.69 | 13.82 |
| 48 | 250000 | 10000 | 4.15 | 4.18 | 23.54 |

| Weak Scaling | | | Communication Scheme | | |
|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **Blocking** | **Non-Blocking** | **Alltoall** |
| 4 | 2000000 | 1000 | 1.83 | 1.95 | 3.18 |
| 8 | 4000000 | 1000 | 8.03 | 8.30 | 6.94 |
| 16 | 8000000 | 1000 | 15.19 | 15.49 | 13.89 |
| 32 | 16000000 | 1000 | 49.33 | 53.29 | 35.13 |
| 48 | 24000000 | 1000 | 92.64 | 79.04 | 235.38 |
| 4 | 250000 | 10000 | 0.38 | 0.34 | 0.60 |
| 8 | 500000 | 10000 | 3.71 | 3.68 | 4.07 |
| 16 | 1000000 | 10000 | 4.36 | 4.38 | 8.20 |
| 32 | 2000000 | 10000 | 6.22 | 6.43 | 19.51 |
| 48 | 3000000 | 10000 | 9.35 | 9.42 | 28.76 |

**Table A.10:** Bit-Packing Experiments: Ethernet, 10Hz Activity.

| Strong Scaling | | | Pivot Point | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **0** | **1** | **2** | **3** | **10** | **20** |
| 4 | 2000000 | 1000 | 5.62 | 2.06 | 1.92 | 1.87 | 1.94 | 1.85 |
| 8 | 2000000 | 1000 | 11.57 | 5.36 | 5.35 | 5.45 | 5.33 | 5.49 |
| 16 | 2000000 | 1000 | 12.43 | 5.71 | 5.70 | 5.69 | 5.83 | 5.71 |
| 32 | 2000000 | 1000 | 16.17 | 6.39 | 6.62 | 6.24 | 6.39 | 6.33 |
| 48 | 2000000 | 1000 | 18.26 | 7.56 | 7.42 | 7.44 | 7.38 | 7.58 |
| 4 | 250000 | 10000 | 0.85 | 0.40 | 0.38 | 0.39 | 0.37 | 0.39 |
| 8 | 250000 | 10000 | 3.92 | 3.38 | 3.37 | 3.38 | 3.29 | 3.40 |
| 16 | 250000 | 10000 | 4.13 | 3.45 | 3.48 | 3.46 | 3.45 | 3.47 |
| 32 | 250000 | 10000 | 4.44 | 3.70 | 3.68 | 3.69 | 3.72 | 3.73 |
| 48 | 250000 | 10000 | 5.06 | 4.10 | 4.23 | 4.08 | 4.12 | 4.12 |

| Weak Scaling | | | Pivot Point | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **0** | **1** | **2** | **3** | **10** | **20** |
| 4 | 2000000 | 1000 | 5.78 | 1.90 | 1.92 | 1.94 | 1.90 | 1.89 |
| 8 | 4000000 | 1000 | 21.92 | 8.15 | 7.89 | 8.12 | 7.84 | 7.89 |
| 16 | 8000000 | 1000 | 55.24 | 15.64 | 17.35 | 17.26 | 15.47 | 19.01 |
| 32 | 16000000 | 1000 | 132.93 | 40.14 | 39.65 | 38.31 | 39.91 | 39.20 |
| 48 | 24000000 | 1000 | 226.11 | 77.33 | 80.05 | 80.55 | 79.42 | 79.37 |
| 4 | 250000 | 10000 | 0.87 | 0.38 | 0.37 | 0.38 | 0.34 | 0.37 |
| 8 | 500000 | 10000 | 4.93 | 3.74 | 3.53 | 3.72 | 3.71 | 3.75 |
| 16 | 1000000 | 10000 | 8.21 | 4.37 | 4.37 | 4.32 | 4.38 | 4.40 |
| 32 | 2000000 | 10000 | 15.50 | 6.34 | 6.60 | 6.47 | 6.37 | 6.35 |
| 48 | 3000000 | 10000 | 27.18 | 9.52 | 9.67 | 9.62 | 9.74 | 9.54 |

**Table A.11:** Communication Scheme Experiments: Ethernet, 30Hz Activity.

| Strong Scaling | | | Communication Scheme | | |
|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **Blocking** | **Non-Blocking** | **Alltoall** |
| 4 | 2000000 | 1000 | 5.38 | 5.46 | 4.20 |
| 8 | 2000000 | 1000 | 10.68 | 11.31 | 8.22 |
| 16 | 2000000 | 1000 | 11.96 | 13.94 | 12.24 |
| 32 | 2000000 | 1000 | 15.30 | 15.52 | 21.20 |
| 48 | 2000000 | 1000 | 18.85 | 19.11 | 34.36 |
| 4 | 250000 | 10000 | 0.82 | 0.81 | 1.10 |
| 8 | 250000 | 10000 | 3.88 | 3.99 | 4.30 |
| 16 | 250000 | 10000 | 4.05 | 4.07 | 7.47 |
| 32 | 250000 | 10000 | 4.37 | 4.43 | 17.00 |
| 48 | 250000 | 10000 | 4.90 | 4.91 | 23.91 |

| Weak Scaling | | | Communication Scheme | | |
|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **Blocking** | **Non-Blocking** | **Alltoall** |
| 4 | 2000000 | 1000 | 5.32 | 5.29 | 4.11 |
| 8 | 4000000 | 1000 | 21.03 | 22.12 | 12.67 |
| 16 | 8000000 | 1000 | 70.41 | 64.25 | 29.45 |
| 32 | 16000000 | 1000 | 194.21 | 146.62 | 232.97 |
| 48 | 24000000 | 1000 | 364.11 | 245.22 | 377.52 |
| 4 | 250000 | 10000 | 0.86 | 0.82 | 1.14 |
| 8 | 500000 | 10000 | 4.80 | 4.82 | 4.92 |
| 16 | 1000000 | 10000 | 7.67 | 7.09 | 9.95 |
| 32 | 2000000 | 10000 | 25.29 | 15.23 | 25.61 |
| 48 | 3000000 | 10000 | 29.76 | 28.80 | 38.97 |

**Table A.12:** Bit-Packing Experiments: Ethernet, 30Hz Activity.

| Strong Scaling | | | Pivot Point | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **0** | **1** | **2** | **3** | **10** | **20** |
| 4 | 2000000 | 1000 | 6.19 | 5.53 | 5.44 | 5.30 | 5.43 | 5.48 |
| 8 | 2000000 | 1000 | 11.63 | 12.34 | 10.61 | 11.97 | 10.52 | 9.90 |
| 16 | 2000000 | 1000 | 13.52 | 13.84 | 14.18 | 13.09 | 12.75 | 12.68 |
| 32 | 2000000 | 1000 | 16.12 | 237.04 | 25.30 | 15.27 | 19.07 | 242.93 |
| 48 | 2000000 | 1000 | 18.33 | 17.66 | 19.60 | 18.84 | 18.68 | 18.78 |
| 4 | 250000 | 10000 | 0.93 | 0.78 | 0.85 | 0.85 | 0.84 | 0.76 |
| 8 | 250000 | 10000 | 4.12 | 3.96 | 3.79 | 3.99 | 3.98 | 3.99 |
| 16 | 250000 | 10000 | 4.20 | 3.98 | 4.04 | 4.04 | 4.07 | 4.06 |
| 32 | 250000 | 10000 | 4.75 | 4.43 | 4.33 | 4.38 | 4.43 | 4.44 |
| 48 | 250000 | 10000 | 5.32 | 5.05 | 4.97 | 4.94 | 4.89 | 5.06 |

| Weak Scaling | | | Pivot Point | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **0** | **1** | **2** | **3** | **10** | **20** |
| 4 | 2000000 | 1000 | 6.41 | 5.49 | 5.39 | 5.36 | 5.34 | 5.30 |
| 8 | 4000000 | 1000 | 22.58 | 21.04 | 23.52 | 21.48 | 23.36 | 25.03 |
| 16 | 8000000 | 1000 | 54.85 | 62.73 | 57.29 | 66.67 | 98.17 | 72.82 |
| 32 | 16000000 | 1000 | 133.43 | 165.18 | 152.16 | 158.69 | 236.06 | 191.45 |
| 48 | 24000000 | 1000 | 286.30 | 252.61 | 253.78 | 241.15 | 262.10 | 256.16 |
| 4 | 250000 | 10000 | 0.95 | 0.82 | 0.80 | 0.80 | 0.85 | 0.83 |
| 8 | 500000 | 10000 | 5.37 | 4.73 | 4.81 | 4.79 | 4.83 | 4.84 |
| 16 | 1000000 | 10000 | 8.34 | 9.68 | 9.44 | 7.04 | 6.97 | 7.14 |
| 32 | 2000000 | 10000 | 26.97 | 159.68 | 26.40 | 234.89 | 16.91 | 22.33 |
| 48 | 3000000 | 10000 | 28.29 | 25.79 | 29.52 | 28.47 | 30.03 | 29.87 |

**Table A.13:** Communication Scheme Experiments: Ethernet, 50Hz Activity.

| Strong Scaling | | | Communication Scheme | | |
|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **Blocking** | **Non-Blocking** | **Alltoall** |
| 4 | 2000000 | 1000 | 8.64 | 9.11 | 6.76 |
| 8 | 2000000 | 1000 | 17.18 | 17.14 | 10.91 |
| 16 | 2000000 | 1000 | 24.86 | 23.41 | 15.51 |
| 32 | 2000000 | 1000 | 29.22 | 23.86 | 26.61 |
| 48 | 2000000 | 1000 | 27.95 | 29.11 | 39.58 |
| 4 | 250000 | 10000 | 1.18 | 1.27 | 1.41 |
| 8 | 250000 | 10000 | 4.52 | 4.53 | 5.17 |
| 16 | 250000 | 10000 | 4.73 | 4.74 | 8.00 |
| 32 | 250000 | 10000 | 4.92 | 5.25 | 14.99 |
| 48 | 250000 | 10000 | 5.83 | 5.95 | 24.61 |

| Weak Scaling | | | Communication Scheme | | |
|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **Blocking** | **Non-Blocking** | **Alltoall** |
| 4 | 2000000 | 1000 | 9.18 | 9.06 | 6.25 |
| 8 | 4000000 | 1000 | 39.77 | 31.73 | 18.84 |
| 16 | 8000000 | 1000 | 109.70 | 95.48 | 213.88 |
| 32 | 16000000 | 1000 | 272.96 | 293.19 | 425.38 |
| 48 | 24000000 | 1000 | 419.45 | 376.31 | 630.91 |
| 4 | 250000 | 10000 | 1.21 | 1.22 | 1.19 |
| 8 | 500000 | 10000 | 5.99 | 5.97 | 5.76 |
| 16 | 1000000 | 10000 | 11.54 | 10.73 | 11.52 |
| 32 | 2000000 | 10000 | 34.41 | 23.52 | 28.32 |
| 48 | 3000000 | 10000 | 45.67 | 47.20 | 48.82 |

**Table A.14:** Bit-Packing Experiments: Ethernet, 50Hz Activity.

| Strong Scaling | | | Pivot Point | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **0** | **1** | **2** | **3** | **10** | **20** |
| 4 | 2000000 | 1000 | 7.11 | 6.78 | 8.94 | 9.17 | 8.94 | 8.96 |
| 8 | 2000000 | 1000 | 11.91 | 12.34 | 18.24 | 16.80 | 17.04 | 16.15 |
| 16 | 2000000 | 1000 | 14.16 | 14.30 | 24.19 | 22.23 | 23.80 | 20.15 |
| 32 | 2000000 | 1000 | 16.22 | 16.26 | 26.62 | 25.21 | 28.08 | 25.18 |
| 48 | 2000000 | 1000 | 18.15 | 17.32 | 30.53 | 30.20 | 32.29 | 30.91 |
| 4 | 250000 | 10000 | 1.00 | 0.99 | 1.21 | 1.22 | 1.24 | 1.21 |
| 8 | 250000 | 10000 | 4.22 | 1.59 | 4.50 | 4.55 | 4.49 | 1.55 |
| 16 | 250000 | 10000 | 4.30 | 4.29 | 5.59 | 4.75 | 4.65 | 4.70 |
| 32 | 250000 | 10000 | 4.54 | 4.45 | 5.26 | 5.09 | 5.19 | 5.08 |
| 48 | 250000 | 10000 | 4.97 | 5.01 | 5.81 | 5.78 | 5.90 | 5.91 |

| Weak Scaling | | | Pivot Point | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **0** | **1** | **2** | **3** | **10** | **20** |
| 4 | 2000000 | 1000 | 6.80 | 6.78 | 8.59 | 8.82 | 8.98 | 9.41 |
| 8 | 4000000 | 1000 | 22.48 | 23.16 | 35.23 | 35.71 | 38.12 | 33.55 |
| 16 | 8000000 | 1000 | 56.46 | 55.87 | 110.51 | 113.05 | 110.06 | 116.20 |
| 32 | 16000000 | 1000 | 132.68 | 133.80 | 260.28 | 251.28 | 257.27 | 253.73 |
| 48 | 24000000 | 1000 | 201.72 | 197.08 | 392.48 | 395.30 | 382.98 | 360.71 |
| 4 | 250000 | 10000 | 1.00 | 0.99 | 1.22 | 1.21 | 1.21 | 1.23 |
| 8 | 500000 | 10000 | 5.27 | 5.05 | 5.91 | 5.97 | 5.12 | 5.97 |
| 16 | 1000000 | 10000 | 9.55 | 8.02 | 11.48 | 9.94 | 12.41 | 10.33 |
| 32 | 2000000 | 10000 | 16.67 | 16.59 | 24.64 | 24.30 | 24.31 | 25.39 |
| 48 | 3000000 | 10000 | 28.01 | 28.70 | 47.11 | 48.44 | 48.63 | 47.85 |

**Table A.15:** Communication Scheme Experiments: Ethernet, 80Hz Activity.

| **Strong Scaling** | | | **Communication Scheme** | | |
|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **Blocking** | **Non-Blocking** | **Alltoall** |
| 4 | 2000000 | 1000 | 16.20 | 14.01 | 9.78 |
| 8 | 2000000 | 1000 | 29.16 | 24.48 | 16.63 |
| 16 | 2000000 | 1000 | 39.14 | 36.98 | 21.38 |
| 32 | 2000000 | 1000 | 46.83 | 43.75 | 44.42 |
| 48 | 2000000 | 1000 | 62.37 | 51.28 | 52.45 |
| 4 | 250000 | 10000 | 2.05 | 1.91 | 2.06 |
| 8 | 250000 | 10000 | 5.01 | 5.47 | 5.01 |
| 16 | 250000 | 10000 | 5.75 | 6.75 | 9.64 |
| 32 | 250000 | 10000 | 6.31 | 6.52 | 16.86 |
| 48 | 250000 | 10000 | 7.19 | 7.35 | 23.19 |

| **Weak Scaling** | | | **Communication Scheme** | | |
|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **Blocking** | **Non-Blocking** | **Alltoall** |
| 4 | 2000000 | 1000 | 14.94 | 15.69 | 10.22 |
| 8 | 4000000 | 1000 | 56.78 | 60.68 | 78.58 |
| 16 | 8000000 | 1000 | 197.00 | 216.01 | 570.75 |
| 32 | 16000000 | 1000 | 555.59 | 481.38 | 942.53 |
| 48 | 24000000 | 1000 | 956.47 | 673.16 | 1445.66 |
| 4 | 250000 | 10000 | 1.93 | 1.83 | 1.74 |
| 8 | 500000 | 10000 | 9.32 | 10.96 | 7.09 |
| 16 | 1000000 | 10000 | 22.39 | 21.04 | 167.58 |
| 32 | 2000000 | 10000 | 48.50 | 40.44 | 201.09 |
| 48 | 3000000 | 10000 | 86.21 | 73.22 | 238.44 |

**Table A.16:** Bit-Packing Experiments: Ethernet, 80Hz Activity.

| Strong Scaling | | | Pivot Point | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **0** | **1** | **2** | **3** | **10** | **20** |
| 4 | 2000000 | 1000 | 8.03 | 7.76 | 7.97 | 15.38 | 15.63 | 14.33 |
| 8 | 2000000 | 1000 | 13.69 | 13.64 | 12.60 | 27.06 | 29.19 | 26.91 |
| 16 | 2000000 | 1000 | 16.85 | 16.89 | 14.40 | 42.47 | 52.34 | 41.32 |
| 32 | 2000000 | 1000 | 16.76 | 16.27 | 16.81 | 45.56 | 46.33 | 65.91 |
| 48 | 2000000 | 1000 | 15.34 | 18.44 | 18.04 | 51.73 | 52.85 | 51.69 |
| 4 | 250000 | 10000 | 1.07 | 1.13 | 1.13 | 1.94 | 1.86 | 1.89 |
| 8 | 250000 | 10000 | 4.32 | 4.31 | 4.30 | 5.53 | 5.65 | 5.36 |
| 16 | 250000 | 10000 | 4.77 | 4.36 | 4.36 | 5.70 | 7.27 | 5.74 |
| 32 | 250000 | 10000 | 4.84 | 7.04 | 4.70 | 6.37 | 6.32 | 6.34 |
| 48 | 250000 | 10000 | 5.22 | 5.22 | 5.20 | 7.34 | 7.45 | 7.40 |

| Weak Scaling | | | Pivot Point | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Nodes** | **Cells** | **Connections** | **0** | **1** | **2** | **3** | **10** | **20** |
| 4 | 2000000 | 1000 | 7.51 | 7.71 | 7.83 | 15.08 | 15.47 | 15.78 |
| 8 | 4000000 | 1000 | 23.57 | 24.41 | 24.25 | 54.64 | 58.78 | 66.61 |
| 16 | 8000000 | 1000 | 55.00 | 62.10 | 56.10 | 180.90 | 210.29 | 205.21 |
| 32 | 16000000 | 1000 | 134.76 | 135.33 | 135.90 | 412.85 | 514.29 | 351.14 |
| 48 | 24000000 | 1000 | 209.46 | 209.46 | 207.97 | 759.45 | 765.08 | 685.88 |
| 4 | 250000 | 10000 | 1.15 | 1.08 | 1.18 | 2.02 | 1.97 | 1.87 |
| 8 | 500000 | 10000 | 5.49 | 5.69 | 5.43 | 9.72 | 8.79 | 9.54 |
| 16 | 1000000 | 10000 | 10.46 | 8.43 | 14.93 | 21.76 | 57.27 | 20.55 |
| 32 | 2000000 | 10000 | 16.52 | 16.65 | 16.18 | 48.21 | 48.43 | 47.04 |
| 48 | 3000000 | 10000 | 27.78 | 28.16 | 27.65 | 77.37 | 76.48 | 74.80 |

# Appendix B

# BrainGames Use Case Diagrams

**Use Case 1** Create System

| | |
|---|---|
| **Description** | Create a system object that implements the BaseSystem interface. The unique bit id for this system will be registered. |
| **Assumptions** | The world object has been instantiated successfully. |
| **Preconditions** | This System has not been registered before. |
| **Postconditions** | This System will be registered and available. |
| **Actors** | • World<br>• BaseSystem |
| **Steps** | 1. The environment will instantiate the new System with default parameters.<br>2. The environment registers the new System with the world.<br>3. The world assigns this System a Bit Id.<br>4. The system is added to the System Container. |
| **Extensions** | 2. A system of this type has already been added.<br>2.1 The program raises an exception. |

**Use Case 2** Initialize System

**Description** Loop through the Systems and initialize their data members. If needed, the components the system needs will be added.

**Assumptions** The System objects have been created and registered with the World.

**Preconditions** All systems have been created and registered.

**Postconditions** The System's components will be registered and its mappers will be created.

**Actors**
- World
- BaseSystem
- ComponentTypeManager

**Steps**

1. The world asks the System to initialize itself giving access to the componentTypeManager.
   - 1.a The System gets the IDs for the component classes it needs.
   - 1.b The System compiles its ComponentType bit string based on the IDs.
   - 1.c The System instantiates its vector mappers.

**Extensions**

1.a A component of this type has already been added.

1.a.1 The program raises an exception.

**Use Case 3** Add System Type

**Description** Add a new SystemType to the SystemTypeManager
**Assumptions** RTTI can get the type information.
**Preconditions** This System has not been registered before.
**Postconditions** This System will be registered with the manager.
**Actors**
- SystemTypeManager

**Steps**
1. The SystemTypeManager searches for the typeIdStr of the System in the data structure.
2. A new SystemType object is instantiated.
3. A SystemTypePtr is added to the data structure.

**Extensions**
1. The SystemType exists in the structure.
   1.1 The program raises an exception.
3. Insertion of the SystemTypePtr into the structure fails.
   3.1 The program raises an exception.

**Use Case 4** Add Component Type

**Description** Add a new ComponentType to the ComponentTypeManager
**Assumptions** RTTI can get the type information.
**Preconditions** This Component has not been registered before.
**Postconditions** This Component will be registered with the manager.
**Actors**
- ComponentTypeManager

**Steps**
1. The ComponentTypeManager searches typeIdStr of the Component in the Data Structure.
2. A new ComponentType object is instantiated.
3. A ComponentTypePtr is added to the structure.

**Extensions**
1. The ComponentType exists in the structure.
   1.1 The program raises an exception.
3. Insertion of the ComponentTypePtr into the structure fails.
   3.1 The program raises an exception.

**Use Case 5** Create Entity

**Description** Create an empty entity and return a pointer of it to the caller.
**Assumptions** The entity manager has created a list of available Entities.
**Preconditions** The World has been created and initialized
**Postconditions** A new Entity will be created and placed active.
**Actors**
- World
- EntityManager
- Environment

**Steps**
1. The Environment requests a new Entity object pointer from the World.
2. The request is passed to the EntityManager that will create the Entity.
   - 2.a The next available Entity is removed from the available list.
   - 2.b The Entity is added to the active list and returned to the caller.

**Use Case 6** Delete Entity

**Description** Remove an entity from the active world.
**Assumptions** The
**Preconditions** Stuff that must be set before
**Postconditions** Stuff that will be changed
**Actors**
- Environment
- World
- EntityManager

**Steps**
1. The World asks the Environment to remove the provided Entity
2. The EnvironmentManager Sets the typeBits to 0.
3. The Entity is then updated/refreshed on all of the systems.
4. The associated Components for this Entity are removed from the ComponentMap.
5. The Entity is moved from the active list to the available list.

**Issues** The container objects are still unclear.

**Use Case 7** Refresh Entity

**Description** Something within the entity has changed so the systems need to be made aware of that.

**Assumptions** The entity has changed type or system bits

**Postconditions** The systems that are interested in this entity have been updated.

**Actors**
- World
- EntityManager
- BaseSystem

**Steps**
1. Loop through each of the Systems notifying them that a change was made.
    - 1.a Compare the Entities System and Type bits for compatibility.
    - 1.b Add or Remove this entity as needed.

---

**Use Case 8** Add Component

**Description** Add a component to and Entity

**Postconditions** The component will be associated with this Entity.

**Actors**
- Environment
- Entity
- EntityManager

**Steps**
1. Get the ComponentType for the Component to be added.
2. Add the Component to the ComponentMap and associate it with the Entity.

**Extensions**

1.a The Component has not been register with the ComponentTypeManger
    - 1.a Follow Use Case B.4

2.a The Component does not have an entry in the ComponentMap
    - 1.a Add a new entry to the ComponentMap using the ComponentTypeId as the key.

# Appendix C

# Hardware efficiency analysis

For comparison with the neuromorphic hardware (NH) three commercial off-the-shelf (COTS) components were selected, standard central processing unit (CPU), field programmable gate arrays (FPGA) and graphical processing unit (GPU). The hardware analysis was constructed to give the COTS components an advantage over the NH estimates. The NH values were selected based on $90nM$ CMOS processes while the COTS components all used $40nM$ or smaller. The NH estimates were made based on SyNAPSE hardware and published VLSI level models of the simple hybrid neuron.

The estimates begin with the floating point operations (FLOPs) per second for the neurons in a particular model, $F_{neuron}$, is defined as

$$F_{neuron} = M f_s g_n. \tag{C.1}$$

Where $M$ is the total number of neurons, $f_s$, is the integrations steps per second and $g_n$ are the floating point operations per neuron. The synaptic FLOPs per second, $F_{synaptic}$, is calculated using

$$F_{synaptic} = N f_s g_{synapse}. \tag{C.2}$$

Where $N$ is the Total number of synapses and $g_{synapse}$ is the number of FLOPS per second for the synapses. The total FLOPs per second for a model is the sum of neuron and synaptic FLOPs per second.

Using the power, in Watts, required for a model can be estimated by

$$P_{model} = \frac{F_{model}}{\beta}, \tag{C.3}$$

and the estimated energy per spike is

$$E_{model} = \frac{P_{model}}{f_{model} N}. \tag{C.4}$$

Where $\beta$ is the FLOPS per watt for the particular hardware and $f_{model}$ is the average firing rate of the model. Table C.1 presents the $\beta$ values used for each of the COTS components.

**Table C.1:** FLOPs per watt, $\beta$ for the COTS calculations.

| **Component** | $\boldsymbol{\beta}$ | **Source** |
|---|---|---|
| $CPU$ | $0.18 \cdot 10^9$ | Duan et al. (2011) |
| $FPGA$ | $1.90 \cdot 10^9$ | Duan et al. (2011)<br>Xilinx (2009) |
| $GPU$ | $15.85 \cdot 10^9$ | NVIDIA (2012) |

The theoretical neuromorphic hardware comparisons are based on the SyNAPSE hard-

ware of Srinivasa and Cruz-Albrecht (2012). The power requirement estimates were calculated using

$$P_{model} = 1.1\left[MP_{neuron} + N\left(P_{synapse} + P_{refresh}\right)\right]. \tag{C.5}$$

Where $P_{neuron}$ and $P_{synapse}$ are the power estimates for neurons and synapses respectively. $P_{refresh}$ is the power for memory refreshes required in the hardware. Note that the power estimate is scaled by $1.1$ to include costs associated with routing and switching. The energy per spike was calculated using

$$E_{model} = \frac{P_{model}}{f_s N}. \tag{C.6}$$

**Table C.2:** Power estimates for neuromorphic hardware.

| **Variable** | **Power** | **Source** |
|---|---|---|
| $P_{neuron}$ | 20 nW | Rangan et al. (2010)<br>Demirkol and Ozoguz (2011) |
| $P_{synapse}$ | 2.4 nW | Cruz-Albrecht et al. (2012)<br>Personal communication |
| $P_{refresh}$ | 18 pW | |

**Table C.3:** Parameters used for each of the models.

| Variable | Description | Action Selection Model Section 9.2.4 | Parkinsonian Model Section 9.2.5 |
|---|---|---|---|
| $M$ | Total number of neurons | $1,152$ | $50$ |
| $N$ | Total number of synapses | $25,124$ | $160$ |
| $f_{model}$ | Average firing rate of the model | $20$ | $80$ |
| $f_s$ | Integration steps per second | $1,000$ | $1,000$ |
| $g_n$ | FLOPS per neuron | $13$ | $13$ |
| $g_{synapses}$ | FLOPS per neuron | $1$ | $1$ |