ANALYSIS FRAMEWORK FOR ADAPTIVE SPIKING NEURAL NETWORKS

BY

FELIX Y. WANG

THESIS

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Electrical and Computer Engineering in the Graduate College of the University of Illinois at Urbana-Champaign, 2014

Urbana, Illinois

Adviser:

Professor Stephen E. Levinson
ABSTRACT

Learning is an inherently closed-loop process that involves the interaction between an intelligent agent and its environment. In the human brain, we assert that the basis for learning is in its ability to represent external stimuli symbolically in an associative memory. Historically, statistical methods such as the hidden Markov model have been used in order to provide the internal symbolic representation to external signals from the environment. This work approaches similar themes by investigating the function of the neocortex, with the ultimate goal of understanding how mental states might arise from spiking activity. Cortical modeling has traditionally focused on the mechanisms and behaviors at the cellular level. However, developments with respect to group or population level phenomena indicate that a shift in focus is necessary to understand how learning and representation of stimuli might occur in the brain. We present a Simulation Tool for Asynchronous Cortical Streams (STACS) for studying spiking neural networks exhibiting adaptation in a closed-loop system.
ACKNOWLEDGMENTS

The author would like to thank his graduate advisor, Professor Stephen E. Levinson, for his continual encouragement, guidance, and patience; Alex M. Duda for insightful discussions on approaching machine intelligence from a biological perspective; Luke A. Wendt for his expertise in control theory and optimization; the Language Acquisition and Robotics Group for their support; and the NeuroEngineering IGERT Program for their generosity. The author would also like to thank his parents, Donghai Wang and Xiuzhi Sun for their love and care.
# TABLE OF CONTENTS

**CHAPTER 1** INTRODUCTION .............................................. 1
  1.1 Foundations for Learning ........................................ 2
  1.2 A Biological Perspective ...................................... 3
  1.3 Neural Simulation .............................................. 4

**CHAPTER 2** LITERATURE REVIEW .................................... 6
  2.1 Neural Substrate .................................................. 6
  2.2 Learning and Memory .......................................... 13
  2.3 Spiking Neural Networks ...................................... 17
  2.4 Simulation Tools ................................................. 22

**CHAPTER 3** FRAMEWORK FOR STUDY ................................. 26
  3.1 Design Philosophy ............................................... 27
  3.2 Problem Representation ....................................... 35
  3.3 Simulation Engine ............................................. 40
  3.4 Neural Network Simulation ................................... 43

**CHAPTER 4** CONCLUSION .............................................. 50
  4.1 Discussion ....................................................... 50
  4.2 Future Work .................................................... 51

REFERENCES ............................................................. 52
CHAPTER 1

INTRODUCTION

The underlying drive of this work is to provide a path toward a more complete understanding of the adaptive learning process that is observed in large networks of spiking neurons. By understanding how mental states might arise from spiking activity in the brain, we hope to gain insight into how human language is acquired, a major focus of the Language Acquisition and Robotics Group. That is, we hope to advance the development of intelligent robots with the ability to learn and use natural language in a manner similar to humans. Intrinsic to a biological perspective on machine intelligence are several of the philosophies embraced by our lab: natural language learning is a semantic and associative process; embodied cognition, that is, interaction with the environment is essential; and learning is performed from a tabula rasa, i.e. without pre-programmed knowledge [1]. In studying the biological mechanism, we may also inform more classical models of machine intelligence in the development of algorithms which may not necessarily require a neural basis.

A key part to the learning problem is representation along with the self-organizing principles that lead to that representation. Methods found in reinforcement learning operate through exploration and exploitation of the environment using iterative approaches such as temporal difference. Here, the internal representation is in the form of a value function or state-action pairs which guide the behavior of the agent in the environment. The hidden Markov model treats the signals measured from the environment as an emission of an underlying state. These states and the transition probabilities between them are learned through the application of dynamic programming. In the biological system, we observe the strengthening and weakening of a synapse in response to spiking activity at the lowest level. But although the synapse may be the seat of memory, the specifics of its representation are still unclear.

Experimentally, neuroscience has experienced tremendous growth in the available technologies analyzing brain function, from multi-electrode arrays
targeted at the cellular level, to a wide range of neuroimaging methods at the whole brain level. A considerable amount of modeling research in computational neuroscience has been performed with respect to the electrochemical and morphological properties of the different neuron types and their connections [2, 3]. At the other end of the biological spectra, we observe many higher order behaviors such as language, fine motor control, and models of agency [4, 5]. Although there exists a gap between these two levels of abstraction, we may fill it in using knowledge procured from either side. Neural simulation and mathematical theory provide a promising tool for studying this gap in our understanding [6]. The approach taken by this work comes from the bottom up, grounded in well-established properties of the brain, while guided by principles from a top-down perspective.

More recently, studies focused at the population level provide some promising insights as to the phenomena that may emerge from the neural substrate in terms of potential representation of state [7, 8]. It is this level of abstraction that we wish to examine. However, with increasing scale and complexity, we cannot simply extrapolate and apply our understanding in a reductionist manner [9]. That is, analysis of neural networks requires methods targeted at neural networks, beyond what we may currently possess with respect to a small number of neurons. Moreover, a framework for study must address the closed-loop system within which the learning process occurs [10]. We hope to provide a computational tool that may accurately and efficiently simulate a spiking neural network as well as analyze the adaptive behavior exhibited.

1.1 Foundations for Learning

Historically, theories of mind and language have largely sidestepped neuroscience in terms biophysical plausibility with respect to the construction of mathematical models. Rather, the systems constructed were tied more closely with the contemporary technology, the popular metaphor being that of the computer [1]. Although these theories experienced significant success, by representing thought as a purely symbolic process, they were limited in terms of what they could achieve. Perhaps the most crucial aspect that is often overlooked by traditional methods to learning natural language is the motor-sensory feedback loop that enables the grounding of semantic meaning. With respect to the biological system, neurons are never found in isolation to stimuli and motor control, even in the most basic organisms.
such as the C. elegans nematode worm [11]. Collectively, these concepts fall under the field of cybernetics, which aims to study communication and control as it exists in machines and living organisms [4]. With respect to the development of intelligent machines, Turing offers these words [12]: “It can be maintained that it is best to provide the machine with the best sense organs that money can buy and then teach it to understand and speak English. This process could follow the normal teaching of a child. Things would be pointed out and named, etc.”

To enable a machine to learn as a human child does, however, requires an understanding of the self-organizing principles that underlie the learning process in humans [13]. Because self-organization allows for the extraction of information from the environment as opposed to having that knowledge a priori, we posit that these principles are central to the formation and manipulation of memory. Furthermore, this memory is associative, referring to the ability to bind related components of an aggregate state such that the activation of a component facilitates the activation of other, associated components. This then leads to the activation of the aggregate. At the lowest level, these components may correspond to collections of neurons involved in the identification of some external (or internal) stimulus, or at a higher level, a system that is capable of recalling related concepts such as the auditory signature of a word with its grounded meaning and actionable behavior.

1.2 A Biological Perspective

The influence from biology has traditionally been of a top-down nature, where we have observed the gross behavior of the intelligent agent and have effectively guessed at models with similar features. With many of the recent developments in experimental and computation neuroscience, we are finally able to reasonably approach the problem of modeling learning from the bottom up [6]. That is, advancements in capabilities of computing machinery and our understanding of the neural substrate now enable us to develop systems of learning that are biophysically plausible at their basis. A major advantage to this is that we can be confident that the results retrieved from simulation correspond to the actual phenomena.

Due to the inherent complexity that they present, however, even with a successfully simulated system, analysis and extraction of the principle functions may not be transparent [9]. The difficulty lies in modeling only the
relevant, information processing characteristics while traversing the levels of abstraction. At a low level, many of these are known. Examples include the precise timing of axonal delays, the neurocomputational properties of the spiking neuron, and the changing of the synaptic weights in response to spiking activity [14]. With respect to the spiking neural network as a collective, we have observed the rate encoding of neurons in the representation of a low dimensional vector [7]. Additionally, we have observed the formation of spatial-temporally locked firing of neural groups in networks exhibiting persistent activity [8]. How these different phenomena interact with each other and the environment is still an open question, one which we hope to address through the development of the proper tools.

1.3 Neural Simulation

Simulation provides a valuable tool for the study of neural systems. It allows for the testing of toy models of behavior at scale, while remaining biophysically plausible. In this way, extending experiments to simulation allows for more in depth analysis than is possible with the constraints of a live culture. Furthermore, without relying on a biological component, the versatility and range of experimentation are vastly increased. As invaluable of a tool as simulation is in studying neural networks, however, this presumes that it is able to accurately reproduce the biophysical phenomena.

Currently, several simulation tools as well as a handful of domain specific languages seek to facilitate computational neuroscience research [15]. Perhaps the most well known of these is NEURON, a simulation environment used to model empirically tested neuron types and networks [16]. While the tool provides graphical interfaces that are rather useful for instruction at small scale, because less emphasis is placed at the population level, the construction of complex networks consisting of many neurons quickly becomes intractable. This limits its ability to adequately describe networks, both from a constructional standpoint and in terms of analysis. To some extent, these limitations are addressed by the introduction of domain specific languages, such as NeuroML, that map onto a number of simulation tools [17]. In addition to supporting the description of the spiking neural network at multiple scales, these provide a common ground for researchers to exchange and test neural models. More recently, simulation tools such as NEST have gained popularity due to their ability to efficiently simulate large networks at the expense of biological precision [18].
With respect to learning in a large spiking neural network, however, there are some shortcomings in the existing set of simulation tools. Although each simulator performs well for each of their specific focuses, by the same token, none are capable of adequately simulating large networks interacting in a realistic setting for embodiment while maintaining a biologically faithful neural substrate. A few key features that are required of this task include the ability to store state data across simulation runs for continuous simulation, models of the spiking neuron and synapse which require interoperability in their dynamics, hardware support for interaction with the world in a closed-loop manner, and the ability to implement modules defined over neurons as a collective for population level analysis. In the development of our Simulation Tool for Asynchronous Cortical Streams (STACS), in addition to taking the above items into consideration, we must also pay attention to design strategies required for efficient parallelization. In this way, we may provide a tool that is not only suitable as a framework for study, but also practical in use.
The gap in our understanding lies between two, rather distinct levels of abstraction. At the base, we have rather well-established models of the neural substrate that describe the local behavior of the learning system at a cellular level. At the top, we observe the result of the fine-tuned system in the form of language acquisition, decision making, and complex motor trajectories. The models at this level are not necessarily tied to the biology, but rather attempt to implement the behavior in a functionally equivalent way. In the cybernetic paradigm, these models lie at the intersection of control and information theory [4]. Although still largely an open problem, some models targeted at explaining the collective behavior in networks of spiking neurons have met with reasonable success in terms of experimental verification [7, 8]. An analysis framework that aims to bridge the gap in our understanding should encompass all of these levels. In the following sections, we cover the relevant literature with respect to these different levels in turn.

2.1 Neural Substrate

At the basis of our study is the neural substrate, at the level of neurons and synapses. The rationale for starting at this level is twofold. First, below the bulk cellular level, many of the dynamics that compose the behavior of the cell are regulatory in nature and unnecessary in fully characterizing the storage and flow of information by the system [19]. Instead of focusing on the processes that enable the neuron as a metabolically constrained system to maintain functionality, we observe only the processes that are important in representing the time-evolution of the information bearing signal. With respect to the neuron, this is the phenomena of the depolarizing action potential of the cell membrane in response to applied current. This qualitatively distinctive behavior from the resting state of the neuron is in the form of a spike. With respect to synapses between neurons, there are two important phenomena. There is propagation of the spike from
the pre-synaptic neuron to the post-synaptic neuron based on the strength of the connection, and there is the change in synaptic strength [14]. Moreover, the models of the neural substrate at this level are rather well studied and well-established in terms of their ability to capture these biophysical phenomena.

Second, the dynamics of the level above, that of networks of neurons, are less established and lack the same degree of biological faithfulness as that of the cellular level. In particular, models at this level impose simplifications a priori to a full understanding of how the underlying dynamics project to the network, producing behavior that is inconsistent with respect to what has been experimentally verified [8, 20]. These inconsistencies include treating a population of neurons effectively as a single neuron, poor scaling to patterns of stimuli, and the dismissal of fundamental properties such as delay times. If we are to construct an accurate model of how information is represented in the brain, we do not want to handicap our search by oversimplifying beforehand. Rather, we chose to begin at the cellular level in the neural substrate in order to ensure that the representation that falls out of the analysis is biophysically implementable, or at worst prohibited only by mechanisms that are yet unknown to us.

2.1.1 Spiking Neuron

The most distinctive feature of a spiking neural network is the spiking activity of the neuron. The neuron is a specialized cell composed of three major components [21]. At one end, stimuli from the environment or other neurons in the form of neurotransmitters cause current to accumulate from the receiving dendrites in a fan-in fashion at the cell body. At the other end, the neuron releases neurotransmitter at the axon terminals to adjacent neurons in a fan-out fashion. Between these two lies the axon, which propagates a qualitatively all-or-nothing response to stimuli in the form of a spike in the membrane voltage. The dynamics of the membrane voltage with respect to the applied current can be described by the diffusion of mediating ions across the cell membrane through chemically, via neurotransmitter, and electrically, via membrane voltage, gated channels.

Conductance-Based Models

This biophysical mechanism is captured in conductance-based models of the spiking neuron. The most popular of these is the Hodgkin-Huxley type
neuron model that was first empirically modeled from experiments involving the axon of the giant squid [22]. The classic HH-model equations, containing three major ion currents, $Na^+$, $K^+$, and leakage, and an applied current, $I$, are given in eq. 2.1.

$$
C \frac{dv}{dt} = -g_{Na}m^3h(v - E_{Na}) - g_{K}n^4(v - E_{K}) - g_{L}(v - E_{L}) + I
$$

$$
\frac{dn}{dt} = \alpha_n(1 - n) - \beta_n n
$$

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

$$
\frac{dh}{dt} = \alpha_h(1 - h) - \beta_h h
$$

The states of the model are $v$, the membrane voltage, and $n, m, h$, the gating variables indicating how closed or open an ion gate is. The other parameters, $C, g_{\star}, E_{\star}, \alpha_{\star}, \beta_{\star}$, describe the capacitance of the cell membrane, the maximal conductance of the ion channels, the reversal potentials of the ion currents, and voltage-dependent transfer rate constants for the gating variables. Together, these equations produce fast dynamics with respect to the membrane voltage that capture a wide array of neurocomputational properties such as integration of incoming spikes, accommodation to slow changing currents, and tonic spiking, continuous spikes in the presence of continuous applied current. With parameter tuning and additional ion channels, these conductance-based models can exhibit more complex behaviors such as spike bursting and phasic spiking, spikes only at the onset of continuous applied current.

Integrate-and-Fire Models

Although the HH-model realistically captures the underlying biophysical processes that generate spiking activity, due to its complexity and resolution, it is computationally prohibitive when scaled to networks of appreciable size (e.g. in the thousands of neurons). Much simpler neuron models such as the widely used Leaky Integrate-and-Fire model trades this computational cost at the expense of losing biophysical adherence as well as many of the neurocomputational properties exhibited by real neurons [23]. This model, as its name suggests, integrates the incoming current and spikes and resets once the voltage passes above a threshold, see eq. 2.2. Parameters to the model are $r, \tau_L$ the membrane resistance and leakage time constant of an
equivalent RC circuit, respectively.

\[ \tau_L \frac{dv}{dt} = -v + rI \]

if \( v \geq v_{\text{thresh}} \), then \( v = v_{\text{reset}} \) \hspace{1cm} (2.2)

Phenomenological Models

More recent neuron models such as the Izhikevich model and the Adaptive Exponential Integrate-and-Fire model contain additional states and dynamics that allow them to better capture the phenomenological behavior of the neuron \[24, 25\]. Most importantly, they include what can be considered as a recovery variable that enables increased adaptation to the input. Depending on the parameters chosen for these models, they are capable of exhibiting, effectively, all of the neurocomputational properties that are observed in the conductance-based models. While these phenomenological models do not capture the underlying mechanism, they are capable of producing accurate fits to the spike timing of real neurons while maintaining low computational cost. The dynamics of the Izhikevich model are given in eq. 2.3.

\[ \frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \]

\[ \frac{du}{dt} = a(bv - u) \]

if \( v \geq v_{\text{thresh}} \), then \( \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \) \hspace{1cm} (2.3)

In comparison to the conductance-based models, the fast voltage dynamics are now approximated by a quadratic function with a nonlinear reset. The gating variables are replaced by a single recovery variable, \( u \), and the parameters to the model, \( a, b, c, d \), are constant as opposed to voltage dependent. Computationally, the Izhikevich model is roughly two orders of magnitude less intense over the HH-model, measured in number of floating-point operations per second (FLOPS) needed to simulate the model.

2.1.2 Synapse

The synapse refers to the junction between the axon terminal of one neuron and the receiving dendrite of another neuron \[21\]. This is where the spike in the membrane voltage is transmitted electrochemically via neurotransmitters across the synaptic cleft, generating a current in the post-synaptic
neuron. Unlike the all-or-nothing neuron model, the synapse implements a weighting that affects the amount of applied current, mediated the amount of neurotransmitter released. Moreover, depending on the type of neurotransmitter released by the pre-synaptic neuron, the response may be excitatory or inhibitory. Because multiple neurons are incident at the post-synaptic neuron, the transmission of a spike is dependent on other factors such as pre-synaptic spike timing as well. With increasing network size, this results in increasingly complex behavior.

Synaptic Transmission

Fortunately, the model for generating currents for a single synapse is relatively simple [14]. This is because the summation of post-synaptic currents occurs closer to cell body, and the dynamics local to the synapses are decoupled. Biophysically, we model the rise and decay of a concentration of neurotransmitter at the synaptic cleft. Because the mechanism for generating current is the same as the conductance-based models for the neuron, the influx of ions at the receiving dendrite of the post-synaptic neuron, the form of eq. 2.4 is similar. The current depends on the time-varying conductance of the ion channels, \( g(t) \), and is weighted by the synaptic strength, \( w \).

\[
I_{\text{post}} = w g(t) (v - E_{\text{rev}}) \quad (2.4)
\]

\[
g(t) = g_{\text{syn}} \eta \left( e^{-t/\tau_{\text{decay}}} - e^{-t/\tau_{\text{rise}}} \right) \quad (2.5)
\]

The time evolution of the synaptic conductance is given in eq. 2.5, where \( g_{\text{syn}} \) is the maximal conductance, \( \eta \) is a normalizing factor such that the bracketed quantity has peak at 1, and \( \tau_{\text{rise}}, \tau_{\text{decay}} \) give the time constants of the release and re-uptake of the neurotransmitter, respectively. Due to the short time scales for the rise times, we may approximate the post-synaptic current as a step followed by the exponential decay. This allows for a single term in the neuron dynamics handling the summation of all the post-synaptic current contributions, eq. 2.6.

\[
\tau_{\text{decay}} \frac{dI_j}{dt} = -I_j + \sum_i w_{ij} g_{\text{syn}} \bar{\eta}(v(t) - E_{\text{rev}}) \sum_m \delta(t - t^m_i) \quad (2.6)
\]

The indices on \( w_{ij} \) correspond to the pre-synaptic, \( i \), and post-synaptic, \( j \), neurons, respectively, \( \bar{\eta} \) is an adjusted normalization term that accounts for the omission of the rise times, and the summation is over the spike trains of the pre-synaptic neurons.
Spike-Timing-Dependent Plasticity

Whereas the dynamics of the neuron model modify the membrane voltage, the dynamics of the synapse modify the weighting factor, and the resulting distribution of weights and synapse types affects the response of the post-synaptic neuron to incoming stimuli [26, 27]. In this way, the synapse provides a basic mechanism for memory storage at the cellular level, defining the input-output relationships for a neuron [28]. Learning, then, occurs with the modification of the synaptic weight.

The rules governing these modifications to synaptic weight depend on the spike timing of the pre-synaptic and post-synaptic neurons, giving it the name spike-timing-dependent plasticity (STDP). Historically, this notion was addressed by Hebb in the Hebbian plasticity model where neurons that spike nearby in time experience an increase in synaptic strength [13]. This has since been refined to incorporate the ordering in time in addition to the closeness. The basic STDP rule captures a causal relationship between neuron pairs, eq. 2.7. Increases to synaptic strength, by $A_{LTP}$, occur when the pre-synaptic neuron fires shortly before the post-synaptic neuron, leading to long-term potentiation of the synapse. On the other side, decreases to synaptic strength, by $A_{LTD}$, occur when the order is reversed, leading to long-term depression. Importantly, the pre-synaptic spike time is measured at its incidence at the synapse, not from the integration at the axon hillock. As a result, axonal delays also play a role in shaping the memory landscape.

As above, the indices on $w_{ij}$ correspond to the pre-synaptic and post-synaptic neurons, respectively, $t_i^m, t_j^n$ are the spike trains of the pre-synaptic and post-synaptic neurons, respectively, where $m, n \in \mathbb{N}$ index the individual spike times, and $\tau_+, \tau_-$ are time constants that determine how quickly the change in weight falls off as the spike times move further apart.

The above model of STDP is typically implemented in an online fashion by recording the spike history as traces, $m_i(t), n_j(t)$, that update by some amount, $a_+(m), a_-(n)$, during a pre-synaptic or post-synaptic spike, respectively, and otherwise decay over time. The change in synaptic weight is then a combination of these two traces via eq. 2.8.
\[ \tau_x \frac{dx_i}{dt} = -x_i + a_+(x_i) \sum_m \delta(t - t_m^i) \]
\[ \tau_\pm \frac{dy_j}{dt} = -y_j + a_-(y_j) \sum_n \delta(t - t_n^j) \]
\[ \frac{dw_{ij}}{dt} = A_{LTP}(w_{ij}(t)) \sum_n \delta(t - t_n^j) \]
\[ - A_{LTD}(w_{ij})y_j(t) \sum_m \delta(t - t_m^i) \]  \[[2.8]\]

Recent experimental studies have shown additional dependencies on the post-synaptic membrane voltage [29]. In particular, a spike triplet updating rule is observed, where potentiation of the synapse occurs in the presence of post-pre-post spike order instead of just the pre-post ordering. By incorporating a voltage dependence, the STDP model is able to accommodate for the effects of backpropagation of the post-synaptic action potential to the dendrites. The formulation of an online method, eq. 2.9, is similar to the basic STDP model above, with the main addition of low-pass filtered versions of the post-synaptic membrane voltage.

\[ \tau_\pm \frac{d\bar{v}_\pm}{dt} = -\bar{v}_\pm(t) + v(t) \]
\[ \tau_x \frac{d\bar{x}}{dt} = -\bar{x}(t) + X(t) \]
\[ \frac{dw}{dt} = A_{LTP}\bar{x}(t) Y(t) \left[ \bar{v}_+ - E_{rev} \right]_+ \]
\[ - A_{LTD}X(t) \left[ \bar{v}_- - E_{rev} \right]_+ \]  \[[2.9]\]

where \( X(t) = \sum_m \delta(t - t_m^i) \) and \( Y(t) = \sum_n \delta(t - t_n^j) \) are the pre-synaptic and post-synaptic spike times, respectively, and \( \bar{x}(t) \) is a low-pass filtered version of the pre-synaptic spikes with associated time constant \( \tau_x \). The low-pass filtered versions of the post-synaptic voltage, \( \bar{v}_+, \bar{v}_- \), with associated time constants \( \tau_+, \tau_- \) enter into the equations for both LTP and LTD via the reversal potential \( E_{rev} \), which generally corresponds to the resting potential. The operator \( [\cdot]_+ \) takes the maximum between \( \cdot \) and 0.

The usage of an online, event-based method for modifying the synaptic weights during simulation of large spiking neural networks is crucial to reducing the overall computational cost. By evaluating the synaptic dynamics only at the sparse spiking events as opposed to integrating a large, coupled system of differential equations, the FLOPS required per neuron is several orders of magnitude lower.
In addition to modification of synaptic strength by STDP, the synapse is also subject to modification of the plasticity rules themselves [30, 31]. That is, with the introduction of neurotransmitters such as dopamine or noradrenaline, the STDP curves are modulated. The effects vary with brain region as well as the neurotransmitter modulating the changes, in some cases enabling plasticity where there was none. With dopamine, the main effect exhibited is an overall facilitation in increasing synaptic strength, irrespective of the order in the spike timing. During a learning task, the dopamine system responds to discrepancies between expected outcomes and actually observed outcomes, serving as an indication of prediction error [32]. Because the synaptic connections are what drive the response of a neural network to incoming stimuli, we may consider this release of dopamine as a reward-based mediator to learning. While the mechanisms of this phenomenon are yet unclear, and the projection of these local changes to the network level even more so, it is certain that the neuromodulation of synaptic plasticity will play an important role in developing more complete models of learning.

2.2 Learning and Memory

We may draw several parallels between the behaviors that are observed at a high level with those mechanisms found at the level of the neural substrate. With respect to learning, the dominant theory of how the brain processes and learns from the environment is through reinforcement. At the behavioral level, classical conditioning captures the idea of a strictly associative reinforcement [33]. In Pavlov’s experiments, dogs were simultaneously given two distinct stimuli, the sound of a bell and the presentation of food. The involuntary response of the dog to food (i.e., salivation) was later found to also occur with just the sound of the bell where it had not been exhibited previously. Projecting to the level of the neural substrate, association of stimuli has its analog in the mechanisms for synaptic plasticity, where neurons that are “repeatedly active at the same time” are strengthened in the Hebbian sense [13]. Another type of reinforcement at the behavioral level is operant conditioning [34]. Unlike classical conditioning, where the association concerns involuntary responses, operant conditioning is concerned with the learning of voluntary behavior. This is achieved through the application of a reward or punishment as a response to desired or undesired behavior. Through reinforcement, the agent learns the association between
an action and its consequence, leading to a change in the agent’s behavior. This application of a reward signal is expressed in the neural substrate via the modulation of STDP through neurotransmitters.

Because these behavioral models have some projection onto the level of the neural substrate, they help to guide the search onto how the substrate might be driven to implement certain algorithms at a higher level. That is, they give insight into what we might expect to see at the network level, enabling us to more intelligently analyze the system. Mathematical models of reinforcement and association are of particular interest to us. Although they may not directly model the biology, like the phenomenological models of the spiking neuron, they are able to capture aspects of the overall function. As developments are made coming from a bottom-up approach, the aim is to inform and refine these high level models of learning and memory to better reflect mental processes. In this way, we may abstract out the need for a neural substrate while maintaining the relevant function.

2.2.1 Reinforcement Learning

The general model for reinforcement learning involves the learning of a utility function over the environment such that actions taken by the agent maximize the amount of expected reward [35]. Without any preconceptions on the environment, the agent learns through a process of exploration and exploitation. Typically, there is a discounting factor that preferentially gives less weight to rewards that are further away in some sense.

Mathematically, the reinforcement learning problem is typically formulated as a Markov decision process, MDP. Its construction involves a set of states $s \in S$ that determine where the agent is in the environment, a set of actions $a \in A(s)$ that the agent is able to take in a given state, a transition function $T(s'|s, a): S \times A \mapsto S$ that determines how the agent moves from state to state depending on its action, the reward $R(s)$ given to the agent when it arrives at a state, and a policy $\pi: S \mapsto A$ for which action to take in which state. The utility function $U^\pi$ is given by eq. 2.10, where depending on the policy, the expected reward may change.

$$U^\pi(s) = E \left[ \sum_{t=0}^{T} \gamma^t R(s_t) \right] \quad \text{where } s_0 = s \quad (2.10)$$

The discount factor $\gamma$ determines the weighting of rewards farther in the future, and the horizon $T$, which may be either finite or infinite, determines how far into the future the agent computes the expected reward estimate.
During reinforcement learning, the utility function is learned through exploration of the states. Additionally, the agent performs exploitation of the learned utility by traveling along the state trajectories it believes to maximize the expected reward.

Temporal Difference Method

Direct methods for finding the utility use dynamic programming to solve the underlying MDP. Unfortunately, this requires the estimation of the state transition probabilities in addition to finding the policy.

A popular method for learning the utility function without the need for a transition model is temporal difference [36]. Instead of computing utility as the expected reward of a state through the summation in eq. 2.10, the method of temporal differences uses previously estimated values for the utility. The adjustment of the existing utility estimate is given by eq. 2.11, where $\alpha$ is the learning rate and the bracketed quantity is referred to as the prediction error, as it is the difference between the actual utility and the estimated utility obtained by traveling to state $s$.

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha [R(s) + \gamma U^\pi(s') - U^\pi(s)]$$  \hspace{1cm} (2.11)

There are several variants of reinforcement learning that use the temporal difference method to perform policy estimation. Q-Learning is of particular interest because it collapses the utility function into a state-action pair $Q(s, a) : S \times A \mapsto \mathbb{R}$ [37]. With respect to the neural substrate, the amount of neurotransmitter released at the synapse in response to the spiking of the pre-synaptic neuron can be thought to correspond to the action that maximizes $Q$. Here, the reward is computed from the spike history at that synapse and is split between maximizing the correlation between stimuli and minimizing metabolic costs. As a result, the update of $Q$, and thus the learning of the policy $\arg\max_a Q(s, a)$, is analogous with the effects of STDP. The update rule for Q-Learning is given in eq. 2.12. It is similar to eq. 2.11, with the difference being that the policy is made more explicit and the reward being a result of the current action as opposed to the current state.

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[ R(s') + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$  \hspace{1cm} (2.12)
2.2.2 Associative Memory

The ability of our brain to associate related, and sometimes unrelated, signals spans multiple scales [38]. At the level of the neural substrate, neurons or groups of neurons responsible in the identification of external stimuli share connections among each other via a spatially preserving topology. Traveling up the neural pathways, these shared connections allow for associations among the various sensory modalities. At the behavioral level, the brain implements a system that is capable of recalling related concepts and the ability to perform analogical reasoning. With respect to memory, associativity refers to the property where the activation of components of an aggregate state facilitates the activation of associated components of that aggregate.

In the framework of dynamical systems, this may be thought of as the movement on a manifold toward an attractor state or stable limit cycle [39]. Here, associative recall starts when the push from component stimuli is sufficient to drive the system toward its attractor. The learned internal dynamics of the system then generate multistationary stable periodic trajectories among the attractor for the component, the attractor for the aggregate, and that of other components. A simple example of multi-modal associativity is the visual recall of what an object (e.g. an apple) looks like when you hear or read the word for that object.

Cascaded Hidden Markov Model

A powerful method for implementing an associative memory is through cascading multiple hidden Markov models (HMMs) by imposing a hierarchical structure, where the outputs of HMMs lower in the hierarchy are used as the inputs to HMMs higher in the hierarchy [40]. With respect to an embodied agent, these HMMs may correspond to the various sensory modalities at the base of the hierarchy and multi-modal sensory or motor associations traversing up the hierarchy.

The construction of a hidden Markov model assumes two interrelated mechanisms, implementing a stochastic signal model over a set of hidden states characterized by a Markov process where each state observes a probabilistic output [41]. In the discrete case, the sequence for of hidden states $Z = z_1, z_2, \ldots, z_T$ and observations $Y = y_1, y_2, \ldots, y_T$ for times $t \in [1, 2, \ldots, T]$ are parameterized by a state transition matrix, $A : Z \mapsto Z$, with initial state distribution $\pi$ and a state observation matrix, $B : Z \mapsto Y$, respectively. Given this parameterization, the model may perform state esti-
information at a given time from the observation sequence by computing the posterior conditional probabilities via the forward-backward algorithm. Applying dynamic programming to the observation sequence, we may also compute the most likely sequence of states via the Viterbi algorithm. Additionally, in the absence of known parameters, the model allows for parameter estimation from an observation sequence by performing estimation-maximization via the Baum-welch algorithm. This is extended to enable online learning via the recursive maximum-likelihood estimation algorithm.

In the cascaded form, the model shares many elements of the biological system. First, we observe temporal structure in the incoming signals. This is captured by modeling the underlying hidden states responsible for these signals as a Markov process. In terms of representation, this allows the model to detect spatio-temporal patterns such as the frequency sweeps found in speech. Second, the internal state is symbolic, where we may additionally attach higher order meaning to the incoming signal. At higher levels of the cascaded HMM, these would represent aggregate concepts such as the apple example above. Third, the internal model is learned through an iterative process. For a single HMM, this corresponds to associating a signal to its hidden state by estimating the model parameters from the observations. This parallels the contraction performed by the temporal difference method in reinforcement learning. When the signals to the HMM are the states of other HMMs, this association becomes tying component states to an aggregate state. Fourth, the model is generative, meaning that given learned parameters, the model is able to reconstruct a state trajectory that produces the observed signal. With respect to the activation of an HMM at higher levels of the hierarchy, this enables multiple forms of recall at the lower levels. That is, it completes the associative process by facilitating the activation of component states from the activation of an aggregate state by other component states.

2.3 Spiking Neural Networks

The question of what algorithms are implemented at the level of the neural network is still not well understood. With our knowledge of the mechanisms underlying a spiking neural network and the observed emergent behaviors, the goal is to bridge the gap in our understanding between these two levels of abstraction. In much the same way that models of the spiking neuron were established, we wish to provide a reduced order, yet phenomenologi-
cally accurate, model of the network behavior by comparison and validation with an ab initio implementation. With respect to the network, the element that we are most concerned with is the representation of information. In addition to the temporal component of spike timing, the spiking neural network also admits a spatial component in its topology. At the level of the neural substrate, storage of information about the environment is in the synapse. Understanding how this projects onto how information is stored in the network is critical in extracting the algorithms that process this information [42].

Current models at the level of the spiking neural network span its organization with respect to sensory input, the encoding of motor output, as well as the intermediary, potentially symbolic representation. Although the mathematical constructions of these models vary, they all are predicated on the requirement of multiple neurons in achieving collective functions that are unattainable with isolated cells. With respect to experimental verification, some of these models have found reasonable agreement to real neural networks. We discuss a few of these successful models below. Briefly, these include the spatially preserving topological mapping of the receptive fields, the population vector inducing movement direction, and the polychronization of spiking activity by groups of neurons in a network.

2.3.1 Topological Mapping

The organization of neurons receptive to external stimuli is not random. Rather, the structure of the world projects onto structure in the organization of the receptive fields [21]. Signals that are closer to each other, with respect to some metric, in the environment are encoded by neurons that are closer to each other physically. Moreover, this mapping has been shown to be preserved along the way as well as in regions of the cortex. Examples of this may be found in both the auditory and visual systems. In the former, this is expressed as a tonotopic map where the metric is frequency, and in the latter, this is expressed as a retinotopic map where the metric is spatial distance in the visual field.

There are many advantages to encoding the stimuli from the environment using a spatially preserving map in the neural system. The major advantage is that signals that are closer with respect to some metric tend to be related to each other in an information-theoretic sense [43, 44]. In particular, the
mutual information of two signals, $X, Y$, is described by eq. 2.13.

$$I(X, Y) = H(X) - H(X|Y)$$

(2.13)

Here, $H(X)$ is a measure of entropy, or uncertainty in the information communicated by the signal $X$, and $H(X|Y)$ is the uncertainty in $X$ conditioned on knowledge of the signal $Y$. For two related signals, $H(X|Y) < H(X)$ which leads to the mutual information measure $I(X, Y) > 0$. The spatial mapping allows for information encoded by groups of neurons of a given signal to be accessed by the neurons responsible for encoding a related signal. With respect to the physiology, this allows for decreased entropy in the signal under metabolic constraints, as the spatially preserving topological mapping reduces the average necessary axon length. With respect to the visual system, the retinotopic map preserves the continuity of objects, starting from the center-surround receptive fields in the retina to the detection of orientation in the primary visual cortex. In the auditory system, the tonotopic map preserving frequencies allow for greater precision in detection through the inhibition of neighboring frequencies at the cochlea, and in the primary auditory cortex, a spatial-temporal mapping enables the detection of frequency sweeps.

While the topological structures of sensory neurons may share commonalities, the methods by which stimuli from the environment are encoded are many and varied [21]. Because single neurons are generally unreliable, coding of any particular stimuli is handled by groups or populations of often overlapping neurons. A common model for coding is through firing rate. In rate coding, the stronger the stimuli, the faster neurons that are responsible for encoding that stimuli fire. Experimentally, this is given by the peri-stimulus time histogram where firing rates are determined by finding the probability of the number of spikes that fire within a certain time interval through averaging over a number of trials. Related to this is frequency locking, found primarily in the auditory system. Whereas in rate coding, the inter-spike interval may vary from spike to spike, with frequency locking, as its name suggests, the spike trains of the relevant neurons are locked to the frequency of the stimuli. For frequencies above the maximum firing rate of a neuron, what has been observed are spike trains aligned to the subharmonics. Temporal coding is another major model for how neurons encode certain classes of stimuli where rate coding may be insufficient. Unlike rate coding, the precise timing of temporally coded spikes is important. An example of where temporal coding occurs is in the onset of stimuli, important
to sound localization from binaural input and spatial structure in the visual field.

2.3.2 Population Vector Model

The population vector model describes how continuous state representation may be implemented by a population of neurons [7]. Its formulation begins with a population of $n$ neurons where each neuron codes for a particular preferred direction vector on a $k$-dimensional space, $k < n$, spanned by the population. The population vector is then found by taking a weighted sum of the neural basis vectors with respect to firing rate. For a desired reference direction $x \in \mathbb{R}^k$ the resulting population vector $\hat{x} \in \mathbb{R}^k$ is given by eq. 2.14. The weighting $w_i(x)$ corresponds to the contribution by the $i$th neuron with preferred direction vector $x_i \in \mathbb{R}^k$. The preferred direction for a given neuron is soft, overlapping with that of other neurons in the population. That is, the weighting function $w_i(x)$ admits a continuous distribution over $x$ as opposed to a impulse at $x = x_i$.

$$\hat{x} = \sum_{i=1}^{n} w_i(x)x_i \quad (2.14)$$

Based on the firing rates of the neurons in a population with known preferred directions, a prediction can be made about the population vector. Experimentally, this model for prediction has been used to accurately estimate the direction of movement from spiking activity in the motor cortex. Given a population of neurons where the individual preferred directions are unknown, they may be found by performing least-squares estimation, minimizing eq. 2.15.

$$E = \frac{1}{2} \int_{\mathbb{R}^k} \left(x - \sum_{i} w_i(x)x_i\right)^2 dx \quad (2.15)$$

Based on this representation, populations of neurons may be thought to be performing vector computations in their mathematical abstraction, from which we may approach analysis from a linear algebra standpoint. Although a promising representation of the spiking activity, the model possesses certain drawbacks that prevent immediate generalization to the biology. Primarily, the model does not take into consideration the network, leaving out the function of the synapses while the neurons are treated in isolation. That is, how a population of $n$ neurons physically implements the lower $k$-dimensional representation, whether this is through a spatially driven topo-
logical structure in the same sense as that found the sensory system, is unclear. Additionally, the model treats the population as a homogeneous collection of excitatory type neurons, and the interaction of interneurons performing inhibition and how that shapes the firing rates is not captured. In spite of these drawbacks, however, the population vector model provides an invaluable guide as to what behaviors we should expect to extract from a spiking neural network.

2.3.3 Polychronization

The phenomenon of polychronization is another such behavior exhibited by spiking neural networks [8]. Unlike the population vector model, which uses rate coding, polychronization in a network rests on top of precisely timed spiking. The term *polychronous* refers to time-locked spiking behavior among a group of neurons, and is distinguished from synchronous in that the behavior spans multiple times, and polysynchronous which refers to multiple sub-groups of synchronous spiking. Polychronization in a network results from the interplay among inhibition, axonal delays, and synaptic plasticity. It exists in the form of stable polychronous neural groups that exhibit reproducible, time-locked patterns of spiking within millisecond precision. Although current models of polychronization rest on top of randomly connected networks, the existence of precisely timed spatio-temporal patterns of spiking has been observed experimentally in dissociated cell cultures [45]. However, the extension of the model, perhaps through shaping the effective topology of the network through synaptic plasticity, in capturing the laminar and columnar microstructure found in the brain is still unclear.

At a high level, polychronous firing shares similarity with the idea of a cell assembly, a network of neurons that tend toward spiking as a collective due to its strong synaptic connections. This is extended by the inclusion of axonal delay and inhibition which enables a spiking neural network to exhibit multiple polychronous groups. Due to overlap in neurons belonging to multiple groups, the number of groups that exist in a network may well exceed the number of neurons in the same network. Moreover, a neuron may participate in the activation of multiple polychronous groups of which it belongs to in the same time period, with each group spontaneously recurring on the order of multiple seconds.

With respect to representation, the activation of a polychronous neural group resulting from incoming stimuli parallels the signal to symbol transformation by an HMM. Like the firing of a neuron at the level of the neural
substrate, polychronous firing can be thought to be the information bearing signal at the level of the network. Because the spiking activity of a polychronous group exhibits a particular spatio-temporal pattern similar to that of incoming stimuli, the extension to cascaded processing is straightforward. Following the coexistence of multiple active groups, mechanisms for binding related groups form the basis of a neural implementation of working memory.

2.4 Simulation Tools

Study of spiking neural networks with resolution at the level of the neural substrate requires effective simulation [6]. Traditional methods for obtaining electrophysiological recordings from neurons are simply not suitable for working with these systems at scale. Patch clamping, while a valuable tool in studying the individual ion currents of a cell, involves the puncturing of the cell membrane to obtain recordings. In addition to requiring this to be performed on an individual cell basis, it is also extremely difficult to patch clamp more than one neuron at a time, much less that of an entire cell culture. More recently, the development of multi-electrode arrays (MEAs) has advanced the study of networks considerably through the electrical recording and stimulation of multiple sites in the network. This is done by measuring the extracellular field potentials generated by spiking activity of the neurons near the recording site and interfacing with specialized acquisition and analysis software such as Neuralynx or MeaBench. Because these devices are non-invasive, they enable experiments that range over an extended period of time, providing the ability to monitor the evolution of a network. In terms of resolution, these systems typically provide 64 electrode sites in vitro with some in vivo probes going up to 256 sites. Compared to the volume of cells in the network, on the order of tens to hundreds of thousands, even for two-dimensional monolayer cultures, this is extremely low.

The constraints of using a live cell culture impose many additional difficulties. Great care must be taken to ensure that the cells survive for the duration of the experiment. These procedures include, but are certainly not limited to, the replacement of supporting media containing the appropriate nutrients, the use of incubators to maintain the proper temperature and humidity levels, and controlling for and operating in a sterile environment to prevent infection. The major advantage of studying neural cell cultures or brain tissue is that they do require the additional biophysical modeling that
simulation tools must adhere to in order to be accurate to the underlying physiology. Additionally, with the ability for both recording and stimulation, the use of MEAs in experiments that close the loop on the motor-sensory system enables real-time experiments on how embodiment shapes network activity and development [46, 47]. Experiments of this nature that focus on learning and representation in a spiking neural network are of particular interest to us.

The goal of neural simulation is to provide a framework for study with respect to the behavior of the biological system while freeing experimentation from many of the constraints that a physical system imposes. We review a selection of existing simulation tools in terms of their capabilities and shortcomings in addressing the study of spiking neural networks.

2.4.1 NEURON

The NEURON simulation environment is mainly focused on providing a domain-specific tool for modeling empirically based computational models of neurons and their networks [16]. To this end, NEURON provides a great degree of conceptual control with respect to model specification. Without having to deal with the underlying data structures, users are able to define cells in terms of the differential equations governing the gating properties of ion channels or assemble different sections in a cell simply by specifying the topologies and geometries. Furthermore, NEURON comes with a graphical user interface (GUI) that provides a large number of tools with respect to constructing, simulating, and analyzing models. The advantages of decoupling the ‘natural syntax’ of the neuron with the simulation code are twofold. First, it enables users who are less familiar with programming to conduct simulated experiments where the computational model accurately reflects the hypothesis formulation. Second, it enables greater computational flexibility in the algorithms that solve the model system equations for increased robustness, accuracy, and efficiency. For greater computational control, users may also write hoc or python scripts with the appropriate language extensions.

Although the domain of NEURON has since expanded from single-cell models, its capabilities with respect to network modeling are limited. For network construction, the user is typically expected to modify multiple hoc scripts generated through the GUI with loops to establish the necessary synaptic connections and associated parameters. Parallel simulation is implemented by NEURON through the addition of parallel classes that oper-
ate over Message Passing Interface (MPI), a standardized message-passing library specification. The ParallelNetManager class, for example, allows the user to set up and manage a larger network simulation on a cluster of workstations. Similar to network construction, however, the user is expected to write the hoc script that specifies how the parallel code should be run. For anyone unfamiliar with the elements of parallel programming, this is prohibitive in terms of generating a working and efficient simulation. That is, we no longer see a decoupling of the model specification with the underlying computational engine for large networks of spiking neurons.

2.4.2 NEST

The NEST simulation tool is aimed at simulating large networks of neurons with biologically realistic connectivity [18]. Unlike the conductance-based and morphologically precise models developed in a system like NEURON, NEST treats individual cells as point-like objects similar to the phenomenological models in sec. 2.1.1. This is because NEST is focused more on the dynamics and structure of neural systems at scale, trading complexity at the substrate level for complexity at the network level. At this scale, computation is dominated by the synaptic connections for which networks on the order of $10^5$ neurons have $10^9$ synapses. This alone requires a distributed representation. The key challenges are in efficiency, not just in representing the connections but also with transmitting the spiking events between neurons. NEST addresses these challenges by implementing a hybrid parallelization strategy using message-passing (MPI) across nodes and multithreading (pthreads) across the CPU cores of a node. Additionally, by exploiting efficient caching, NEST has been shown to exhibit supra-linear scaling for distributed network simulations.

For network construction and programming, NEST provides a high-level scripting language called SLI which contains several functions for creating and connecting networks. Python scripting in the form of PyNEST is also available, although its support of the different MPI implementations is limited. In terms of user interaction, emphasis is placed on providing the appropriate functions in a scripting language as opposed to a graphical interface. The rationale is that network specification of a large network is procedural in nature and that data analysis is generally performed off-line. In terms of the neural substrate, NEST comes with many built-in neuron and synapse types as well as modules to stimulate and record from the network. Computationally, each module implements its own solver independent of that of
the other modules in the network. As long as the basic interfacing functions to the kernel are defined, this gives each module the same type of computational flexibility that was found in NEURON. User developed modules may also be added to the simulation kernel.

While the parallel infrastructure of NEST is fairly developed, there are a few critical shortcomings that prevent it from carrying out experiments similar to those performed on an MEA. Perhaps most importantly, there is no ability to save network state between different simulation runs, prohibiting longer-term experiments on the time scales that are relevant to real neural networks. Furthermore, even without interfaces with real-world hardware, the ability to save state enables the user to analyze and control the stimuli fed into the network based off of observed output. This would effectively allow for closed-loop learning experiments. With respect to the neural substrate, the departure from biologically faithful models of the cell is a step too far in the right direction. In particular, NEST treats the nodes (neurons) and connections (synapses) more as separate entities than as components of a unified cell. For synapse models where the state of the neuron is important in determining the postsynaptic potential, all computation is performed by the module for the neuron, whereas the module for the synapse simply passes along the delay and weight of the connection. Updates to the synaptic weight are also cumbersome in that they must access ‘archived’ spike information from the neuron in order to compute the spike-timing dependency. Voltage-dependency in the synaptic update, which accounts for back propagation of the postsynaptic spike (see sec. 2.1.2), is not available under this scheme.
CHAPTER 3

FRAMEWORK FOR STUDY

The main goal of STACS is to provide a suitable framework for studying large networks of spiking neurons exhibiting adaptive behavior as part of a closed-loop system. To this end, we borrow certain design strategies that are successful in the existing set of tools as well as adapt ideas from other areas to the context of neural simulation. Similar to how NEURON provides a great deal of conceptual control over its specific domain of study, STACS aims to describe the network and the neural substrate in its ‘natural syntax’. Comparable to NEST, and as a matter of practicality, STACS also aims to provide both accuracy and efficiency in its parallel infrastructure. On top of the simulation engine, STACS also utilizes a portable communication protocol for interaction with external devices, both sensory and motor, enabling the development of user defined interfaces. In doing so, we hope to provide a valuable tool in dealing with the different levels of abstraction with respect to network and its substrate, supplementing experimentation of live cell cultures on MEAs, and developing a more complete understanding of the emergent behavior of a spiking neural network.

There are several key challenges and considerations that must be met in realizing these ideas. Primarily, we must solve the problem of representation. That is, how we should translate the natural syntax of the network into something that may be manipulated computationally. Although there are certainly general architectural and algorithmic concepts that are important in the development of any application, the best results in terms of capabilities and performance are only achieved by paying attention to the particular problem domain. As the target platform for STACS is multi-core and multi-node, thinking under a parallel paradigm from the beginning is important in addressing concerns of modularity, efficiency, and scalability. With respect to the underlying biological processes of the neural substrate, we must also ensure that the computational model is biologically faithful and at the same time computationally feasible. Because the simulation tool is ultimately to be used for analysis, its ability to query the network should
be broad in scope and flexible to extension.

We review how STACS addresses these challenges in the following sections. To begin, we explore the design philosophies and strategies that form the basis of our computation and communication model. We then tackle the problem of representation accordingly, with the goal of a general formulation that is specific to the problem domain. Finally, we discuss the core components of the resulting simulation engine that spans the multiple layers of abstraction in a spiking neural network as well as provide an illustrative example of a neural network simulation.

3.1 Design Philosophy

The design philosophies underlying the construction of STACS embody a few main ideas: efficiency, accuracy, and practicality. As a matter of scale, the spiking neural network must be distributed across multiple processes in order to be computationally tractable. In response to this, STACS is designed to be parallel from the ground up to enable the most appropriate use of resources. In a similar light, STACS admits a preference toward minimalism where possible, maintaining only as much as is necessary to complete a given task. This enables the algorithmic flow of STACS to be both lightweight and streamlined to prevent wasting resources. Of course, this is balanced with careful attention to the problem domain to prevent sacrificing accuracy with respect to the underlying biological processes. In order to provide conceptual control, STACS decouples the code responsible for simulation and language specific to the problem domain. This not only allows for accuracy between different methods of integration by different substrate models, but also encourages thinking in terms of the problem domain as opposed to implementation specifics.

3.1.1 Parallelization

In a typical cell culture on an MEA, there will be on the order of $10^5$ to $10^6$ neurons with roughly $10^3$ to $10^4$ synapses per neuron. Already, we begin to reach memory limits on a single machine merely in terms of storing the network state [48]. For an average spiking rate of 50 Hz, the network will experience on the whole $10^7$ propagating spike events per ms. At this scale and larger, a distributed method of processing is required not only to accommodate the problem size, but also to accelerate the wall-clock time of the
simulation to a practical time scale for experimentation. Efficient utilization of parallel resources involves at a minimum overlapping computation with communication where possible. This is achieved by STACS using a paradigm of asynchronous parallel objects encapsulating self-contained partitions of the network. With respect to the communication patterns specific to a spiking neural network, aggregation of messages and attention to locality between processes where possible is also performed in order to reduce overhead.

Efficiency

To provide a basis for comparing the effectiveness of a parallel algorithm relative to its serial counterpart, we use the concept of efficiency [49].

\[ E_p = \frac{C_1}{C_p} = \frac{T_1}{pT_p} = \frac{W_1 V(M/p)}{W_p V(M)} \]  

The parallel efficiency, \( E_p \), of an algorithm is defined to be the ratio of the serial cost, \( C_1 \), to the parallel cost, \( C_p \), where cost is in processor-seconds. That is, cost is measured as \( p \times T_p \) where \( p \) is the number of processors utilized and \( T_p \) is the computation time measured as the wall-clock between the beginning and the end of the computation. Computation time may also be modeled as the ratio between the total number of operations required for the problem and the operations per unit time capable by the processor: \( T = W/V(M) \). Here, the processor speed, \( V \), is a factor of the amount of storage required for the given problem, \( M \). This is due to the effects of a memory hierarchy in real systems where effective caching will greatly improve the processor speed.

Apart from a potential increase in efficiency due to faster memory utilization, the efficiency of a parallel algorithm may only degrade, all else being equal. This is because parallelization introduces overhead, additional work not present in the corresponding serial implementation, by nature of having to communicate between processors. Other factors that decrease efficiency are the potential lack of processor concurrency, a measure of how many processors are active simultaneously, and load balancing, the distribution of the work across the processors. As a result, the design of an efficient parallel algorithm attempts to exploit the memory hierarchy, minimize the communication overhead, maximize processor concurrency, and minimize load imbalance across the processors.
Asynchronous Parallel Objects

The primary goal of partitioning the network into a collection of self-contained parallel objects is to better utilize the memory hierarchy. Fortunately, because the network states between different neuron and synapse models are mostly independent of one another, save for the occurrence of spiking events, their separation is relatively straightforward. The only caveat is that the dynamics of the afferent synapse to a neuron may in some circumstances depend on the state of the neuron. This suggests that synapse state should reside on the same parallel object as the associated neuron state to improve memory locality and prevent unnecessary communication.

Depending on their size and distribution across the available processors we may also mitigate losses to efficiency by poor load balancing or concurrency [50]. Traditionally, parallel algorithms typically entertain one partition of the problem space per processor where an algorithm is split into multiple computation-communication phases, broken up by global synchronization operations. That is, processors cycle between computing on local data and communicating any data dependencies in a given iteration. Under this framework, the parallel work may be modeled (eq. 3.2) as the contributions of time spent during computation, time exchanging messages, and idle time due to potential load imbalances.

\[ W_p = W_{\text{comp}} + W_{\text{comm}} + W_{\text{idle}} \] (3.2)

Unfortunately, this leads to less than ideal processor utilization due to time spent not performing relevant computation. Furthermore, computation may only progress as fast as the slowest process/processor. For a poorly load balanced system, this introduces a significant amount of idle time as processors wait on data dependencies. As the problem size increases, we also see a drop in efficiency due to poor scaling with respect to any global operations.

By over-decomposing the problem space to multiple parallel objects per processor, we reduce the amount of time spent idle simply by virtue of having multiple processes on the same processor: when one process waits on data, another may compute. With respect to sending and receiving messages, this paradigm also enables more uniform usage of the physical communication layer connecting the processes as there are no longer phases of bulk communication. Rather, the messages are sent immediately after they are generated and processed only in the absence of pending computation. As a result, the asynchronous model masks latencies in the communication layer.
by overlapping the time with computation, reducing the effective communication time. Because this paradigm emphasizes local dependencies over global synchronization, we also achieve better scaling as the problem size increases. A conceptual comparison between the two parallel paradigms is given in fig. 3.1. More information may be found in section 3.3.1.

Figure 3.1: Parallel work: Traditional approach with distinct computation-communication phases (top); Asynchronous parallel objects approach with overlapped computation-communication (bottom).

Message Aggregation

Given the heavy communication load for a spiking neural network, the goal of message aggregation is to minimize the communication overhead as well as the total volume of messages that must be sent between processors. In doing so, we also reduce congestion in the communication layer as messages no longer have to compete over bandwidth simultaneously. The amount of time required to send a message across the communication layer is modeled by eq. 3.3, where $t_s$ is the startup time or latency for a message, and $t_w$ is the transfer time per word over a communication link ($1/t_w$ is the bandwidth) [50]. For messages sent concurrently over the same communication link, the time required per message is both a function of the average message length, $L$, and the number of contending messages, $S$. 

30
\[ T_{\text{msg}} = t_s + t_w SL \]  

(3.3)

For most parallel platforms, \( t_s \gg t_w \), meaning that sending a single, larger message is typically faster than sending multiple, smaller messages. Although the paradigm of asynchronous parallel objects minimizes communication concurrency between processes, the communication load for a single process is still considerable. In the context of a spiking neural network the communication pattern sees a large number of spiking events that need to be broadcast where the amount of data per event is small. Because multiple connections may exist between the neurons of a given pair of partitions, message aggregation is particularly important to communication efficiency.

3.1.2 Minimalism

In order to produce as lightweight and streamlined a simulation as possible, we aim to reduce the amount of redundancy both in the data requirements per parallel object and the data that must be communicated between objects. Fortunately, by nature of an event-based communication scheme, the separation between the type of data that is stored and sent is fairly straightforward. The majority of the data that is kept local to a parallel object encompasses the state information of that partition of the network, whereas the messages sent between objects should carry only as much information as is necessary to produce the intended behavior. Although the separation between these roles is clear, there are several instances in which data glut may be prevented. In the most general sense, data dependencies required for computation may be classified as either network state or network events.

Network State

In terms of the neural substrate, state information must be stored for both the neuron and the synapse models. Additionally, network level information such as the afferent and efferent connections should also be maintained. These two levels of abstraction together enable each partition of the spiking neural network to perform the necessary operations for state update, generating and sending events, and receiving external events. With respect to state information, because there is no overlap of cells between partitions, we do not need to provide data buffers at the boundaries of the partition. Rather, we require only a single copy of any stationary data. Furthermore,
any redundancies in this stationary data should be minimized to the extent that it encourages more effective utilization of the memory hierarchy, in particular the cache.

One notable example of where this is applicable is in the specification of the substrate models. Generally, these models provide a set of parameters that determine the behavior of the model dynamics along with a set of states that evolve according to these dynamics (sec. 3.2.3). Because only the states are variable across the lifecycle of the simulation, the procedure is to store only one copy of the parameters per substrate type. Because a number of the substrate models are heavily parameterized by physical quantities like maximal conductances, resting potentials, and time constants, duplicating these values across each individual neuron or synapse only serves to inflate the memory footprint unnecessarily. Moreover, because each copy of these parameters would need to be loaded into cache per individual network element, we will incur more frequent access from slower memory.

The other major area in which we may improve cache utilization is through carefully laying out the data in memory. In particular, the goal is to accommodate prefetching of relevant data into cache for subsequent state updates as the current computation is progressing. Just as the overlap between computation and communication masks any latencies in the communication layer of the parallel platform, the overlap of computation with memory loads avoids idle execution cycles.

Network Events

The communication scheme rests on top of the spiking events between neurons. At a minimum, a network event requires a timestamp along with some sort of identifier of where the event originated so that it may be processed accordingly. Just as with the network state, and on top of the message aggregation strategies discussed in section 3.1.1, there are areas in which we may reduce the amount of data that must be sent across the communication layer. Although the event-based communication scheme is heavily branched in terms of the number of targets per origin, it is sufficient to provide a single copy of the event data per network partition possessing multiple targets for said event. Thus, the strategy for event communication is to broadcast only the common event data while any target specific data, such as delay times, is stored on the receiving network partition.

With respect to the processing of events during the state update, the concept of an ordered event list is particularly useful in achieving effective
utilization of the memory hierarchy [51]. As network elements steps through their state update, the event list is accessed sequentially. However, this implies a gap between the type of data structure responsible for sending events across the communication layer and that of how these events are eventually processed. As a result of these two strategies, a method for transfer between receiving events and processing events must be devised. Fortunately, this may be overlapped with the computation responsible for any target specific modifications. Redundancies in this transfer process that should be addressed include ways of accessing the incoming event data, the target specific data, and the generation of event lists from multiple messages.

3.1.3 Conceptual Control

More important than the design strategies guiding the implementation of a framework for study is its ability to reason about the problem domain. That is, we wish to enable and encourage asking the right questions and providing convenient tools with which to answer them. This entails the formulation of a language in the natural syntax with respect to a spiking neural network, encompassing both language targeted at the neural substrate and that of the network as a whole. Moreover, this language should be decoupled from the specific implementation in order to provide computational flexibility with respect to the representation rather than be restricted to a rigid construction. In essence, we aim to capture as general a structure as possible without straying significantly from the problem domain. Finding the right balance enables the details of a particular hypothesis to be specified by the researcher without sacrificing performance through lack of focus. Ideally, the language should entertain the idea of modular extensions on top of a core functionality.

At the heart of the framework is the representation of the network. Before anything else, we must provide a way to construct the network topology. After all, a collection of neuron and synapse models without organization of their interaction is meaningless in the context of studying their collective behavior. With respect to this, we may categorize rules for building the topology as either local or global constructs. Locally, we are concerned with the neuromorphology, how any individual neuron is connected to its neighbors depending on properties such as neuron type and their spatial proximity. Globally, we are concerned with system characteristics such as the distribution of neuron types, cell density, and sensory-motor interfaces. Extensibility to changes in the network topology as the simulation progresses
is also important to longer term experiments that explore synaptic growth and pruning. Furthermore, we distinguish the language used in describing and analyzing the network with that of the neural substrate composing it, enabling the treatment of connectivity patterns independent of the specific neuron or synapse models. This distinction additionally opens up opportunity to study the effects of different substrate models on the emergent network level behavior.

In terms of analysis, language that permits and facilitates collective operations on the network or subsets thereof is critical in formulating metrics that may provide greater insight into the gross behaviors at a network level. The key insight motivating this construction is that although the behaviors of the neural substrate are important, individual elements are unable to capture the degree of complexity that is observed at the level of the network. Rather it must be the interaction of many such elements that contributes to network phenomena, and a shift in focus is necessary for the extraction of these phenomena. Additionally, collective operations are a natural way of applying stimuli to the network. Examples of stimuli include the application of current to a localized area of the network mimicking that of an MEA probe or the application of a reward signal in the form of neurotransmitter density in response to desired output behavior. By enabling the rapid prototyping of methods for both stimulation and measurement, we allow experimentation to progress without being hindered by the technical details of implementation.

On top of the network infrastructure and underlying the network dynamics are the substrate models, each admitting its own set of dynamics. Similar to the network, the goal is to provide a language that is capable of abstracting away the implementation specific details in order to provide a common ground with which to describe and compare different models. In the most general sense, a model of the neural substrate should support two forms of update dynamics depending on the application. For the most part, we observe the classic data-driven scheme where state update progresses with respect to previous state and any external information such as applied current. Here, we may describe the state evolution as following a set of ordinary or partial differential equations. To a great extent, providing a common language to describe this form of neural model dynamics is the thrust of the NeuroML project, which maps general, stand-alone model descriptions to a number of existing simulation languages [17]. This is accomplished by expressing a model in terms of its input-output dynamics along with additional model properties such as cell morphology and synaptic transmission. In the
case of biologically detailed and multi-compartment models, the language also supports the description of channel properties and cable equations. The other form of update dynamics that we see in substrate models, particularly synapse models, is according to an event-driven scheme where we see a series of discrete, discontinuous jumps. Most notably, these dynamics capture the behavior of spiking events. The benefit of this approach in modeling network dynamics is that spiking activity forms a large foundation of how information is processed, both in terms of providing the information bearing signal and in terms of reshaping synaptic weights.

3.2 Problem Representation

How we formulate and define a problem is critical to our ability to manipulate the relevant ideas in thinking about that problem. Here, our goal is to extract and translate the important features of the physical system into a mathematical model. As discussed in section 3.1.3, language describing a spiking neural network should encompass both the network topology and the dynamics of the neural substrate. This section introduces the language used to represent the problem domain. As a practical matter, we explore how the problem representation may be realized computationally in addition to simply providing the mathematical formalism.

3.2.1 Network Topology

For a spiking neural network, a natural representation of the connectivity and overall network structure is in the language of graph theory [52]. Precisely, we may formulate the network as a directed graph $G$ where the vertex set $V(G) = \{v_1, v_2, \ldots, v_n\}$ corresponds to the spiking neurons, the edge set $E(G) = \{e_1, e_2, \ldots, e_m\}$ corresponds to the synaptic connections. The direction of an edge $e_l : v_i \rightarrow v_j$, for $l \in 1, \ldots, m$ and $i, j \in 1, \ldots, n$, corresponds to the propagation of a spike where $v_i$ is the pre-synaptic neuron and $v_j$ is the post-synaptic neuron.

Each parallel object then receives a partition of the network containing a set of vertices and the associated edges such that $|V_1| + |V_2| + \ldots|V_k| = |V|$ form a $k$-way partition on $k$ parallel objects. Because we must accommodate for both the incoming synaptic connections as well as the outgoing connections of a given neuron, it is convenient to distribute the set of edges along with its inversion. That is, for $E(G) : \{e_l : v_i \rightarrow v_j\}$ we have the
inversion \( E'(G) : \{ e'_i : v_i \leftarrow v_j \} \). Effectively, this is equivalent to storing the undirected edges of a vertex along with a flag indicating whether the edge is incoming, outgoing, or both.

The standard method for storing the edges of a graph is through an adjacency matrix \( A \) of size \( n \times n \) where a non-zero entry at \( a_{ij} \) corresponds to the existence of an edge \( e_{ij} : v_i \rightarrow v_j \). For a realistic network of \( 10^5 \) to \( 10^6 \) neurons with \( 10^3 \) to \( 10^4 \) outgoing synaptic connections, the overall connectivity is no greater than roughly 10%. This implies a sparse graph structure where the majority of the entries in the adjacency matrix are zero. Due to its sparsity as well as the parallel nature of the problem, we chose the distributed compressed sparse row (dCSR) format for storage of the network topology. In the non-distributed case, a CSR file simply contains a list of edges for each vertex of the graph. Each line of the file corresponds to a vertex in the graph, and the line lists the vertices, indexed by line number, for which there exists an edge. In the distributed format, the CSR file above is split across several files. An additional distribution file that contains the offsets of the vertex index for each of these files is also included. Because we may have multiple parallel objects associated with each of these dCSR files, we modify the distribution file to provide offsets with respect to the network partitions as opposed to just the individual file offsets.

3.2.2 Spatial Partitioning

Due to the specific nature of the network topology, we may partition it such that we reduce the amount of edges between parallel objects. By doing so, we effectively reduce the amount of messages that must be sent across the communication layer. The motivation behind a spatially dependent partitioning scheme is that, biologically, the connection lengths of any given neuron are limited to within a small neighborhood of the cell body [53]. Thus, if we observe the adjacency matrix for a network, we see that the connectivity is not only sparse, but also that the sparsity is structured in a way that neurons that are closer to each other spatially have a higher percentage of connections that with neurons that are further apart. This particular graph structure is considered small-world, where although the majority of the vertices are not connected to one another, the path length, measured in terms of the number of edges required to transition from one vertex to another, is small [54]. To some extent, this small-world characteristic gives rise to the topological mapping discussed in section 2.3.1. For the purposes of simulation, it leads to optimal communication requirements [55].
To give an illustration of the degree to which we reduce the communication costs by using a spatial partitioning scheme, we compare it with a modulo partitioning scheme in the context of a toy network (fig. 3.2). Consider a graph containing 144 vertices aligned to a $12 \times 12$ equally spaced grid. Vertices that are a Manhattan-distance of at most 2 away from any given vertex admit an edge between them, whereas vertices greater than distance 2 do not. This gives 1492 connections or roughly 7.2% connectivity in the adjacency matrix. If we partition this graph into 9 partitions of equal size (16 vertices per partition), even a naive spatial partitioning scheme that gives the naive $3 \times 3$ coarsening on top of the grid will produce good results. The number connections that must be communicated across the partitions, alternatively, the number of edges cut by the partition, under this scheme is 448, or 32.7% of the total connections. A modulo partitioning scheme assigns each vertex to a partition by taking the modulo of the vertex index with the total number of partitions. This scheme is useful for randomly generated networks where there is no spatial dependence and the goal is to simply minimize congestion across any given communication link by exploiting the randomness. However, for our structured toy network, where the vertices are indexed by a standard row-major ordering, this scheme gives an edge cut of 1316, or 88.2% of the total connections.

Figure 3.2: Naive spatial partitioning of toy network: vertices admit edges to both local (solid arrows) and remote (dashed arrows) partitions.
Due to its relevance to many problem domains in scientific computing, methods for efficient and high-quality graph partitioning have been extensively studied. Current state-of-the-art methods involve a multi-level approach whereby a graph is successively approximated until it reaches a manageable size such that an optimal partition may be found combinatorically. This initial partition is then propagated and refined back up the approximations to obtain a partitioning of the original graph. We bootstrap this partitioning process using information about the spatial distribution of vertices found from the neural network. For the ParMETIS algorithm, a $k$-way partitioning to $k$ parallel objects is performed according to the following phases [56]:

1. **Graph Coloring:** A parallel implementation of Luby’s coloring algorithm is first used to determine the structure and sequence of the computation. This is particularly relevant to the coarsening and uncoarsening phases which are performed iteratively according to the colors, and the different colors determine the set of vertices to be used in matching or to be projected, respectively.

2. **Coarsening Phase:** The graph $G_0$ is transformed into a sequence of smaller graphs $G_1, G_2, \ldots, G_m$ such that the number of vertices are progressively reduced $|V_0| > |V_1| > \cdots > |V_m|$. Heuristics are used to group vertices together for each subsequent approximation to the graph. ParMETIS uses a heavy-edge-matching heuristic where it orders the edges by weight and groups a vertex with the one sharing the heaviest edge. In the case of a tie, a vertex is randomly selected.

3. **Partitioning Phase:** A $k$-way partition $P_m$ of the graph $G_m = (V_m, E_m)$ is computed that partitions $V_m$ into $k$ parts, each containing roughly a fraction $V_m/k$ of the vertices of $G_m$. The algorithm used by ParMETIS is a multilevel recursive bisection based off of the Kernighan-Lin (KL) heuristic.

4. **Uncoarsening Phase:** The partition $P_m$ of $G_m$ is projected back to $G_0$ by going through intermediate partitions $P_{m-1}, P_{m-2}, \ldots, P_0$. At each stage of the projection, refinements to the partition are performed to include the reintroduced vertices while satisfying balance constraints. A variation of the Kernighan-Lin (KL) algorithm provides this (greedy) refinement.
3.2.3 Substrate Models

The representation of the substrate models making up the spiking neural network should first and foremost be able to capture both the data-driven and event-driven state dynamics of the physical process. A natural mathematical formulation of the dynamics may be found in stochastic differential equations [57]. These are equations similar to the models of synaptic transmission discussed in sec. 2.1.2 and generalized in the form of eq. 3.4.

\[ dx = f(x)dt + \sum_{i=1}^{n} g_i(x)dN_i \]  

(3.4)

The state, \( x \), evolves continuously in time according to a drift process, \( f(\cdot) \), as well as stochastically according to diffusion processes, \( g_i(\cdot) \), indexed by the event type. Although the diffusion is commonly modeled by a Wiener process, giving rise to Brownian motion, we model the stochastic behavior as a counting process, \( N(t) \), that experiences jumps, \( dN \), corresponding to event times.

On top of this formulation, the goal is to provide a generic interface across the different model types such that they may be easily interchangeable with respect to the simulation tool regardless of the underlying implementation. Computationally, this requires the specification of abstract classes containing virtual methods that are subsequently overridden by the particular model definition. At a minimum we must provide a method to step forward in time with respect to the drift process and a method to handle the discrete events of the diffusion process. Because there may exist multiple synapse types that are incident on a given neuron, we require flexibility in the method functionality in addition to the model class. With respect to the synaptic connections, we must also accommodate for the fact that the synapse type is a property of the presynaptic-neuron whereas the computation is performed at the post-synaptic neuron. This is accomplished by treating network state in the most general sense, as mutable data, where the methods for modifying the data are parameterized solely by the type of computation to be performed and a reference to the data to compute over. Because the computation requirements of the neuron and synapse models fall between the drift and diffusion processes rather cleanly, we provide separate abstract classes tailored for each type of process, respectively. For modeling drift, we provide a time step for the dynamics to evolve over, \( dt \), and for modeling the diffusion, we provide a timestamp of when the event occurred, \( dN \).
3.3 Simulation Engine

Culminating from the design philosophies laid out in section 3.1 and the chosen problem representation in 3.2 is the simulation engine that powers the analysis framework. This is what provides the core functionality of STACS, enabling an accurate and efficient simulation of large spiking neural networks. Fundamentally, the simulation engine is comprised of only a few key elements: the parallel infrastructure that embeds the network topology, the time-driven computation that evolves the network state, and the event-based protocol that determines the network communication both within the network and external to the system via asynchronous data streams. These elements are elaborated below in turn.

3.3.1 Parallel Programming Paradigm

The parallelization of the network by partitioning to a collection of parallel objects is realized by the Charm++ parallel programming system [58]. Charm++ establishes a programming paradigm of over-decomposition of an application into logical work and data units called *chares*. Scheduling and execution is managed through an adaptive runtime system based on message-driven, migratable objects. That is, the parallel objects defined by the application may move across processors during the lifetime of the execution, enabling dynamic load balancing for more efficient utilization of resources. By allowing several parallel objects to share the same processor, Charm++ is able to mask network latency by encouraging the overlap of computation with communication.

The runtime system also manages the communication between parallel objects, which is asynchronous in nature. In particular, Charm++ employs a communication model through remote method invocations where computation occurs only when the required data dependencies are received. Using the *CkMulticast* libraries, Charm++ is capable of generating optimal spanning trees that facilitate methods for broadcasting and reducing data to and from subsets of chares. Moreover, the modular structure encourages the development of collective operations, such as network metrics or stimulation, on top of the parallel application.
3.3.2 Time-driven Computation

With respect to the computation of network dynamics, we adopt a time-driven approach borrowing elements from NEST [48]. As described in 3.2.3 we model the neural substrate generally as stochastic differential equations. We treat the occurrence of spike events as a counting process with a delay time equivalent to the axonal delay between the pre-synaptic neuron and post-synaptic neuron. Because the minimum axonal delay of the network as a whole corresponds to the minimum amount of time before a spike generated by one neuron may affect any other neuron, we are able to decouple the computation of the network accordingly.

Specifically, we are able to evolve the drift process of the neuron state independently of any communication within this minimum delay time. This enables the use of model specific integration schemes that allow for increased flexibility or precision of computation as necessary. In this way, the simulation of the network is able to evolve along a coarse time grid while still retaining precision of event times. With respect to performance, this also has an advantage over a global time step method where increased precision of events comes at the cost of increased computation required of the entire network. The communication overhead is also reduced as the number of synchronization points is minimized.

Whereas the neuron dynamics evolve alongside a potentially variable time step, the synaptic dynamics are computed at discrete times coinciding with the event timestamp. By evaluating these dynamics only at discrete times as opposed to integrating a large coupled system of differential equations, we lower the number of floating-point operations per second required per unit of simulated time by several orders of magnitude. We reduce the amount of memory loads and stores by a similar amount. This is accomplished by using the online methods described in section 2.1.2.

3.3.3 Event-based Communication

Much of the computation that occurs in a spiking neural network rests on top of events, namely the spiking of a neuron. Although these events happen relatively infrequently when compared to the ‘resting’ state of a neuron, their precise timings are important to the network level behaviors such as the polychronous activity discussed in section 2.3.3. Timing is also important at the level of the neural substrate, for example, in the synapse where the timing between the pre-synaptic and post-synaptic spikes determines
changes in strengthening or weakening.

The natural representation for handling this type of dynamic is through the use of event lists. As new events arrive on the network partition, they are placed on the list associated with its network element in order of their timestamps. However, implementing a single event list for a given element such as a neuron is impractical as the total number of events will be greater than the number to be processed in any given iteration. This is because axonal delays greater than the minimum will effectively place events at future iterations, imposing unnecessary computation with respect to sorting. To accommodate this, we employ the use of calendar queues such that each ‘day’ is chosen to be the length of an iteration time interval and the number of days in a ‘year’ is chosen such that we may account for the maximum axonal delay without overlapping [59]. Any event delays that are greater than this are placed in an additional buffer to be distributed at the beginning of each new year.

To reduce the amount of global communication that occurs as the parallel system scales, we employ a neighbor-only communication method whereby network partitions only communicate event data to partitions for which there exists an edge between them. This local exchange of data falls in line with the spatial partitioning described in section 3.2.2. Effectively, this method exchanges only as many messages in a given iteration as is necessary. By using the broadcast libraries provided by the Charm++ runtime system, the communication overhead is further reduced.

3.3.4 Asynchronous Streams

As an extension to the event-based communication protocols, the simulation tool also admits asynchronous data streams external to the spiking neural network. Unlike the network events where we must wait on neighboring partitions, however, there is no explicit data dependency on external information. In other words, the timing between the network and the environment is loosely coupled. Although the entry method invocations for Charm++ provide a good platform for asynchronous communication within the application, there is no way to natively invoke methods from outside of the application. In order to transmit external events to the simulation tool as the network evolves, we provide a separate communication protocol that is capable of inter-process communication.

The YARP project, Yet Another Robot Platform, provides such a communication protocol [60]. The target application of YARP is in the infrastruc-
ture development for humanoid robotics where hardware operations include but are not limited to: audio, video, and tactile sensory, fine motor control, and the data processing in between. In line with the goal of STACS in studying a spiking neural network in a closed-loop environment, the communication model supported by YARP is designed and built for closed-loop systems composed of many processes on many processors. In terms of stimulating the neural network, we use the YARP libraries to provide callback functions that may be triggered upon receiving data through remote procedure calls. These callback functions are responsible for any transcription of the message into network events. In terms of recording from or driving control using the neural network, we simply reverse the process by defining where and how the network should send data generated through computation.

3.4 Neural Network Simulation

To illustrate the capabilities and practical application of STACS, we provide a simulation of a neural network with representative biological scale and topological specification. For completeness, we step through the process of network construction prior to the simulation proper. We also examine how stimulation may be applied asynchronously to the network as it is running. Throughout, we compare and contrast the simulated network to its biological counterpart, such as what might be found on an MEA, and we highlight the advantages of using STACS in studying network behavior.

3.4.1 Network Construction

To emulate the scale of neural networks that are grown on MEAs, we construct a network consisting of 10000 neurons distributed randomly over a circle of radius 800µm, or area approximately 2mm². Connectivity is spatially dependent, and the connection probability between any two neurons as a function of their separation distance is modeled according to [53]. A plot of the distribution used to determine the connections in our network is shown in fig. 3.3. This gives approximately 1000 synaptic connections per neuron for an overall network connectivity of 10%.

With respect to the neural substrate, we employ biologically faithful models. For the neuron, we use the phenomenological model developed by Izhikevich (sec. 2.1.1). At the synapse, synaptic transmission is handled using the
Figure 3.3: Connection probability between two neurons as a function of their separation distance.

typical conductance-based model, and synaptic plasticity admits voltage dependencies in addition to the standard spike-timing-dependent component. We implement online, event-based methods for both these models (sec. 2.1.2). Axonal delay between neurons is distributed between 1ms and 20ms, and like the spatially dependent connectivity, is also determined as a function of separation distance.

3.4.2 Network Simulation

Network simulation is performed on a compute server consisting of two Dell PowerEdge R410 systems operating under Debian 6 (squeeze) and connected through a gigabit ethernet switch. Each compute node is equipped with a quad-core 2.4 GHz Intel Xeon E5530 processor and 16 GB RAM.

For the simulation, we generate a 16-way partition of the network (fig. 3.4) into parallel objects according to the spatially dependent scheme proposed in sec. 3.2.2. In contrast to a naive approach, we see that although each partition preserves the spatial locality to some degree, there is also significant spatial overlap as a result of the refinements that occur in the uncoarsening phase. This produces an edge cut of approximately 32.6% of the total number of connections.

To measure the performance of STACS under realistic spiking activity, we drive the neurons such that the network produces asynchronous firing of an average spiking at a baseline rate of 10Hz. The network is simulated for 10 seconds of biological time and the wall-clock time is recorded and averaged
Figure 3.4: Spatial partitioning: network partitions overlap (top) while preserving spatial locality (bottom).
over 5 simulation runs. Simulation was performed on 4 and 8 CPU cores, resulting in 4 and 2 parallel objects per core, respectively. This yielded wall-clock times of 354.2 and 195.4 seconds, respectively. In terms of raw performance, this is significantly better than the serial counterpart (1338 seconds) by a speedup of roughly $3.78 \times$ and $6.85 \times$, respectively.

With respect to scalability, although there is a definite speedup ($1.81 \times$) in moving from 4 to 8 CPU cores, attributed to the division of computation over more cores, we see that this speedup is not quite linear. There are two main factors that contribute to this slight loss of performance. First, by moving from a single node to multiple nodes, we incur overhead through the addition of inter-node communication. Second, by admitting fewer parallel objects per CPU core, we also observe a decrease of the overlap in computation-communication.

To illustrate the above, as well as provide a breakdown of the parallel work (eq. 3.2), we perform a trace of the network simulation using the Projections performance analysis and visualization tool [61]. Snapshots for 4 and 8 CPU cores are given in fig. 3.5.

![Simulation Trace](image)

Figure 3.5: Simulation trace: parallel work is divided among computation (blue), communication (orange), and idle (black); lines (yellow) indicate sample communication pathways.

We immediately see that the over-decomposition on 4 CPU cores enables considerable overlap between the computation and communication of the simulation. Furthermore, there is relatively little idle time, limited mostly by subtle load imbalances. Unfortunately, these inconsistencies become more
apparent on 8 CPU cores, and we find that computation tends to progress slower than, as opposed to as fast as, the slowest processor. While this may be alleviated through repartitioning the network to more parallel objects, it was found empirically that the trade-off of a larger edge cut as a result of repartitioning was more detrimental to performance in this case.

3.4.3 Network Analysis

A standard method to analyze the output of a spiking neural network is to record the spike times in a spike raster, indexed by neuron. A sample spike raster for one second of our asynchronously firing network is shown in fig. 3.6. Here, the horizontal banding is an artifact of the spatial partitioning, as each parallel object contains both excitatory and inhibitory type neurons.

![Network Spike Raster](image)

Figure 3.6: Network spike raster for one second of biological time of asynchronous firing at 10Hz.

In comparison to recording from an MEA, the increase in resolution both spatially and temporally is considerable. On an MEA, we may have a grid of 64 20\(\mu\text{m} \times 20\mu\text{m}\) regions separated by 100\(\mu\text{m}\), corresponding to the physical placement of the electrodes. Temporally, although the sampling rate of an MEA is in the kHz range, in order to prevent data glut, the recording software typically bins detected spike times according to the total acquisition time of an experiment. For an experiment lasting on the order of minutes, a sane amount of bins comes out to be on the per millisecond interval at best. In STACS, not only do we have the precise Cartesian coordinates of
the neural cell body, but because we record spike times on an individual neuron basis, we also allow for sub-millisecond resolution in spike timing without suffering from data glut. Additionally, we may access any portion of the network arbitrarily.

3.4.4 Network Stimulation

Currently, STACS allows stimulation to the network in the form of applied current pulses. These may be applied to the entire network simultaneously, or, more realistically, to a localized region of the network. In particular, the user is capable of specifying a spherical volume by providing a center spatial coordinate along with a radius. The goal is to mimic how stimulation is typically applied to an electrode on an MEA. Stimulation to the simulated network is much more versatile, however, as we are not limited to specific locations in the network. Furthermore, each individual current pulse is defined by its amplitude, onset time offset, and duration, and may be combined to provide arbitrary signals. That is, we may provide arbitrary stimulation to arbitrary regions of the network.

To illustrate, we provide a simple square wave of duration 25ms with a period of 5ms (5 pulses) to a region first centered around the North-Eastern edge and then to the South-Western edge of our network (fig. 3.7). The square wave has a duty cycle of 40% and an amplitude of 15µA/cm², and the centers of the stimuli are located at (500µm, 500µm) and (−500µm, −500µm) with a radius of 200µm.

![Figure 3.7: Stimulation: target regions and affected neurons (left); applied square wave current pulse (right).](image-url)
To study the effects of stimulation on the network, we observe the spike raster (fig. 3.8). From this, we immediately see an increase in spiking density in the regions of the network where the stimuli were applied. Additionally, we may see subtle changes in the network activity from asynchronous firing to having slightly more structure in the spiking density over time. In particular, there is a lull in spiking density immediately after the removal of stimuli, followed by slow oscillations that propagate through the network as a result of both spatial connectivity and axonal delays. In contrast to experimentation on an MEA, where electrodes may be used to stimulate or record but not both, the ability to simultaneously stimulate and record from the same region of the network is critical to detecting and analyzing these changes in the network activity. Ultimately, it is from the analysis of the network in response to stimuli at this level where we expect to gain the most in terms of furthering our understanding of brain function.

Figure 3.8: Network spike raster for one second of biological time with multiple stimulation.
We have presented motivation and supporting background information relevant to the formation of an analysis framework for the study of biophysically plausible networks of spiking neurons. From this, we undertook the problem of representation both mathematically and computationally such that we could construct a simulation tool, STACS, that was both a practical and faithful realization of this framework. Several key considerations along the way were also discussed and an application highlighting the capabilities of the tool was examined.

4.1 Discussion

Starting at the neural level, we hope to provide a path toward understanding the adaptive learning mechanism that is exhibited at the network level. Through bridging this gap in our understanding, the goal is to develop mathematical models of the network analogous to the phenomenological models of the substrate. Toward this end, the development of a simulation tool that is capable of providing a closed-loop system to a spiking neural network opens up a wide range of experimentation as well as facilitates more comprehensive analysis.

STACS is such a simulation tool. In addition to providing a suitable parallel infrastructure, most importantly, it offers a platform by which external data streams may be delivered asynchronously to the network. Although an example was given with respect to stimulation using current pulses (sec. 3.4.4), the range of potential streamed data types is certainly not restricted to this. Most notably, we may now observe how the adaptive application of a reward signal in modulating the synaptic plasticity affects the learning process. The ability to perform motor control further facilitates this study.

STACS also provides other features that facilitate experimentation and analysis. The ability to save network state, for example, enables the study of how learning may diverge through the application of different inputs.
Starting from identical initial state, random perturbations to the network may also be used to determine robust network phenomena during learning. As a result of the general representation of the neural substrate, we may also compare different substrate models according to their effects on metrics at the network level. Moreover, these metrics themselves may be compared for their effectiveness in modeling the network phenomena.

4.2 Future Work

Currently, there is significant room for expansion with respect to the implementation of additional modules that would lie on top of the core functionality. These include a wider variety of neuron and synapse models forming the neural substrate, methods for transcribing audio and video input into stimulation events onto the network, and the development of a number of metrics that operate at the network level such as detection of polychronous groups. Support for birth and death processes with respect to the synaptic connections would also be beneficial for studies at longer time scales. Effectively, the goal is to expand the range of simulation tools both in terms of experimentation and in terms of analysis.

A general exploration of the extent to which simulation tools may accurately capture network behavior when compared to the biological system is also pertinent. Here, the use of cortical cell cultures on devices such as MEAs would provide the appropriate baseline for comparison. The goal of such a study would be to compare how close the two systems are to each other based on network metrics, and be able to modify the gross network properties of the simulated approach so that it more closely resembled the physical system. In this way, we may gauge the validity of the simulated network. Through this process, we may also hope to highlight the extent to which neural simulation in its current form may address questions of learning. By brushing up against the limitations of the current toolset, such as the lack of structural plasticity, the goal would be to provide future direction on where to focus development.
REFERENCES


