A Neuroscience-Inspired Approach to Training Spiking Neural Networks

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

James Michael Ghawaly Jr.

May 2020

© by James Michael Ghawaly Jr., 2020
All Rights Reserved.

This thesis is dedicated to my wife Rebekah.

Acknowledgments

I would like to thank my wife Rebekah and also my family for their support and patience throughout the long hours of working on this thesis. I would especially like to thank my advisor Dr. Sadovnik and my committee members for their guidance and mentoring throughout this process.

Abstract

Spiking neural networks (SNNs) have recently gained a lot of attention for use in low-power neuromorphic and edge computing. On their own, SNNs are difficult to train, owing to their lack of a differentiable activation function and their inherent tendency towards chaotic behavior. This work takes a strictly neuroscience-inspired approach to designing and training SNNs. We demonstrate that the use of neuromodulated synaptic time dependent plasticity (STDP) can be used to create a variety of different learning paradigms including unsupervised learning, semi-supervised learning, and reinforcement learning. In order to tackle the highly dynamic and potentially chaotic spiking behavior of SNNs both during training and testing, we discuss a variety of neuroscience-inspired homeostatic mechanisms for keeping the network's activity in a healthy range. All of these concepts are brought together in the development of a SNN model that is trained and tested on the MNIST handwritten digits dataset, achieving an accuracy of 62%. In order to achieve this, we also introduce a custom Python library called Ganglion that can be used to rapidly design and test SNN architectures.

Table of Contents

1 Introduction **1**

1.1 Motivations and Contributions 1

1.2 Biological Neurons, Neural Networks, and Learning Principles 3

1.2.1 Neurons 3

1.2.2 Synaptic Plasticity 8

1.2.3 Homeostasis 10

1.2.4 Neuromodulated STDP 16

1.2.5 Network Structure 21

1.3 Previous Work in Biologically Inspired Spiking Neural Networks 26

1.3.1 The Diehl-Cook Model 26

1.3.2 Convolutional Spiking Neural Networks 28

2 Biologically Inspired Training of Artificial SNNs **30**

2.1 Neurons 31

2.1.1 Synaptic Input Current 33

2.2 STDP 33

2.2.1 Neuromodulated STDP 37

2.3 Homeostasis 39

2.3.1 Intrinsic Plasticity 40

2.3.2 Synaptic Scaling 42

2.4 Network Decision Making 43

2.4.1 Output Encoding 44

2.4.2 Supervised Intrinsic Short-term Plasticity 45

3 Oriented Bar and MNIST Digit Classifcation **48**

3.1 Example: Oriented Bar Classification 48

3.1.1 Training the Feature Detector 49

3.1.2 Training the Classifier 51

3.2 Example: MNIST Handwritten Digit Classification 56

3.2.1 Network Design 57

3.2.2 Training 61

3.2.3 Results 62

3.2.4 Discussion 63

4 Conclusion **70**

4.1 Discussion 70

4.2 Future Work 72

Bibliography **74**

Appendices **83**

A The Ganglion Spiking Neural Network Simulation Package 84

A.1 Motivation 84

A.2 Solving Differential Equations 85

A.3 Neurons 86

A.4 Synapses 93

A.5 Neural Networks 94

A.6 Network Visualization 96

Vita **97**

List of Tables

1.1 Effects of various neuromodulators on LTP and LTD synaptic eligibility traces as observed in pyramidal cells of the layers II/III of the visual and prefrontal cortices [30]. LTP eligibility traces were created by inducing a pre-before-post firing pattern and LTD eligibility traces were created by inducing a post-before-pre firing pattern [30]. As discussed in Section 1.2.2, pre-post induces LTP and post-pre induces LTD in the standard STDP protocol. 19

3.1 Parameters used for the feature learning stage of training the bars network. Non-default parameter values are shown in bold. 51

3.2 Parameters used for training the neuromodulated STDP synapses for the classification learning stage of training the bars network. Non-default parameter values are shown in bold. Unless otherwise stated in this table, non-listed parameters use the values in Table 3.1 53

3.3 Relevant parameters used for the feature learning stage of training the bars network. Parameters that are not actively used in the simulation are omitted from this table. 68

3.4 Relevant parameters used for the decision layer training stage for the MNIST network. Parameters that are not actively used in the simulation are omitted from this table. 69

List of Figures

1.1 Basic diagram of a single neuron (top) and a small neural circuit (bottom). Figure created with BioRender.com 5

1.2 Basic diagram outlining the electrochemical processes generating the charge imbalance between the neuron and its environment. Figure created with BioRender.com 6

1.3 Synapses for excitatory (left) and inhibitory (right) presynaptic neurons. Figure created with BioRender.com 7

1.4 The basic steps by which pre-before-post synaptic spiking leads to LTP. Figure created with BioRender.com 11

1.5 Shape of the STDP protocol for rat hippocampal neurons as determined by Bi and Poo, with exponential fit [7]. 12

1.6 Approximation of the shape of a typical neuron's F-I/gain curve modelled as an XX1 function convolved with Gaussian noise ($\sigma = 20$) [56] 13

1.7 M-current K^+ ion conductance as a function of membrane potential with and without added intracellular Ca^{2+} ions. Data extracted from figure in [72]. . . 14

1.8 A simplified diagram of visual processing in the visual cortex. Taking a bottom-up approach, the learned visual features progress from elementary edges and corners in earlier layers to complex shapes and object recognition in the final layers. 22

1.9 Dense, convolutional, and local receptive field types demonstrated with a numeric 8 input space and a 4x4 matrix of neurons. The blue shaded regions indicate the receptive fields. 24

1.10 An example of a group of four neurons sharing the same receptive field. Without competitive learning, each of these neurons would likely only learn the most significant feature in the input space, which in this case is the thick line near the top of the receptive fields. 24
1.11 Example of a neuron (center square) projecting lateral inhibitory (red) connection to non-neighboring neuron and local excitatory connections (blue) to neighboring neurons. 25
1.12 Diagram of the Diehl-Cook SNN model [16]. Blue and red represent excitatory and inhibitory synapses, respectively. 27
1.13 Receptive fields of the 100-neuron Diehl-Cook SNN model after training, directly from [16]. 27
1.14 Generalized hierarchy of biological realism in popular neural network models ranging from most (top) to least biologically realistic (bottom). 29

2.1 Response of a single LIF neuron to an input current that starts at a constant value of 1nA and turns off at 50ms. $V_r = -70.6$mV, $V_{\text{thr}} = -50.4$mV, $\tau_m = 9.37$ms, $C_m = 281$pF, and $g_e = 100$nS 32
2.2 Time profile of the alpha synapse conductance model. $\bar{g}_d = 100$nsiem, $\tau_{syn} = 5$msec, $w_d = 1$ 34
2.3 STDP function used by Gerstner et. al. as defined in Equation 2.5 [23]. . . . 35
2.4 Equation 2.16 with p_0, p_1, and p_2 equal to 8.443, 0.273, and -36.072 respectively, plotted against experimental data shown in Figure 1.7 (experimental data from [72]). 41
2.5 The F-I gain curve for the Adaptive M-current Leaky Integrate and Fire (AMLIF) model neuron. 43
2.6 Average firing rate of all 16 output neurons estimated using an exponential moving average over a duration of 10 seconds. The network was shown a single image of a vertical bar on an 8x8 pixel grid. The firing rate is clearly not constant, demonstrating the probabilistic nature of spiking neuron firing activity. 46

3.1 The images in the bars dataset. The network must learn to classify each image as either a horizontal or vertical bar. 49
3.2 Neural network architecture for learning features of the bars dataset. Blue and red arrows represent excitatory inhibitory connections, respectively. The blue squares represent the receptive field of each neuron in the postsynaptic connection. 50
3.3 Weights learned by the feature detector layer when trained on the bars dataset using standard STDP 52
3.4 First iteration of the neural network architecture for bars dataset classification using pre-trained feature detector weights. Blue and red arrows represent excitatory inhibitory connections, respectively. The blue squares represent the receptive field of each neuron in the postsynaptic connection. 53
3.5 Accuracy of the network shown in Figure 3.4 when trained on the bars dataset using the parameters shown in Table 3.2. The accuracy is calculated using a moving average with a window size of 16. 54
3.6 Accuracy of the network shown in Figure 3.4 when trained using different learning rates (η). Higher learning rates yield instability in the training. The accuracy is calculated using a moving average with a window size of 16. . . . 55
3.7 The same network as shown in Figure 3.4, with an additional layer between the feature and decision layer. Blue and red arrows represent excitatory and inhibitory connections, respectively. The blue squares represent the receptive field of each neuron in the postsynaptic connection. 55
3.8 An example selection of handwritten digits from the MNIST dataset. 56
3.9 An example digit from the MNIST dataset before pre-processing (left) and after pre-processing using cropping, normalization, DOG filtering, and deskewing (right). 58
3.10 The architecture used for the MNIST classification problem. There are 30 locally-connected neuron groups. Blue connections indicate excitatory connections and red indicate inhibitory connections. 58

3.11 Lateral inhibition via 1 to 1 connectivity. Blue arrows indicate inhibitory connections. 60
3.12 Weights of the receptive fields learned by the feature detection layer of the MNIST classification network. The features are composed of various edges at different scales and orientations. 64
3.13 (top) Weights of the receptive fields learned by the feature detection layer of the MNIST classification network, zoomed in to just four neuron groups from Figure 3.12. (bottom) The weights of the same 4 groups as for the top figure but without using lateral inhibition during training. 65
3.14 Confusion matrix for the algorithm evaluated over the entire testing dataset. 66

1 F-I curves for the leaky integrate and fire neuron with Euler time step sizes of 0.01 ms (left) and 0.001 ms (right), respectively. 86

Chapter 1

Introduction

1.1 Motivations and Contributions

This thesis is primarily concerned with the development of a general method for training artificial spiking neural networks (SNN), strictly using methods inspired by biological neural networks, especially those of the human brain. Other previous works, which are further discussed in this chapter, generally blend traditional artificial neural network (non-spiking) methods with some biologically-inspired techniques and demonstrate various levels of success. This thesis, however, approaches the task of training SNNs strictly using network architecture designs, learning algorithms, and network simulation principles inspired by biology without the use of any traditional machine learning methodologies. In this way, this method has the advantage that living neural systems, especially those in the mammalian brain, are overwhelmingly successful in nature at innumerable simultaneous tasks. On the other hand, this approach is challenging due to the fact that the processes underlying intelligence and cognition in living neural systems are not fully understood, leading to the question of whether or not the present knowledge is *enough* for the development of useful artificial intelligence.

Despite the challenges involved with designing fully biologically inspired spiking neural networks, this thesis attempts to use some of the most well-established concepts discovered by cognitive neuroscientists that have shown to be important to cognition. This thesis describes these biological principles in Chapter 1; the concepts introduced extend all the way from the basics of the neuron to the methods in which neural networks learn, including

the important influences of neuromodulation on learning. Chapter 1 also gives a short overview of prior work on the topic of biologically-inspired training of SNNs. Chapter 2 then goes on to discuss how the biological concepts in Chapter 1 can be modelled in artificial systems and demonstrates these concepts on a simple classification task. Chapter 3 then introduces the MNIST digit classification problem and how a network was designed and trained to classify handwritten digits from 0 to 9. Chapter 4 primarily gives an overview on original contributions and potential areas of future work. Finally, Appendix A gives a broad overview of the Ganglion SNN simulation library, along with a tutorial on the usage of this library for spiking neurons and neural networks.

Below is a list of the significant contributions that the work in this thesis made to field of spiking neural networks:

- The design of a new neuron model based on the leaky integrate and fire model that includes the voltage-dependent M-current for spike rate adaptation. The M-current is modeled based on real experimental data and the neuron model demonstrates spike rate adaptation in tune with that observed in real pyramidal neurons, which is important for network activity homeostasis. This model is called the Adaptive M-Current Leaky Integrate and Fire (AMLIF) model.

- Developed a flexible, object-oriented spiking neural network simulation library for Python 3 called Ganglion. The library is designed to be extremely flexible and allows developers to rapidly design and simulate complex spiking neural network architectures using a variety of biologically-inspired methodologies. Included is a graphical visualization environment written in OpenGL for observing spiking networks as they are running.

- Demonstrated the use of synaptic time dependent plasticity for unsupervised training of networks to extract useful features from visual data.

- Demonstrated the use of neuromodulated synaptic time dependent plasticity for converting standard STDP into a variety of alternative forms of Hebbian/STDP learning (including standard Hebbian learning, anti-Hebbian learning, standard STDP,

and anti-STDP), as is observed in biological neural networks. The viability of using neuromodulated STDP combined with standard unsupervised STDP for training spiking neural networks on standard machine learning classification tasks is also demonstrated.

1.2 Biological Neurons, Neural Networks, and Learning Principles

1.2.1 Neurons

The neuron is the fundamental unit of a neural network, consisting of three major parts: the cell body, the dendrites, and the axon as seen in Figure 1.1. There are many neuron types in the mammalian brain, however in this thesis we will focus on a specific type of multipolar neuron called a pyramidal neuron. Pyramidal neurons are the primary neuron type present in many cognitive neural circuits such as the prefrontal cortex and may facilitate and enable the development of higher order cognitive processes [19]. The neuron's cell body contains all of the working parts for keeping the cell alive. The dendrites are the location at which the axon of other neurons will connect to this neuron. This connection is called a synapse. Neurons that send signals into a synapse are referred to as presynaptic neurons and the receiving neuron is referred to as the postsynaptic neuron. These concepts are all illustrated in Figure 1.1. The efficacy in which signals can travel across a particular synapse is called the synaptic weight. These weights can change through a variety of mechanisms that will be later discussed, the process of which is called synaptic plasticity. Unlike "neurons" in traditional artificial neural networks (ANN), biological neurons do not output a constant continuous signal, rather, they communicate through discrete voltage spikes called action potentials.

To understand the process of generating an action potential, one must first understand the electrochemical dynamics present within the neuron. The electrochemical dynamics in the cell can be mostly described by the diffusion and electromagnetic forces between four ions present inside and outside of the neuron's plasma membrane: K^+, Na^+, Cl^-, and Ca^{2+} [56].

In order for a neuron to generate an action potential, it must first create a voltage imbalance between the cell and the surrounding environment. This imbalance is generated by the Na^+/K^+ adenosine triphosphatase (Na^+/K^+-ATPase or sodium-potassium ion pump) that constantly works to pump Na^+ ions out of the neuron and K^+ ions to the inside of the neuron via the hydrolysis of ATP [68]. For every three Na^+ ions pumped out of the neuron, two K^+ ions are pumped into the neuron. Also present are potassium leak channels that allow K^+ to diffuse from the inside to the outside of the neuron. The interaction between the sodium-potassium pump and the potassium leak channels results in a balanced excess of positive charge outside of the neuron, giving the neuron a resting potential of about $-70mV$ with respect to its environment. Aside from the charge imbalance, the sodium-potassium pump also creates a concentration imbalance for each ion leading to a diffusion force that further attempts to push $Na+$ ions into the neuron and K^+ outside of the neuron. This process is illustrated in Figure 1.2.

The membrane potential of a postsynaptic neuron can be increased ("excited") by a presynaptic neuron via release of a neurotransmitter called glutamate into the synapse. Glutamate can be released into the synapse via action potential-induced exocytosis of the presynaptic vesicles [73]. Glutamate binds to an ionotropic receptor called α-amino-3-hydroxy-5-methyl-4-isoxazole propionic acid (AMPA) which allows Na^+ ions to flow into the postsynaptic neuron, resulting in an increase in membrane potential. This flow of Na^+ ions into the postsynaptic neuron is called an excitatory postsynaptic current (EPSC). Likewise, the membrane potential of a postsynaptic neuron can be decreased ("depressed) by a presynaptic neuron via release of a neurotransmitter called gamma-aminobutyric acid (GABA) into the synapse. This neurotransmitter attaches to GABA receptors which allows Cl^- ions to enter the postsynaptic neuron, resulting in a decrease in membrane potential. This flow of Cl^- ions is called an inhibitory postsynaptic potential (IPSP). Both postsynaptic potentials take the form of an alpha function which is discussed in more detail in the modelling section, Section 2.2. This process is illustrated in Figure 1.3. Any excess glutamate or other neurotransmitter is then cleaned up from the synaptic cleft through a process called re-uptake.

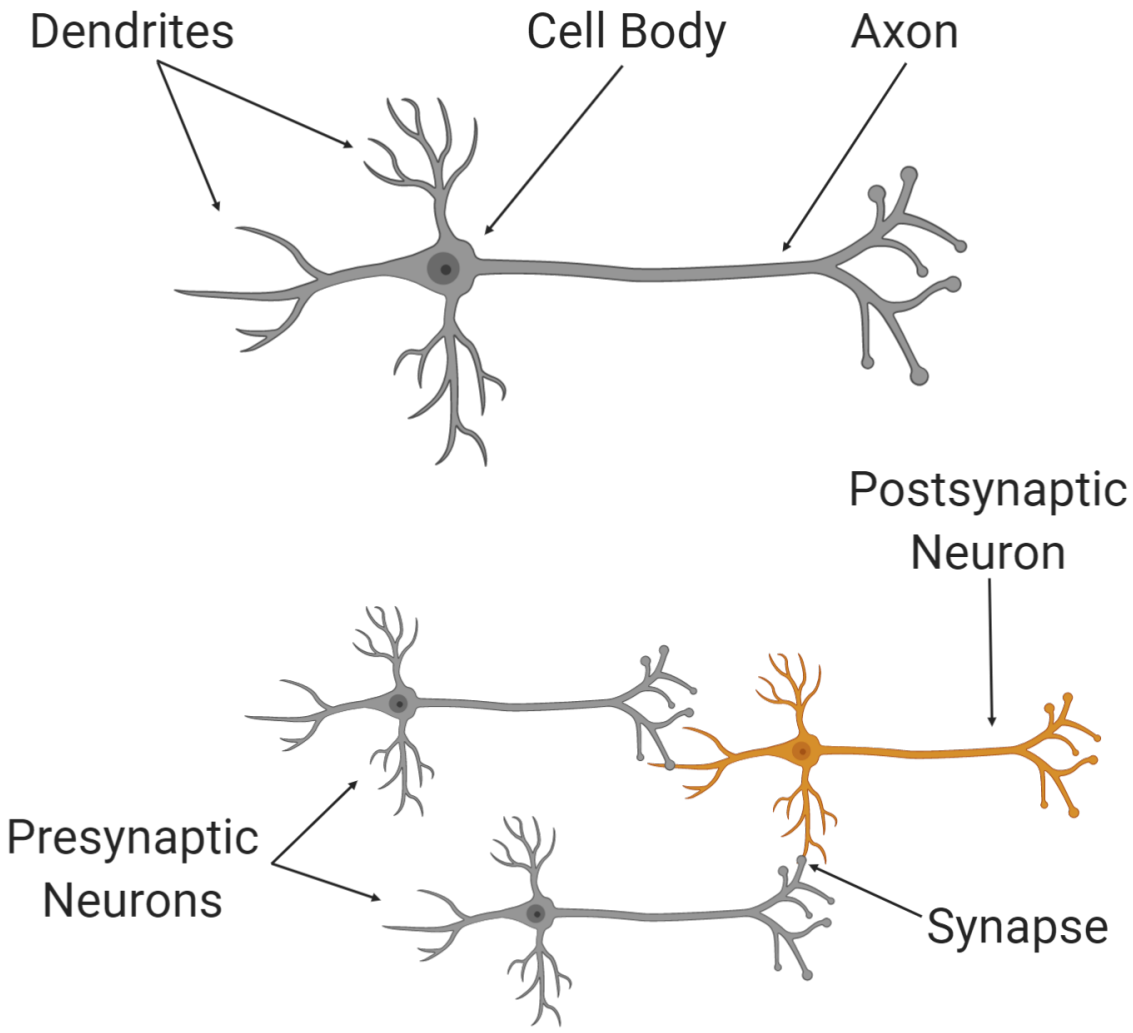


Figure 1.1: Basic diagram of a single neuron (top) and a small neural circuit (bottom). Figure created with BioRender.com

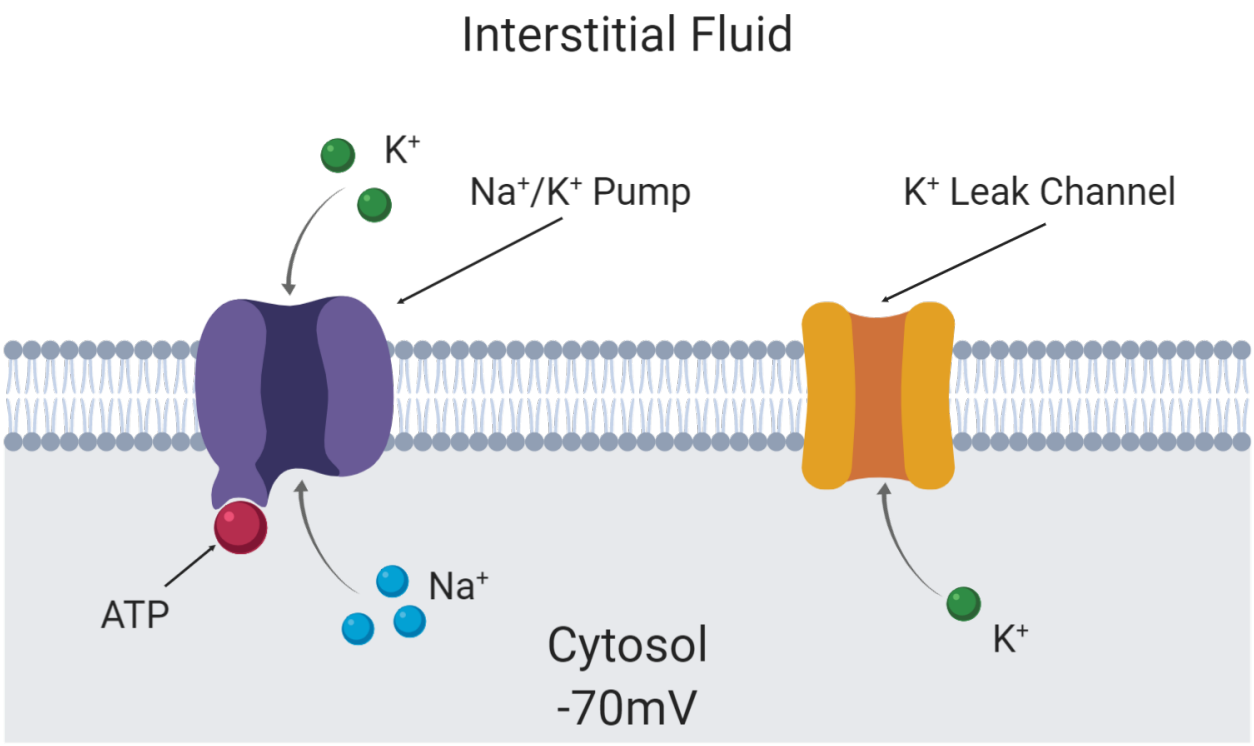
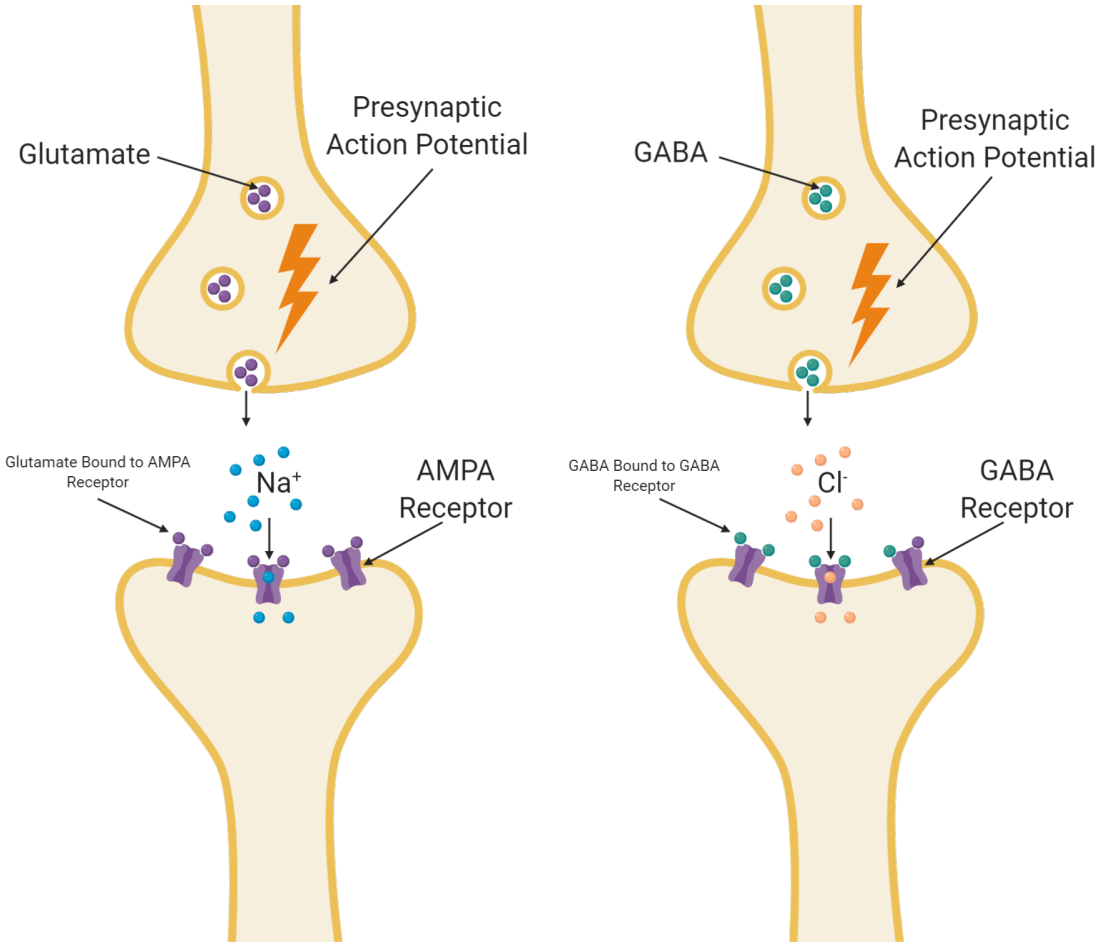


Figure 1.2: Basic diagram outlining the electrochemical processes generating the charge imbalance between the neuron and its environment. Figure created with BioRender.com



(a) Excitatory **(b)** Inhibitory

Figure 1.3: Synapses for excitatory (left) and inhibitory (right) presynaptic neurons. Figure created with BioRender.com

When the membrane potential increases past the saturation threshold (about -55 mV), the depolarization induced by the influx of Na^+ ions leads to opening of further Na^+ ion channels which again leads to further depolarization. This positive feedback loop is called the Hodgkin cycle and results in a rapid cascade of depolarization called the action potential (AP) or "spike" [32]. This depolarization continues until the K^+ channels open and re-polarize the neuron. This repolarization results in an overshoot called hyperpolarization. During the hyperpolarization stage, the Na^+ channels are inactive and and the neuron cannot be activated. This period of time is called the refractory period and generally lasts a few milliseconds.

1.2.2 Synaptic Plasticity

As stated earlier, each synapse has an associated "weight" value that describes how effectively ions can be transferred from the presynaptic neuron to the postsynaptic neuron. For the case of an excitatory presynaptic neuron, the weight of a synapse is largely determined by the amount of glutamate that is released into the synaptic cleft by the presynaptic neuron and the fraction and number of AMPA receptors that are available for binding on the postsynaptic dendrite [56]. Also discussed was the fact that synaptic weights have the ability to be modified through a process called synaptic plasticity. For the process of learning, two forms of long-term synaptic plasticity are considered: a long-term increase in synaptic weight called long term potentiation (LTP) and a long-term decrease in weight called long term depression (LTD). The majority of experimentally-validated forms of synaptic plasticity are forms of Hebbian learning. Donald Hebb introduced Hebbian theory in his book title *The Organization of Behavior* in 1949 [31]. Hebbian theory can be understood via Hebb's postulate: "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased," which is often abbreviated as "neurons that fire together, wire together" [31]. It wasn't until nearly fifty years later in 1997 that researchers discovered experimental evidence for Hebbian processes in actual neural tissue [50]. The researchers demonstrated that the amplitude of excitatory postsynaptic potentials (EPSP) across a given synapse would increase or

decrease proportional to the precise timing between presynaptic and postsynaptic APs. The amplitude of synaptic EPSPs demonstrated LTP following repeated trials of presynaptic APs preceding postsynaptic APs and experienced LTD otherwise. Further, no long-term changes in synaptic efficacy were observed when presynaptic and postsynaptic APs were separated by an amount of time greater than 100ms. Due to the time-dependence of the synaptic efficacy, this process is called synaptic time dependent plasticity (STDP).

The importance of the discovery of STDP for understanding the computational properties of biological neurons cannot be emphasized enough. At a general level, the bidirectional behavior of STDP shows that neurons learn primarily through the principle of causality. A neuron becomes sensitized to other neurons that convey statistically-significant information and desensitized to others. This, along with the high-pass filtering properties of neuron's with spike frequency adaptation (see Section 1.2.3) [4], allows a neuron to become a high-frequency pattern feature detector.

The electrochemical processes underlying STDP are very complicated. The N-methyl-D-aspartate (NMDA) receptor plays a very significant role in the STDP process [50]. The NMDA receptor is a voltage-dependent ion-channel with binding sites for both glutamate and glycine [22]. When these neurotransmitters bind to the NMDA receptor they allow cations to flow, especially Ca^{2+}. However, the flow of Ca^{2+} is voltage-dependent. Unless the neuron is depolarized through excitatory AMPA channels as previously discussed, the passage of Ca^{2+} through the NMDA receptor is blocked due to binding of Mg^{2+} and Zn^{2+} ions to the receptor [18, 58]. The role of Ca^{2+} in synaptic plasticity is well-established [12]. Not only does Ca^{2+} cause the growth of new additional AMPA receptors, but it also increases the single-channel conductance of individual AMPA receptors [12]. As stated earlier, the number and efficacy of AMPA receptors on a synapse are largely responsible for defining the weight of a synapse.

Now that we understand how NMDA receptors allow Ca^{2+} ions to enter the neuron and how these Ca^{2+} ions affect the synaptic weight, we can now discuss the process of STDP with regards to these processes. The process begins when a presynaptic neuron fires an AP through a synapse into the postsynaptic neuron which leads to the release of glutamate into the synaptic cleft which binds to the NMDA receptors, thus opening them. At this

point however, Ca^{2+} ions cannot flow through the NMDA receptors on the postsynaptic neuron as the passage is still blocked by Mg^{2+} and Zn^{2+} ions bound to the receptor. If the postsynaptic neuron then fires an AP shortly after, the backpropagating AP down the dendrites leads to a brief increase in the synaptic potential, allowing the Mg^{2+} and Zn^{2+} that are bound to the NMDA receptors to detach. Now that the NMDA receptors are open and the Mg^{2+} and Zn^{2+} are detached, Ca^{2+} ions can begin to flow into the postsynaptic neuron which can trigger the growth of new AMPA receptors and an increase in efficacy of already existent AMPA receptors on the synapse. The shorter the amount of time between pre and postsynaptic firing, the larger the increase in synaptic efficacy as the glutmate in the synaptic cleft is cleaned up and the synaptic potential repolarizes. The LTP process is illustrated in Figure 1.4. With regards to LTD, there appears to exist a certain Ca^{2+} threshold, below which the synaptic efficacy tends to decrease [56, 12, 7]. This appears to be related to protein dephosphorylation as a result of reduced levels of Ca^{2+} ions in the synaptic cleft. The shape of the STDP protocol was experimentally characterized by Bi and Poo and can be fitted using two exponential functions as shown in Figure 1.5 [7].

1.2.3 Homeostasis

At this point it should be apparent that a potential problem could arise when a network is subjected to either extremely intense stimuli or stimuli deprivation. In the case of an intense excitatory stimuli, a situation can occur where the high level of neuronal excitation leads to further excitation, especially for recurrent excitatory circuits. Besides the obvious computational effects, this positive feedback loop will eventually lead to excitotoxicity and if left in that state, will kill the neuron. While the inhibitory neurons play a large role in down-regulating the burst in neuronal activity, without a perfect balance of excitatory and inhibitory neurons, the feedback loop will remain unbalanced. Further, another positive feedback loop is created due to the fact that higher levels of activation in the network increase the likelihood of Hebbian-based LTP which drives even further excitation. As expected, many homeostatic mechanisms exist to regulate the average activity of neuronal networks at both the cellular and network level. At the time of writing, the exact mechanisms by which this occurs in biological neuronal networks are not fully understood or characterized. That

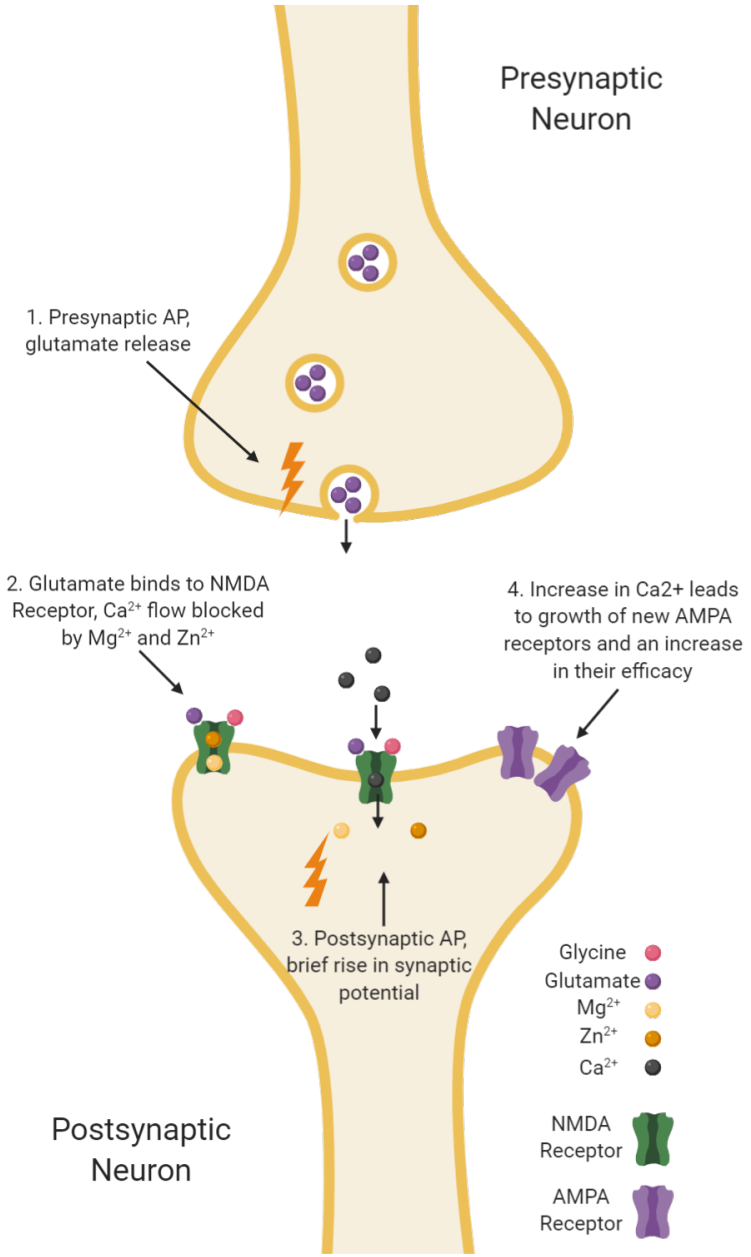


Figure 1.4: The basic steps by which pre-before-post synaptic spiking leads to LTP. Figure created with BioRender.com

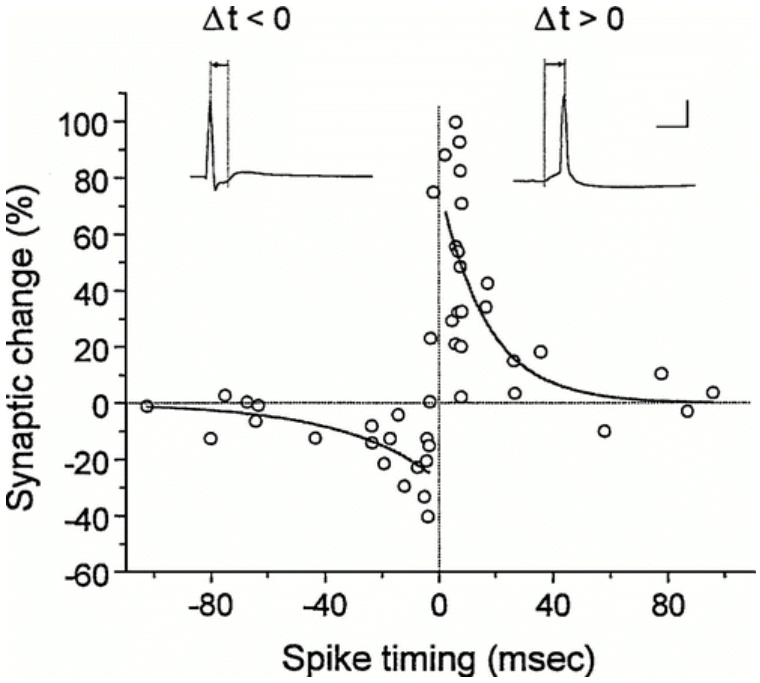


Figure 1.5: Shape of the STDP protocol for rat hippocampal neurons as determined by Bi and Poo, with exponential fit [7].

being said, there are two categories of mechanisms that have been observed experimentally: spike frequency adaptation and synaptic scaling. Spike frequency adaptation is concerned primarily with the first positive feedback loop that was discussed by regulating a single neuron's average spike frequency. Synaptic scaling addresses the second positive feedback loop by scaling a neuron's dendritic synapses such that an average level of stimuli is maintained on its dendrites even when the weights of the synapses change. A large body of research from a computational modelling perspective is available on the topic of homeostasis. In fact, at the time of writing, it appears that the majority of work in this area is computational rather than experimental. The computational models of homeostasis used in this project are discussed further in Section 2.3.

Spike Frequency Adaptation

Spike frequency adaptation is essential from a biological perspective as it helps to prevent excitotoxicity. Furthermore, it is *fundamentally* important to the computational health of

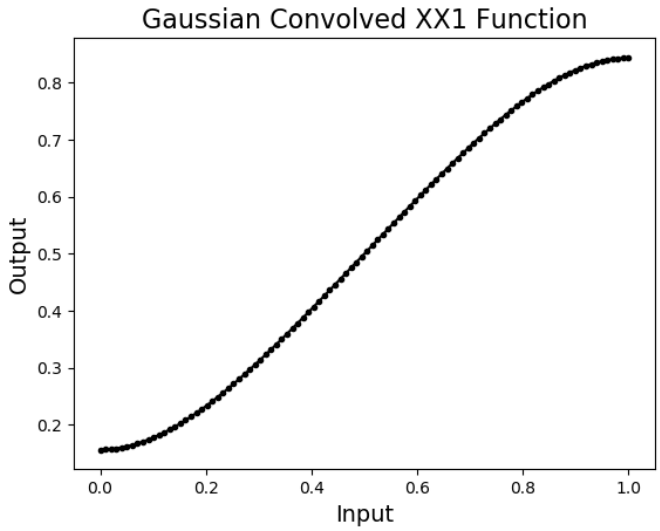


Figure 1.6: Approximation of the shape of a typical neuron's F-I/gain curve modelled as an XX1 function convolved with Gaussian noise ($\sigma = 20$) [56]

the neuron as it has been shown to act as a high-pass filter, allowing the neuron's response to be mostly independent from the average intensity of the incoming stimuli [4]. Without spike frequency adaptation, one can expect that the neuron's firing rate would increase linearly with respect to the synaptic input current. However, this is not the case, in fact experimental data has shown that the neuron's firing rate as a function of input current (called the F-I or gain curve) takes a form similar to that of a XX1 function ($f(x) = \frac{x}{x+1}$) convolved with Gaussian noise as shown in Figure 1.6 [56].

One mechanism behind spike frequency adaptation is mediated by a type of voltage dependent K^+ ion channel called the M-current [9]. The M-current K^+ ion conductance increases when the neuron is depolarized thus forcing more K^+ ions to *exit* the cell [9]. The loss of K^+ ions in the cell reduces the ability of the incoming excitatory Na^+ ion flux to depolarize the neuron and cause an action potential. It has also been shown that this effect can be enhanced by increasing the intracellular Ca^{2+} ion concentration [72], which occurs when the neuron's activity increases [27, 6]. The effect of Ca^{2+} ions on the M-current conductance is shown in Figure 1.7 [72]. As the neuron continues to fire repeatedly, the influx of Ca^{2+} combined with the efflux of K^+ ions from the cell has

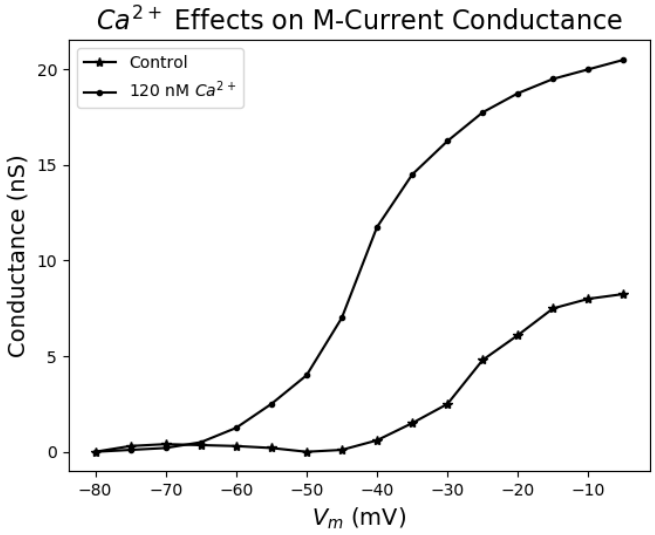


Figure 1.7: M-current K^+ ion conductance as a function of membrane potential with and without added intracellular Ca^{2+} ions. Data extracted from figure in [72].

the effect of temporarily reducing the effective firing rate of the neuron. In fact, it was shown that suppressing M-current channels through the use of muscarine (which mimics the neurotransmitter acetylcholine) resulted in a complete change in neuron firing behavior from phasic to tonic firing, thus demonstrating the importance of m-current channels in regulating the firing pattern of neurons [9].

The M-current is not the only voltage-dependent ion channel responsible for maintaining neuronal firing rate stability. Other voltage-dependent potassium channels have been identified in mice and humans that have similar effects on neuron excitability [39]. The importance of the voltage dependent potassium ion channels is made evident by the fact that irregularities in these channels is well known to be related to epilepsy, which is a disorder characterized by over-active neuronal activity in the brain [39, 2].

Synaptic Scaling

Synaptic scaling is one method by which neurons can combat the Hebbian-induced positive feedback loop that leads to instability in the network activation patterns [5, 71]. Synaptic scaling is a negative feedback loop where the neuron modifies the overall input excitation

levels by uniformly and multiplicatively scaling the dendritic synaptic weights. In this way, the overall level of input excitation on the neuron can be adjusted while still maintaining the relative synaptic weights that were learned via Hebbian mechanisms. It has become clear experimentally that many of the mechanisms governing synaptic scaling operate by adjusting the number of surface AMPA receptors on the postsynaptic neuron [5]. In order for the AMPA receptors to respond to chronic changes in synaptic activity, the neuron must have both a means of detecting said chronic activity and also a method to appropriately reduce the number of AMPA receptors on the postsynaptic dendrites [5].

Studies have shown that increases in synaptic activity lead to expression of a set of genes responsible for modifying the number of AMPA receptors on the postsynaptic dendrites [5]. Of notable importance is the Plk2 gene whose expression can be triggered by sustained synaptic activity in hippocampal neurons [57, 66]. One of the sub units comprising AMPA receptors is encoded by the glutamate ionotropic receptor AMPA type subunit 2 gene (GluA2). GluA2 can be modulated by Plk2; in particular, higher levels of Plk2 can lead to lower levels of GluA2 endocytosis, ultimately leading to a reduction in the number of surface AMPA receptors [5]. In fact, by blocking the Plk2 gene, synaptic downscaling can be eliminated, demonstrating the importance of the Plk2 gene for synaptic scaling [5]. Triggering the Plk2 genes to begin the series of events leading to down-scaling of the synaptic activity is another problem entirely. In the case of Plk2, the dihydropyridine (DHP) calcium ion channel appears to be the primary signalling pathway for the Plk2 gene expression [57]. The DHP ion channel is voltage-dependent, i.e. the calcium ion conductance is maximized when the neuron spikes [62]. As such, expression of the Plk2 gene and its resultant down-scaling of synaptic weights is proportional to the firing rate of the neuron, exhibiting a homeostatic relationship between firing rate and synaptic weight down-scaling.

The Homer1a gene has also been shown to play an important role in synaptic scaling [5, 35, 38]. Like Plk2, expression of the Homer1a gene can be initiated by higher chronic levels of synaptic activity [35, 14]. Specifically, higher levels of activity trigger the Homer1 gene to induce expression of the Homer1a isoform which is the one responsible for regulation of synaptic activity [14]. Homer1a is able to signal group 1 metabotropic glutamate receptors (mGluRs) without the presence of a mGluR agonist [35]. The Homer1a-induced signalling

of the group 1 mGluRs leads to tyrosine phosphorylation of GluA2, resulting in a down-regulation of GluA2 endocytosis and a consequent reduction in the number of postsynaptic AMPA receptors [35]. Interestingly, the Homer1a-induced signalling of group 1 mGluRs has been shown to be a cell-wide mechanism [35]. The cell-wide action of Homer1a demonstrates the multiplicative and weight-ratio preserving nature of synaptic scaling.

As discussed, the down-scaling of synaptic activity resulting from a chronic increase in synaptic activity can largely be attributed to expression of the Plk2 and Homer1a genes, however, these two genes do not account for upscaling of synaptic weights. Cell-wide synaptic upscaling has been experimentally observed following chronic reduction in synaptic activity [21]. One mechanism to explain this synaptic upscaling involves the calcium ion influx. It was shown that a reduction in synaptic activity leads to a reduction in postsynaptic calcium ion influx [37]. This reduction in calcium ion influx leads to the transcription of proteins that ultimately leads to an increase in postsynaptic AMPA receptors [37]. While the exact genes and proteins involved in this process are not fully determined, it has at least been narrowed down to a process that occurs within the postsynaptic neuron, independent of presynaptic spikes [21].

1.2.4 Neuromodulated STDP

One concept of STDP that has been largely ignored thus far is the fact that the behavior of STDP can be significantly altered by chemicals in the brain called neuromodulators. Neuromodulators are similar to neurotransmitters, however, they affect entire groups of neurons simultaneously rather than affecting only single neurons. On its own, STDP is effectively an unsupervised statistical learning rule. As discussed in Section 1.2.2, STDP acts like a high-pass filter, extracting features from noisy input that are statistically significant with respect to the spike rate. The addition of neuromodulators introduces a method by which STDP can become a supervised, reward-based, or semi-supervised learning algorithm. As will be discussed, neuromodulators can affect STDP in many diverse ways by gating its action, reversing its polarity, changing its sensitivity, altering its time course, etc. By combining multiple neuromodulators under different conditions, these effects can be combined to transform STDP into many different learning paradigms.

Many different neuromodulators exist in the brain, however the four major ones are serotonin, norepinephrine (noradrenaline), acetylcholine, and dopamine. The experimentally-verified effects that each of these neuromodulators has on STDP are highly diverse as will be discussed. Because it has been heavily studied in the context of neuromodulated STDP, dopamine is discussed in most detail in the following text.

Dopamine

The role that dopamine plays in reinforcement learning in the brain is extraordinarily complex. Its effects on learning in classical and operant conditioning tasks are manifold and not completely understood. Despite an incomplete understanding, is well-established that dopamine plays a major role in many different cognitive tasks across the brain [26]. Dopamine levels have been shown to increase during the receipt of reward and decrease during the receipt of punishment [60, 69]. In the striatum nucleus of the basal ganglia, it has even been shown that the level of dopamine released by dopaminergic neurons is proportional to the error between the received reward and the expected reward [26, 65], much like the error function used in the temporal difference (TD) algorithm in traditional reinforcement learning [61]. While this particular discovery is exciting from the perspective of AI research, it may be the case that this is network-level effect, thus we will focus on the effects of dopamine at the level of a single or small number of neurons.

While it has been well-established that dopamine is used to represent reward for many decades, the relationship between dopamine and Hebbian learning is still poorly understood. In order to convert STDP from its unsupervised form into a dopamine-modulated/reinforcement learning form, there must be a mechanism by which dopamine "gates" STDP such that only synapses contributing to the receipt of dopamine undergo synaptic modification. This brings up another problem called the credit assignment problem; a reward (and thus dopamine) is often received many seconds or minutes after the reward-invoking stimuli is observed and an action is taken, so how does dopamine affect STDP after the network's state has already been changed? In order to solve this problem, Sutton and Barto introduced the possibility that the synapses may leave a decaying activity trace called an eligibility trace that could later be activated by reward [70]. Interestingly, eligibility

traces that are activated by specific neurotransmitters were *later* discovered to actually take place at the neuronal level [30]. In particular, it was discovered that each synapse actually encodes two separate eligibility traces, one for LTP and one LTD, that each have different temporal structures and are affected in different ways by specific neuromodulators [30]. The electrochemical nature of how these eligibility traces form is still not fully understood, but may be the result of residual activity of calcium/calmodulin-dependent protein kinase II (CaMKII) [30] which is involved in the general process underlying STDP [45] (also see Section 1.2.2). The researchers who discovered these eligibility traces were able to characterize their temporal behavior and experimentally determine the effects that dopamine, serotonin, norepinephrine, and acetylcholine have on each by inducing pre-before-post and post-before-pre firing patterns. It was shown that the pre-post LTP eligibility traces last about 10 seconds with a half-life of roughly 5 seconds and the post-pre LTD eligibility traces last only about 5 seconds [30]. The effects that different neuromodulators had on these eligibility traces are summarized in Table 1.1. Without any induction of neuronal firing, both LTP and LTD did not occur, confirming that the neuromodulators were not enabling synaptic plasticity alone but were actually converting eligibility traces into concrete synaptic plasticity [30]. It is important to note that the eligibility trace experiment only investigated a very small aspect of STDP neuromodulation and is thus not a comprehensive summary of the effects that each neuromodulator has on STDP.

Neuromodulators do not only have a gating effect on STDP, but they also can completely reverse its polarity to generate a process called anti-STDP. It has been shown in various studies that under particular conditions, dopamine can invoke anti-STDP in prefrontal cortices through activation of postsynaptic NMDA receptors [63]. Likewise, the same affect has been demonstrated in the somatosensory cortex [43]. This may enable networks to not only learn what stimuli and actions leads to reward, but which ones *will not* lead to reward.

Acetylcholine

Like dopamine, acetylcholine is largely associated with reinforcement learning tasks in the brain. Studies on mice have shown that acetylcholine is very strongly tied to punishment, or negative reinforcement feedback [28]. Specifically, it was shown that acetylcholine is

Table 1.1: Effects of various neuromodulators on LTP and LTD synaptic eligibility traces as observed in pyramidal cells of the layers II/III of the visual and prefrontal cortices [30]. LTP eligibility traces were created by inducing a pre-before-post firing pattern and LTD eligibility traces were created by inducing a post-before-pre firing pattern [30]. As discussed in Section 1.2.2, pre-post induces LTP and post-pre induces LTD in the standard STDP protocol.

Neuromodulator	**Visual Cortex**		**Prefrontal Cortex**	
	Pre-Post	Post-Pre	Pre-Post	Post-Pre
Dopamine	None	None	LTP	None
Acetylcholine	None	None	None	None
Norepinephrine	LTP	None	LTP	None
Serotonin	None	LTD	None	LTD

released precisely at the time in which the mouse senses a punishment-indicating cue, indicating that acetylcholine is not only an indicator of punishment itself, but an indicator of impending punishment/negative reinforcement [28]. The response during reward/positive feedback on the other hand is quite different. It was shown that while some of the cholinergic (acetylcholine) neurons respond to reward in much the same way as they do to punishment/negative reinforcement, a significant portion of the cholinergic neurons respond proportional to the reward expectation error (the researchers called this "reinforcement surprise" in their article) [28]. Through their study and literature review, this particular team of researchers speculated that acetylcholine enables unsupervised plasticity mechanisms to take the form of a supervised learning rule [28]. This statement is echoed frequently in the theory behind the computational methods in this thesis.

Other studies have demonstrated that in the mouse hippocampus (heavily associated with memory), acetylcholine is able to transform standard asymmetrical STDP into symmetrical STDP (traditional Hebbian plasticity) [10]. The researchers developed a computational model of their theories and showed that acetylcholine along with the dopamine facilitate reward-based navigation tasks [10]. The symmetric form of STDP occurs in various locations in the brain, and has been shown to be especially useful for accurate memory storage and

recall [54]. This is a further demonstration of the flexibility of Hebbian learning principles in biological neural networks.

Norepinephrine

As stated in section 1.2.1, the K^+ ion plays a large role in determining the excitability of a neuron. The K^+ leakage current acts to constantly polarize the neuron, thus an increase in the K^+ leakage conductance results in a neuron with a lower excitability. Also, as discussed in section 1.2.2, the induction of LTP via the STDP mechanism results in a pre-before-post spike chain. Because the K^+ ion leakage conductance is directly correlated with a neuron's excitability and the excitability is directly correlated to the neuron's ability to fire, then the K^+ ion leakage current is also directly correlated with the induction in STDP-enabled LTP.

It has been experimentally determined that norepinephrine increases neuron excitability, however the mechanisms by which this occurs was not fully determined until recently [47]. It was experimentally demonstrated that when norepinephrine is released into the synaptic cleft, it binds to the beta-2 adrenergic receptor [47]. When bound, the beta-2 adrenergic receptor reduces the prevalence of K^+ ion channels on the surface of the cell [34]. As discussed, a reduction in the number or efficacy of K^+ ion channels on the synapse results in a higher excitability of the neuron. This higher excitability promotes STDP and also increases the time window in which the STDP protocol is sensitive [47].

It has been hypothesized that norepinephrine acts to enhance memory and associative learning during states of brain arousal, as this is when norepinephrine is often released [47, 51]. Also, similar studies on the relationship between arousal and norepinephrine have shown that it selectively amplifies learning of salient features and suppresses the learning of other features [51]. These observations may indicate that norepinephrine plays a role in signalling novelty and through STDP acts to enhance associative learning between novel events and their causes.

Serotonin

Less is known about the neuromodulatory effects of serotonin on STDP than for the previous aforementioned neuromodulators. There is some evidence that like dopamine,

serotonin signals the occurrence of an unexpected event [52]. Unlike dopamine however, experimental results have indicated that serotonin signals unexpected events regardless of their reward status, including unexpected events with no definable positive or negative reward [52]. Interestingly, results suggest that serotonin directly inhibits neural circuitry that is responsible for causing the occurrence of unexpected events; i.e. actions that lead to unexpected results are inhibited by serotonin [52].

1.2.5 Network Structure

Because the research presented in this thesis is focused on the computational functions of biological neural systems at a fundamental level (neuron/small network scale), this thesis will not go into too much depth on the extremely complex organization of neural networks in the human brain. There are however a few common features of the neural network architecture in the human visual cortex that have important computational benefits to the network. This section will discuss some of these topics.

Bottom-Up Learning

Bottom-up and top-down are two different approaches to information processing. In top-down information processing, in order to develop an understanding of a complex system, it is broken down into smaller parts and these sub-parts are analyzed for their function. For example, if one was asked to formulate a description of a cat, a top-down approach may take this picture of a cat and break it down into smaller but easier to understand components such as ears, eyes, teeth, whiskers, etc. In bottom-up processing, the process of obtaining understanding about a complex system occurs by first analyzing simple parts and combining them together in a hierarchical fashion to create the overall picture. A bottom-up approach to the cat problem may first attempt to understand very specific low-level features of the cat image such as edges, corners, and basic shapes. It might then entail combining these features to create more complex structures such as ears, eyes, teeth, whiskers, etc. From these more complex features, we can then begin to describe the image of a cat. Both approaches are different but can be complementary. Top-down focuses on the big picture and works its way

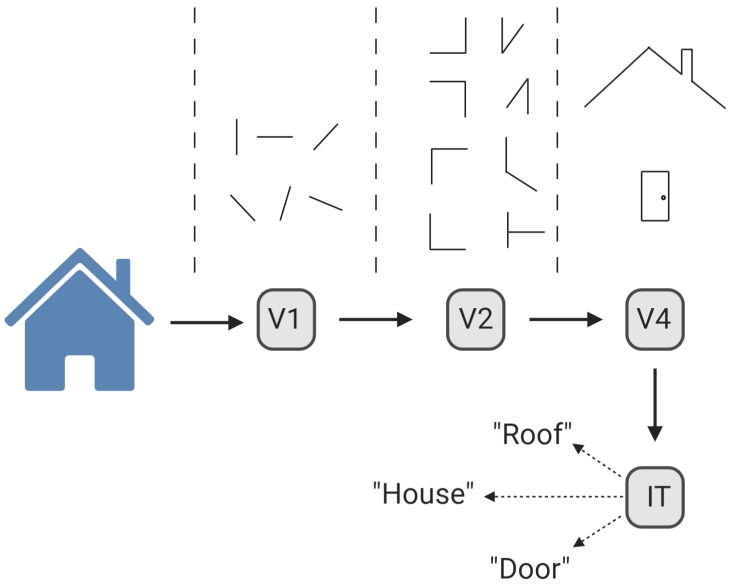


Figure 1.8: A simplified diagram of visual processing in the visual cortex. Taking a bottom-up approach, the learned visual features progress from elementary edges and corners in earlier layers to complex shapes and object recognition in the final layers.

down to the details whereas bottom-up focuses on the details and works its way up to the big picture. As it turns out, the brain performs both forms of image processing [56].

While both forms of learning are present in the brain, many of the most complex information processing neural circuits use bottom-up processing. One such example is the visual cortex. The visual cortex consists of four main areas labeled V1, V2, V4, and the inferior temporal (IT) cortex. As shown in Figure 1.8, the V1 area of the visual cortex is primarily sensitive to edges at different orientations, followed by the V2 area which combines these edges into simple shapes, followed by V4 which combines said shapes into complex objects, finally leading to the IT cortex which answers the question of "what" is in the visual scene. The reality is obviously much more complicated than Figure 1.8 presents, however the big picture is that the visual cortex has a hierarchical connectivity scheme that enables bottom-up processing of very complex information.

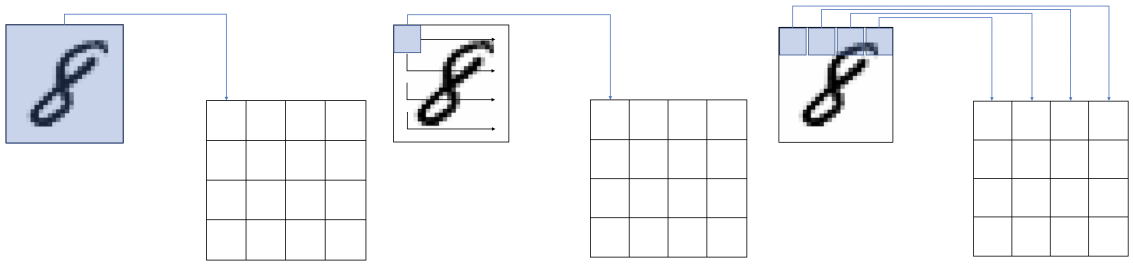
Receptive Fields

One important concept in neural information processing (especially visual processing in the human visual cortex) is that of receptive fields. The receptive field of a neuron is the location of the input space in which the neuron receives information. In the case of traditional ANNs, there are most commonly two different receptive field types referred to as dense and convolutional. In the case of a dense receptive field, each neuron in layer 2 is connected to every neuron in layer 1 as in Figure 1.9a. In the case of convolutional receptive fields, each neuron in layer 2 has a receptive field of size $(k_1 \times k_2)$ that is convolved with the outputs of neurons in layer 1 at a given stride. This is demonstrated in Figure 1.9b where the black arrows indicate the direction of the sliding receptive field. The use of convolutional ANNs, referred to as convolutional neural networks (CNN), resulted in a huge leap in ANN performance on visual processing tasks. When creating CNNs with many layers, which coined the term "deep learning", the neuron's receptive fields have been shown to learn features in a way very similar to that shown in Figure 1.8. This is further discussed in Section 1.3.2.

While convolutional receptive fields have proven their success in the ANN space, there is no mechanism in biology in which neurons can convolve their receptive fields over the input space. This being said, biological neurons do not have dense receptive fields either, rather, their receptive fields are localized to receive input from only a small portion (size $k_1 \times k_2$) of the input space relative to its total size as shown in Figure 1.9c.

Lateral Inhibition and Local Excitation

One can imagine that if neighboring neurons that utilize STDP have similar receptive fields with similar weights, then these neurons may tend to learn the exact same features of the input space. Consider the example in Figure 1.10, where a group of four neurons in V1 of the visual cortex all share the same visual receptive field. In this case, a problem may occur where all of these neurons learn to become sensitive to the most significant feature in the receptive field, which in this case is the thick line near the top of the receptive field.



(a) Dense **(b)** Convolutional **(c)** Local

Figure 1.9: Dense, convolutional, and local receptive field types demonstrated with a numeric 8 input space and a 4x4 matrix of neurons. The blue shaded regions indicate the receptive fields.

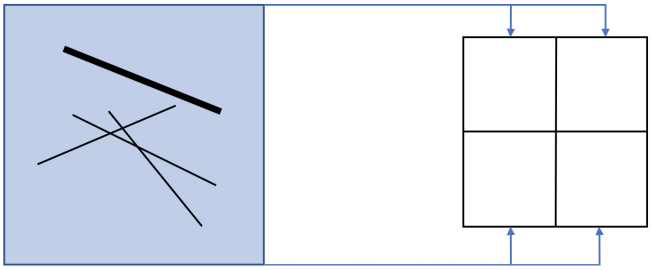


Figure 1.10: An example of a group of four neurons sharing the same receptive field. Without competitive learning, each of these neurons would likely only learn the most significant feature in the input space, which in this case is the thick line near the top of the receptive fields.

With this particular feature begin so strong relative to the others, the other features in the receptive fields will likely be drowned out by the high activity of the significant feature.

Learning the same or similar features among many neurons can be a good thing as it increases redundancy such that if one of the neurons fails to activate, the others may take its place. However, too much redundancy can result in a network that only responds to a small subset of features and is thus limited in what it can detect. The ideal network would achieve

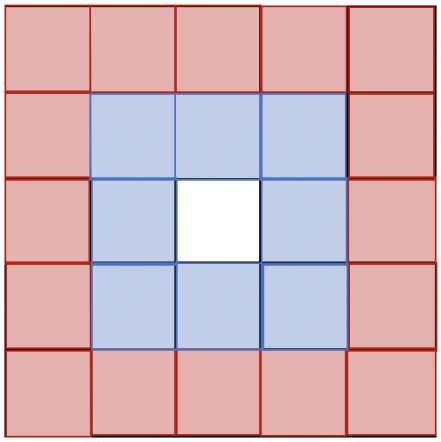


Figure 1.11: Example of a neuron (center square) projecting lateral inhibitory (red) connection to non-neighboring neuron and local excitatory connections (blue) to neighboring neurons.

the right balance between neuronal feature redundancy and specialization. Biological neural networks do this using the concepts of local excitation and lateral inhibition [56]. Local excitation is the property of neurons to grow excitatory connections to neighboring neurons such that they are encouraged to activate to similar stimuli. Likewise, lateral inhibition is the property of neurons to grow inhibitory connections to non-neighboring neurons such that these neurons are discouraged from activating to similar stimuli. Achieving the right balance between lateral inhibition and local excitation allows the network to learn the full space of input features using competitive learning while maintaining a healthy level of redundancy. These two concepts can be visualized in Figure 1.11.

1.3 Previous Work in Biologically Inspired Spiking Neural Networks

1.3.1 The Diehl-Cook Model

One of the most highly cited attempts at using biologically-inspired learning methods for training artificial SNNs is often referred to as the Diehl-Cook model, named after the authors of the paper in which it was presented [16]. The authors focused on learning the ubiquitous Modified National Institute of Standards and Technology (MNIST) dataset which consists of 60000 training and 10000 testing images (28x28 pixels) of handwritten digits from 0 to 9 [42]. The model uses a very simple network architecture consisting of a single input layer in which the input images are encoded, a single computational layer consisting of excitatory neurons, and a single layer of inhibitory neurons for enabling lateral inhibition as seen in Figure 1.12. The input layer is densely connected to the excitatory layer, where each constituent neuron is connected in a one-to-one manner to each neuron in the inhibitory third layer [16]. Each neuron in the third layer then connects to every neuron in the second layer, except for the one that activated it [16]. This lateral inhibition encourages each neuron in the second layer to learn a unique digit in the input datas.

Each digit is shown to the network and a form of online-STDP learning rule is applied, as further discussed in 2. In order to limit the possibility for any single neuron to overrun the network activity, a form of one-sided spike frequency adaptation is used [16]. After training, each neuron in the second layer (output layer) is assigned a class by presenting the training digits to the model and assigning each neuron to the class in which it is most sensitive [16]. During testing, the output spikes are tallied and the class with the highest firing rate wins [16].

Diehl and Cook evaluated their model's performance for a variety of output layer sizes ranging from 100 to 6400 neurons [16]. The 100-neuron and 6400-neuron models achieved accuracy values of 82.9% and 95.0%, respectively [16]. The receptive fields of the 100-neuron model after training are displayed in Figure 1.13. As can be seen, each neuron in the output layer learns the shape of a single digit.

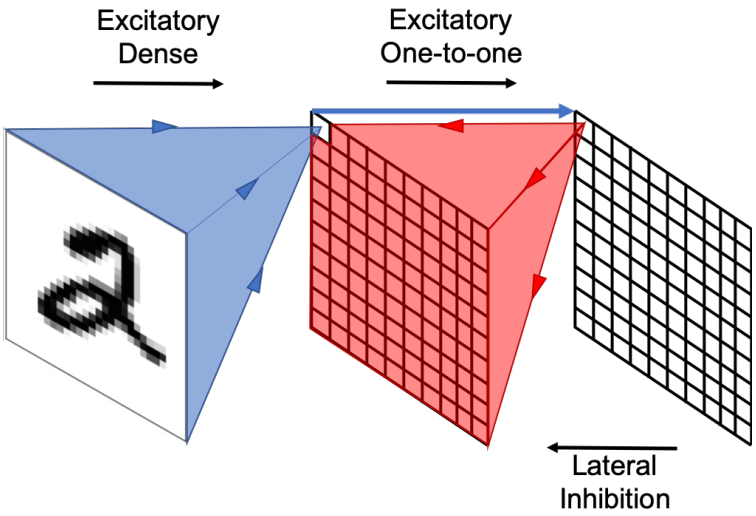


Figure 1.12: Diagram of the Diehl-Cook SNN model [16]. Blue and red represent excitatory and inhibitory synapses, respectively.



Figure 1.13: Receptive fields of the 100-neuron Diehl-Cook SNN model after training, directly from [16].

1.3.2 Convolutional Spiking Neural Networks

Considering the high level of success that traditional artificial neural networks have met in the past decade, a few attempts at creating deep convolutional SNNs (DCSNNs) have been pursued. A large number of these, including one published by one of the authors of the Diehl-Cook model described in Section 1.3.1, are actually traditional DCNNs that are converted into spiking neural implementations [17, 36, 64]. While they do not use biologically realistic neurons or learning paradigms, the processing pathway of traditional deep convolutional neural networks do work similarly to the primate visual pathway in that they process information in a bottom-up manner. The first layer/s in the network are generally develop sensitivity to edges of different sizes and orientations, where the following layers/s assemble the edge responses into corners and other primitive shapes, followed by higher order processing in the final layers. Another motivation behind converting DCNNs into DCSNNs is to achieve the performance of a DCNN while dramatically reducing the inference power consumption by allowing the model to operate on a spiking neural network chip. Some examples of such neuromorphic chips are the IBM True North [53] and the Intel Loihi chip [44]. I believe that efforts such as this are certainly worth pursuing, however given their high level of abstraction, they cannot be considered biologically-plausible. Figure 1.14 shows a generalized hierarchy of different neural network models with respect to biological realism. One of the most important questions in the field of neural network research is what level of abstraction can be applied to the biological mechanisms in order to achieve the same level of performance as a true biological network? This question cannot be answered in this study, however the scope of this particular study lies within tier II neural networks.

Some DCSNN models have been developed that use STDP as the training mechanism [40, 55, 46]. One of these models (referred to as the Kheradpisheh model) uses a series of convolutional and max-pooling layers similar to those used in traditional DCNNs [40]. The convolutional layers in the model use STDP to learn features of the input dataset (Caltech Face/motorbike and ETH-80 datasets in this case) and the max-pooling layers do not use any learning mechanisms but are included in order to achieve spatial feature invariance [40]. The final pooling layer of the model is fed to a linear support vector machine (SVM)

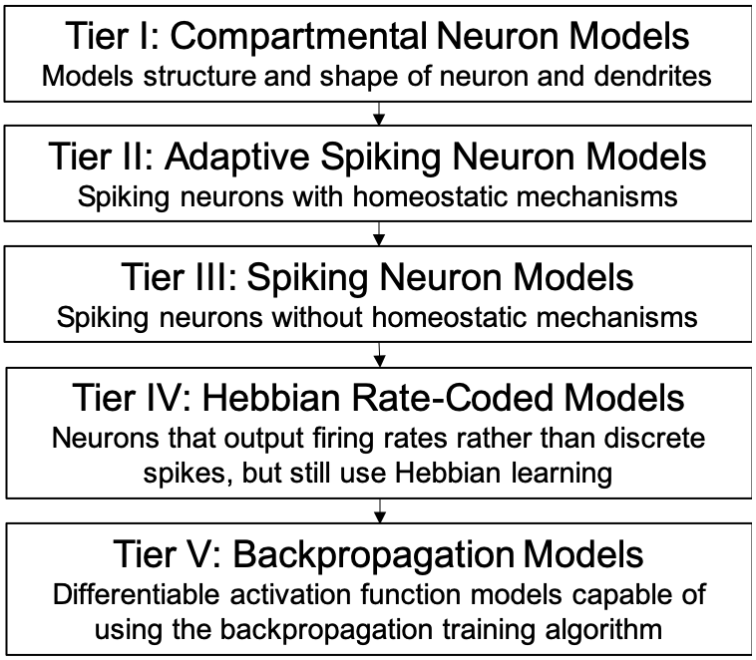


Figure 1.14: Generalized hierarchy of biological realism in popular neural network models ranging from most (top) to least biologically realistic (bottom).

classifier that is trained to output the final classification [40]. Some of the same authors of this model later published another model (referred to as the Mozafari model) that did not use the SVM classifier but used a reward-modulated STDP method to output the final classification decision [55]. This model used first-order spike encoding where the first spike registered in the final classification layer was considered to be the network's "decision" [55]. If the decision was correct, the model would undergo STDP and if the model was incorrect, then the model would use anti-STDP [55]. This approach resulted in an accuracy of 97.2%, a noticeable improvement over the Diehl-Cook model discussed in 1.3.1 [55].

Chapter 2

Biologically Inspired Training of Artificial SNNs

This chapter discusses the theory and methods behind how to implement the biological processes discussed in Chapter 1 into a computational model. The specifics of how these principles were implemented in code are described in Appendix A. The large majority of the equations described here are in the form of differential equations. In the Ganglion simulation package, the differential equations are solved using Euler's method. Euler's method makes the assumption that the shape of the curves at small deltas is roughly linear, such that if one was to use an infinitesimally small integration time step (Δt), then the error in using the Euler's method is zero. Of course, it is not possible to use an infinitesimally small Δt value, thus it must be chosen to be sufficiently small that the linear approximation makes sense. In the case of Ganglion, a Δt value of between 0.01 and 0.5 is generally sufficient for use up to a single neuron firing rate of up to a few hundred Hz. If the Δt value is too large, the neuron's high frequency spiking region may be poorly approximated, similar to that shown in Figure 1 in Appendix A. If using one of the neuron models with spike frequency adaptation, one of which is discussed in this chapter, then the firing rate is self-limiting, making it easier to use larger Δt values, which significantly speeds up the computation.

2.1 Neurons

A large variety of computational models of spiking neurons have been developed over the years with biological realism ranging all the way from highly detailed compartmental models such as GENESIS [8] to simple models such as the integrate and fire (IF) neuron [1]. In fact, the Ganglion simulation environment currently contains four well-known neuron models: the integrate and fire (IF), leaky integrate and fire (LIF), exponential leaky integrate and fire (ExLIF), the adaptive exponential integrate and fire (AdEx), along with a few custom and less common models. For details on each of these models, see Appendix A. In general, the work performed in this thesis will not make any use of the IF model as it does not include any form of leakage current which is important for maintaining homeostatic network activity (see Sections 1.2.3 and 2.3). On the contrary, the exponential depolarization behavior in the ExLIF and AdEx models is rather insignificant and has a negative impact on the computational requirements of the models. Studying the impact of this exponential depolarization behavior on the computational performance of these models is left for future investigation.

The LIF model represents a good middle-ground between extreme biological realism and only minor biological realism. The model is described by the differential equation defined in Equation 2.1, where V_m is the neuron's membrane potential, V_r is the neuron's resting potential, τ_m is the neuron's decay time constant, I_{syn} is the synaptic input current from the neuron's dendrites, C_m is the neuron's membrane capacitance, t is time, t_s is the time of the last spike, and t_r is the refractory period, i.e. the period of time after a neuron spikes in which it cannot become excited.

$$\frac{dV_m}{dt} = \left(-\frac{V_m - V_r}{\tau_m} + \frac{I_{syn}}{C_m}\right)\delta(t - t_s - t_r) \tag{2.1}$$

As can be seen in Equation 2.1, the neuron's membrane potential grows when current is applied to its dendrites at a rate inversely proportional to the neuron's membrane capacitance. The neuron's polarization current causes the membrane potential to exponentially decay with a time constant of τ_m. The time constant τ_m can be related to the K^+ leakage channel conductance g_l by Equation 2.2. From analyzing Equations 2.1 and

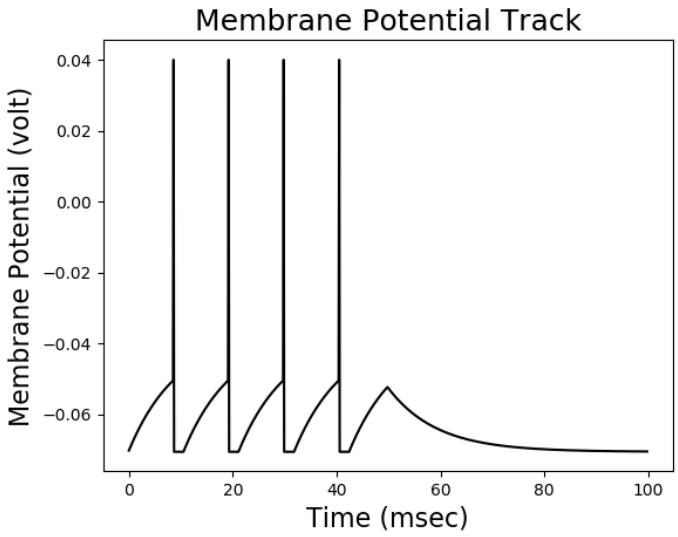


Figure 2.1: Response of a single LIF neuron to an input current that starts at a constant value of 1nA and turns off at 50ms. $V_r = -70.6\text{mV}$, $V_{\text{thr}} = -50.4\text{mV}$, $\tau_m = 9.37\text{ms}$, $C_m = 281\text{pF}$, and $g_e = 100\text{nS}$

2.2, it is hard to not see the resemblance between these equations and those used to define the properties of an RC-circuit in electrical engineering. This relationship provides a good analogy for making sense of the dynamics of the LIF neuron model.

$$\tau_m = \frac{C_m}{g_l} \tag{2.2}$$

The neuron produces a spike at time t when $V_m > V_{thr}$, where V_{thr} is the activation threshold potential for the neuron. For demonstration/plotting purposes, the neuron's spike voltage V_{spike} is set to a value of 40mV, however when using the LIF neuron model and alpha synapse model (Section 2.1.1), the value of V_{spike} does not matter. After the spike occurs, the time of last spike t_s is set to t and V_m is reset to the value of V_r. Figure 2.1 demonstrates the response of a single LIF neuron to an input current that starts at a constant value of 1nA and turns off at 50ms.

2.1.1 Synaptic Input Current

When the neuron spikes, this spike is propagated along the axon of the neuron into each synapse on that axon. The neurotransmitters released into the synaptic cleft result in a time-varying change in the conductance of either Na^+ or Cl^- ion channels for excitatory and inhibitory presynaptic neurons, respectively. This time-dependent variation in ion channel conductance is modelled as an alpha synapse [11, 24]. The time-varying spike-induced synaptic current I_{syn} for a given neuron is described by Equation 2.3, where the variables are defined as follows: d is the dendrite index, N_d is the number of dendrites connected to this neuron, s is the spike index, $N_{s,d}$ is the number of spikes that have arrived at dendrite d during some tracked time window, w_d is the weight of dendrite d, V_{rev} is the reversal potential of the presynaptic neuron to which dendrite d belongs and can be either V_E or V_I for excitatory or inhibitory presynaptic neuron type, respectively, $t_{s,d}$ is the time of spike s on dendrite d, and τ_{syn} is the time constant of the alpha synapse.

$$I_{syn} = \sum_{d=1}^{N_d} \sum_{s=1}^{N_{s,d}} \left[w_d \, \bar{g}_d \, (V_{rev} - V_m) \, \frac{t - t_{s,d}}{\tau_{syn}} \exp\left(\frac{-(t - t_{s,d})}{\tau_{syn}} \right) \right] \tag{2.3}$$

The time profile of the alpha synapse conductance is shown in Figure 2.2 for a set of standard parameter values. Decreasing τ_{syn} will result in a shorter time profile of the conductance spike and will more rapidly activate postsynaptic neurons.

2.2 STDP

There are various methods in which STDP can be implemented computationally. Before STDP was established in literature, when symmetric Hebbian learning principles were still being investigated, Gerstner et. al. formulated a method for performing symmetric Hebbian learning. Let's assume that at a particular synapse for presynaptic neuron j and postsynaptic neuron i exists, with an efficacy ("weight") value of w_j. For a particular time window with f presynaptic spikes and n postsynaptic spikes arriving at times t_i^n and t_j^f, respectively, for a total of N total spikes, Equation 2.4 describes the change in weight for synapse w_j [23].

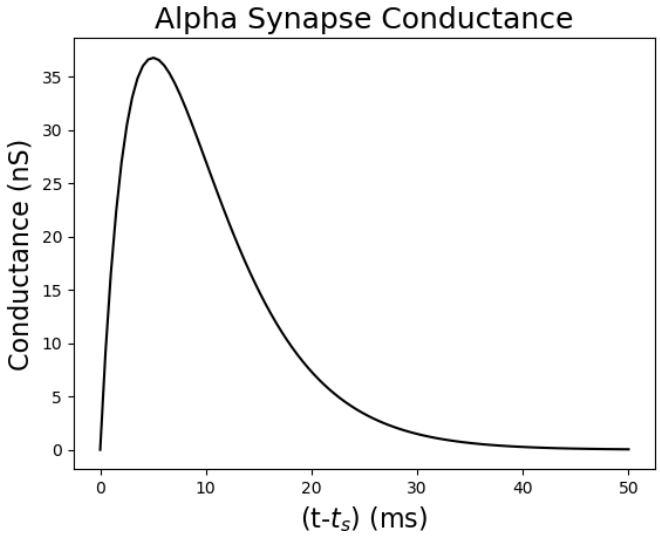


Figure 2.2: Time profile of the alpha synapse conductance model. $\bar{g_d}$ = 100nsiem, τ_{syn} = 5msec, w_d =1

$$\Delta w_j = \sum_{f=1}^{N} \sum_{n=1}^{N} W(t_j^f - t_i^n) \tag{2.4}$$

W is the STDP function, which Gerstner et. al. defined as Equation 2.5 [23]. According to this empirical equation, the STDP function is maximum when the presynaptic spike arrives 0.05 milliseconds prior to the postsynaptic spike, such that $(t_j^f - t_i^n) = -0.05$. For $(t_j^f - t_i^n) < -0.05$, $W(t_j^f - t_i^n)$ will decay exponentially to zero and for $(t_j^f - t_i^n) \geq -0.05$, $W(t_j^f - t_i^n)$ will decay exponentially to a value of approximately -0.13 at $(t_j^f - t_i^n) = 1.75$ and slowly increase back to 0.0 at about $(t_j^f - t_i^n) = 15$. According to this method, LTD only occurs when $15 \geq (t_j^f - t_i^n) \geq 0.45$. This is demonstrated in Figure 2.3.

$$W(t_j^f - t_i^n) = \begin{cases} 0.3 \exp\left[\dfrac{(t_j^f - t_i^n) + 0.05}{0.5}\right] & (t_j^f - t_i^n) < -0.05 \text{ msec} \\ 0.5 \exp\left[\dfrac{-((t_j^f - t_i^n) + 0.05)}{0.5}\right] - 0.2 \exp\left[\dfrac{-((t_j^f - t_i^n) + 0.05)}{5}\right] & (t_j^f - t_i^n) \geq -0.05 \text{ msec} \end{cases} \tag{2.5}$$

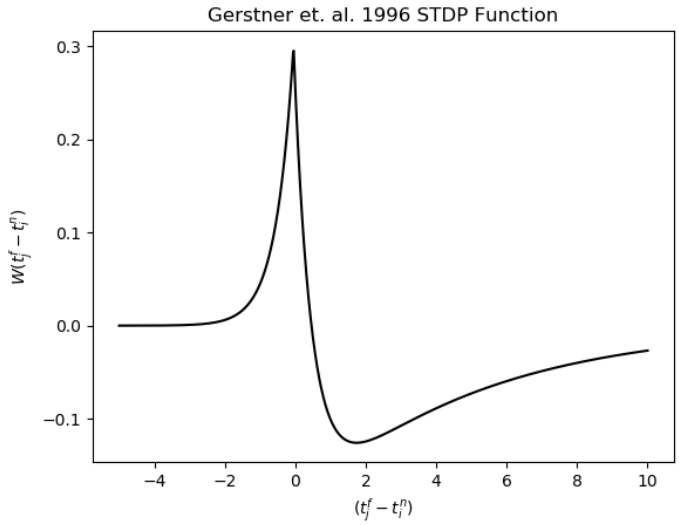


Figure 2.3: STDP function used by Gerstner et. al. as defined in Equation 2.5 [23].

Integrating $W(t_j^f - t_i^n)$ over all values where $W(t_j^f - t_i^n) > 0$ and $W(t_j^f - t_i^n) < 0$ yields values of approximately 106 and -382, respectively, such that assuming a uniform distribution of $t_j^f - t_i^n$, LTD will dominate LTP. With this assumption, this should help aid in the long-term stability of this learning algorithm. Nevertheless, as was discussed in Section 1.2.2, the STDP function generally takes the form of an anti-symmetric exponential function, thus this work will use a different, simpler STDP function that better captures the biological findings. Further, from a programming perspective, Equation 2.4 is rather computationally intensive because it requires the programmer to define a rolling time window for every synapse in which the program will detect and record the timing of spikes. This requires a lot of overhead, especially considering that even a relatively small network may contain hundreds of thousands, or even millions, of synapses. From a biological perspective, there is currently no external mechanism in the brain that observes and tracks activity of every synapse, rather, as discussed in Section 1.2.2, spikes on a given synapse leave chemical traces of their activity in the synapse itself. In particular, preseynaptic spikes release a chemical called glutamate into the synaptic cleft (see Figure 1.4) which is removed from the synaptic cleft by glutamate transporters with a time constant of between 0.7 and 2.0 milliseconds (for the hippocampus)

[13]. Likewise, postsynaptic spikes leave a backpropagating action potential that is sent down the dendrites of the postsynaptic neuron which signals to the synapse that a postsynaptic spike has occurred. This backpropagating AP rapidly increases to a maximum value and then decays exponentially [48]. Not surprisingly, the shape of the backpropagating action potential is very similar to the shape of the forward propagating AP, taking the form of an alpha function as described earlier in this chapter.

Considering the fact that presynaptic and postsynaptic spikes leave traces of their activity at each synapse, we can formulate the STDP protocol in an online fashion. This process is also discussed by Gerstner and Sjöström in the Scholarpedia article on STDP [67]. We will make the assumption that the time-varying trace left by the presynaptic AP in the synapse $a_j(t)$ takes the form of an exponential decay with time constant τ_a as shown in Equation 2.6. Likewise, the postsynaptic AP trace $b_i(t)$ can also be modelled as an exponential decay with time constant τ_b as shown in Equation 2.7.

$$\frac{da_j}{dt} = -\frac{a_j}{\tau_a} \tag{2.6}$$

$$\frac{db_i}{dt} = -\frac{b_i}{\tau_b} \tag{2.7}$$

Each time the presynaptic neuron spikes, $a_j(t)$ is increased by an amount α. Likewise, each time the postsynaptic neuron spikes, $b_i(t)$ is increased by an amount β. For convenience, α and β is generally set to 1. When the presynaptic neuron spikes at time t_s, the weight of the synapse w_j is changed by an amount relative to $a_j(t)$ according to Equation 2.8, where S_{ba} is a scaling factor for post-before-pre spike ordering, $\Lambda(w_j)$ is the weight bounding function, and η is the learning rate. Correspondingly, Equation 2.9 describes the weight update rule for when the postsynaptic neuron fires, where S_{ab} is the scaling factor for pre-before-post spike ordering. $\Lambda(w_j)$ takes the form given by Equation 2.10, where S is either S_{ba} or S_{ab}. This form of a weight bounding rule is called a multiplicative soft bounding rule. In the case where S is positive such that the weight will be increased, the maximum possible weight that can be achieved is 1, such that as the weight of the synapse grows, it becomes more and more difficult to further increase the weight. Contrarily, when S is negative such that

the weight will be decreased, the minimum achievable weight is 0 and it becomes more and more difficult to reach 0 as the weight decreases. The alternative to soft weight bounds is hard weight bounds where the weight is allowed to grow/fall at the same rate regardless of the current weight value, however the soft bounding rule is far more stable and was shown in this project to drastically help the learning process. It also helps to reduce the chance of having over/under active neurons, which are problematic computationally. This concept is well know in the field of traditional deep learning, where weight regularization terms are used to combat the problem [41].

$$\Delta w_j = S_{ba} \times \eta \times \Lambda(w_j, S) \times b_i(t_s) \tag{2.8}$$

$$\Delta w_j = S_{ab} \times \eta \times \Lambda(w_j, S) \times a_j(t_s) \tag{2.9}$$

$$\Lambda(w_j, S) = \begin{cases} w_j & S < 0 \\ 1 - w_j & S \geq 0 \end{cases} \tag{2.10}$$

The scaling factors S_{ba} and S_{ab} allow the programmer to configure a variety of different STDP protocols. By setting S_{ab} to a positive value and S_{ba} to a negative value, the standard STDP protocol is used where pre-before-post synaptic spike ordering leads to LTP and post-before-pre synaptic spike ordering leads to LTD. The magnitude of each scaling factor can be used adjust the strength of LTP and LTD, respectively. By settings both S_{ba} and S_{ab} to positive values, then the traditional Hebbian learning update rule is used. This simple scaling factor can be used to drastically adjust the behavior of the learning rule for different forms of Hebbian learning. The importance of this will be further demonstrated in the following section on neuromodulated STDP (2.2.1).

2.2.1 Neuromodulated STDP

Like many questions in science, the question of how a specific neuromodulator affect synaptic plasticity usually has the same answer, it depends. As described in Section 1.2.4, the

role of these molecules in learning is incredibly complex, dependent upon a plethora of different variables. Given that this subject is so poorly understood even from a biological perspective, the methods here will adopt only a small selection of the biological principles previously discussed, simply out of necessity. This section describes the methods by which these various concepts can be modelled computationally and demonstrates them on the bar-learning problem.

Eligibility Traces and Gated STDP

As discussed in Chapter 1, STDP can be "switched" on and off through neuromodulation with a variety of different chemicals in the brain. This enables the traditionally unsupervised STDP algorithm to perform reinforcement learning by gating STDP at the right times. Also, the credit assignment problem, i.e. the question of how neuromodulators assign credit to the correct spike pairs after they have occurred, was answered by the discovery of synaptic LTP and LTD eligibility traces in biological synapses. In particular, it was shown that two distinct eligibility traces are stored in the synapse, one for pre-before-post synaptic spike ordering E_{ab} and one for post-before-pre synaptic spike ordering E_{ba}. Equations 2.11 and 2.12 describe the exponential decay of these traces with time constants of τ_{ab} and τ_{ba}, respectively.

$$\frac{dE_{ab}}{dt} = -\frac{E_{ab}}{\tau_{ab}} \tag{2.11}$$

$$\frac{dE_{ba}}{dt} = -\frac{E_{ba}}{\tau_{ba}} \tag{2.12}$$

When the pre or postsynaptic neurons on either side of a synapse fire, the STDP spike traces $a_j(t_s)$ and $b_i(t_s)$ are increased by α and β as described in Section 2.2. Under the traditional STDP protocol, Equations 2.8 and 2.9 would be used at this point to adjust the weight of the synapse. In the case of Gated STDP however, the weight does not change but the eligibility traces are updated. When a presynaptic neuron fires, E_{ba} is increased by the current value of the postsynaptic spike trace, $b_i(t_s)$. Likewise, when a postsynaptic neuron fires, E_{ab} is increased by the current value of the presynaptic spike trace, $a_j(t_s)$.

When a reward R, representing the release of a specific neuromodulator is received by the network, the weight of each synapse that is sensitive to that neuromodulator is updated based on the eligibility traces according to Equation 2.13. In order to allow for more flexibility for different forms of STDP, the scaling parameters S_{ab} and S_{ba} take the form defined by Equations 2.14 and 2.15, respectively, where the superscripts (+) and (-) indicate the scaling factors for positive and negative reward values, respectively.

$$\Delta w_j = \Lambda(w_j) \times \eta \times |R| \times [E_{ab} \times S_{ab} + E_{ba} \times S_{ba}] \tag{2.13}$$

$$S_{ab} = \begin{cases} S_{ab}^{+} & R \geq 0 \\ S_{ab}^{-} & R < 0 \end{cases} \tag{2.14}$$

$$S_{ba} = \begin{cases} S_{ba}^{+} & R \geq 0 \\ S_{ba}^{-} & R < 0 \end{cases} \tag{2.15}$$

Defined in this way, a great deal of flexibility is allowed for adjusting the effects of different neuromodulators on STDP. By adjusting the scaling factors in Equations 2.14 and 2.15, traditional STDP, positive and negative symmetric STDP, and anti-STDP can all be implemented. As discussed in Section 1.2.4, different neuromodulators can affect the STDP protocol in many ways, thus formulating the gated STDP protocol in this way allows researchers to study all of these different mechanisms.

2.3 Homeostasis

As described in Section 1.2.3, homeostasis, or the self-regulating property of neuron and neural network activity is a requirement for maintaining the computational health of a biological spiking neural network. The two primary mechanisms guiding these homeostatic properties are intrinsic plasticity, or the ability for neurons to regulate their activity though the modification of intrinsic properties guiding their excitability and synaptic scaling, or the ability for neurons to modify their excitability through the multiplicative scaling of dendritic

weights. These two mechanisms are not fully understood in the biological realm, however enough is known about them to reproduce their general behavior in a computational setting. The methods by which this is done in this thesis will be discussed in this section.

2.3.1 Intrinsic Plasticity

The ability of voltage-dependent K^+ ion channels to regulate neuron excitability was discussed in Section 1.2.3. These ion channels work by increasing the conductivity of K^+ ion channels as the membrane voltage of the neuron increases. As a result, repeated firing of the neuron (repeated increases in membrane voltage) leads to an increasing efflux of K^+ ions from the cell, making it harder for the excitatory Na^+ ions to depolarize the neuron and lead to a spike. In order to reproduce this effect in an artificial neuron, we will use the experimental data from Figure 1.7 to model the M-current conductance as a function of the membrane potential $g_m(V_m)$. Looking at the data, it can be seen that the shape of the $g_m(V_m)$ curve has a form similar to that of a logistic function. As a result, the data was fit to the model in Equation 2.16 where the parameters p_0, p_1, and p_2 are fit parameters.

$$g_m(V_m) = \frac{p_0}{1 + \exp\left[-p_1\left[(V_m - V_R) + p_2\right]\right]} \tag{2.16}$$

The p_0, p_1, and p_2 parameters were fit to the control curve in Figure 1.7 using the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm. The optimum values of p_0, p_1, and p_2 were determined to be 8.443, 0.273, and -36.072 respectively. This fit is shown in Figure 2.4. As can be seen, the model fits the control remarkably well.

From Figure 1.7, it can be seen that the $g_m(V_m)$ curve can be modulated by intracellular Ca^{2+} ion concentrations such that $g_m(V_m)$ becomes $g_m(V_m, C_{Ca^{2+}})$ where $C_{Ca^{2+}}$ is the intracellular Ca^{2+} ion concentration. Due to the lack of data (only two curves) on how the intracellular Ca^{2+} ion concentration affects $g_m(V_m)$, we cannot generate a complete model for $g_m(V_m, C_{Ca^{2+}})$. However, a qualitative analysis of Figure 1.7 indicates that higher levels of Ca^{2+} ion concentration tend to amplify the $g_m(V_m)$ curve. In order to modify Equation 2.16 to support this, the source of $C_{Ca^{2+}}$ and its associated dynamics must be modeled. In Section 1.2.3, it was discussed that $C_{Ca^{2+}}$ increases with neuron activity and vice versa.

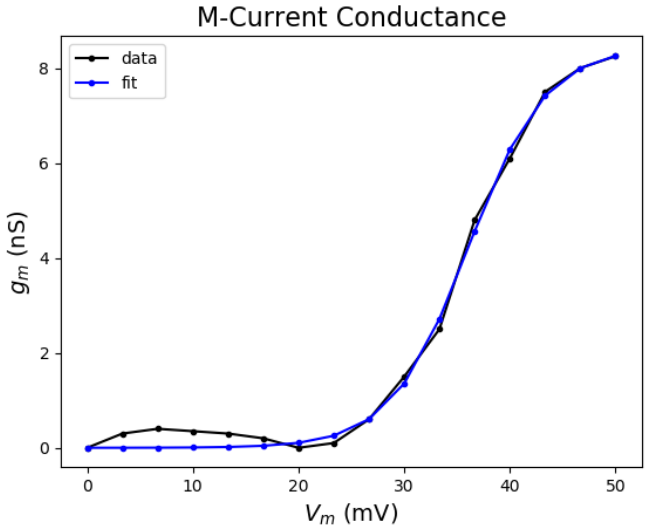


Figure 2.4: Equation 2.16 with p_0, p_1, and p_2 equal to 8.443, 0.273, and -36.072 respectively, plotted against experimental data shown in Figure 1.7 (experimental data from [72]).

This behavior is modelled computationally by introducing a variable C which represents the relative concentration of intracellular Ca^{2+} in the neuron relative to its concentration in an inactive neuron (assumed to be 0). This value decays exponentially according to a time constant τ_C as defined in Equation 2.17.

$$\frac{dC}{dt} = -\frac{C}{\tau_C} \tag{2.17}$$

Using the data for calcium enhancement of M-current conductance from Figure 1.7, it was found that multiplicative modification of the p_0 and p_2 parameters by C in Equation 2.16 was sufficient for modeling the change in g_m as a function of C. This model is described by Equation 2.18. As can be seen, the p_0/p_2 parameter is increased proportional/inversely proportional to C. Intuitively, this makes sense; increasing the intracellular Ca^{2+} ion concentration increases the conductance of the M-current channel and shifts the $g_m(V_m)$ curve to the left such that the M-current is activated at a lower membrane potential.

$$g_m(V_m, C) = \frac{8.443C}{1 + \exp\left[-0.273\left[(V_m - V_r) - \frac{36.072}{C}\right]\right]} \tag{2.18}$$

The M-current can be incorporated into the LIF model as shown in Equation 2.19. This new model is termed the Adaptive M-current Leaky Integrate and Fire (AMLIF) model. The gain curve for the AMLIF model is shown in Figure 2.5 which demonstrates some interesting adaptive-frequency dynamics akin to the behavior of class II neurons as described by Hodgkin Huxley whose work resulted in a Nobel prize [33]. Hodgkin Huxley class II neurons (as opposed to class I neurons) are unable to be activated at arbitrarily low input currents, rather, the spike rate has a minimum current threshold above which the neurons can fire. Beyond this threshold, this model displays spike frequency adaptation where the spike frequency increases and reaches an inflection point beyond which the derivative of the gain curve begins to decrease. The gain curve for the AMLIF model with parameters ($V_r = -70\text{mV}$, $V_m = -70\text{mV}$, $V_{\text{thr}} = -50\text{mV}$, $\tau_m = 10\text{ms}$, $C_m = 1.0\text{pF}$, $\tau_C = 5\text{ms}$, $t_r = 0\text{ms}$) is shown in Figure 2.5. One can expect that the AMLIF model may have a measurable firing rate for subthreshold current inputs due to noise.

$$\frac{dV_m}{dt} = \left(\frac{I_{\text{syn}}}{C_m} - \frac{(V_m - V_r)}{\tau_m} - (V_m - V_r)\frac{g_m(V_m, 1 + C)}{C_m}\right)\delta(t - t_s - t_r) \tag{2.19}$$

2.3.2 Synaptic Scaling

As described in Section 1.2.3, synaptic scaling is a process by which neuron can alter the overall amount of current coming into the cell while maintaining the synaptic weight ratios by multiplicatively scaling the strength of dendritic synapses on the neuron. While the biological mechanisms behind this homeostatic property are quite complex, mathematically they are quite simple. Synaptic scaling is implemented in our model by a simple unit-normalization procedure. Every time a synapse's weight changes, the weights on the neuron's dendrites are scaled according to $\frac{W}{\sum W}$, where $\mathbf{W}$ is the matrix of dendritic weight values for the neuron. This method maintains the relative weights and helps to discourage weight values from becoming too small.

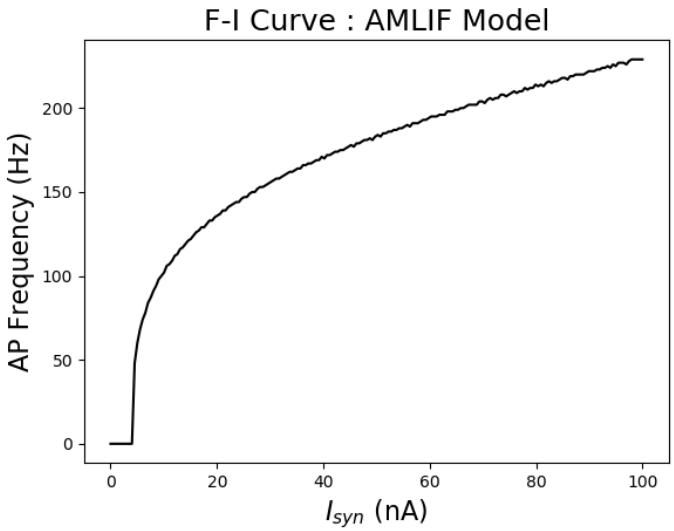


Figure 2.5: The F-I gain curve for the Adaptive M-current Leaky Integrate and Fire (AMLIF) model neuron.

2.4 Network Decision Making

The concepts described thus far in this Chapter provide for a mechanism to train artificial spiking neural networks in a biologically-inspired manner. The question now becomes how to interpret the output of the network, especially, how to know when the network has made a "decision", which could be a classification, detection, taking an action, etc. In the biological realm, this concept is very complicated. Decisions are made in the brain using billions of neurons, dozens of different architectures, many neurotransmitters, etc. There is no single answer to the question and it depends very much so on the context of the question. For example, it is well known that the Basal Ganglia plays a crucial role in making reward-based decisions [56]. The insect olfactory system, which can be considered an insect analog to the mammalian basal ganglia, has also been extensively studied and is useful for understanding one of the many mechanisms behind biological SNN reward-based decision making [15, 20, 49]. Given the complexity and diversity of these decision making mechanisms, we will take a biologically-inspired, yet highly abstracted approach to network decision making that will be sufficient for proof-of-concept purposes. Forming a more strict artificial analog to existing

biological decision making mechanisms, perhaps modelled after the basal ganglia, is set for future work.

2.4.1 Output Encoding

Output encoding is the process of interpreting the output of a neuron/s in the network. The Diehl-Cook model does this using max-output encoding, where each neuron is assigned a class label based on the inputs in which it is most active, post-training [16]. Another model feeds the output of its network into a SVM which is trained to output the final network decision [40]. Finally, another model uses first-spike encoding, where the first output neuron to spike after receiving input stimuli is used as the network's decision [55].

In order to determine the most appropriate output encoding method for our model, we must first understand some of the activity dynamics of the networks when being shown a set of input stimuli. Unlike the output from a traditional ANN, the output from an SNN can be highly probabilistic. The Poisson input data encoding scheme that is used in this work ensures that the activity state of the network is never completely constant. Because of this, the first-spike encoding scheme (which was heavily investigated in this work) is not *always* a good output encoding format. The first neuron that spikes is not necessarily the neuron that is most sensitive to desired features, rather, it may fire simply as a result of the noise in the network. In order to demonstrate the probabilistic nature of the neuron firing, a simple network architecture was created consisting of two layers, an input layer and an output layer. Using Poisson encoding with a firing rate of 2 kHz, a single image of a vertical bar on a 8x8 pixel grid was input into the network. The total average firing rate of all 16 neurons in the output layer (each using the AMLIF model previously described) was tracked over a duration of 10 seconds. This firing rate is shown in Figure 2.6. The average firing rate was estimated using an exponential moving average. The key takeaway from this figure is that the network firing rate over the duration of exposure to a single input encoded using the Poisson encoding scheme is non-constant. There are times when the firing rate rapidly increases and rapidly decays. As a result of these probabilistic dynamics, first-spike encoding is not always suitable as an output encoding scheme, especially for noisy input, even though it has the benefit of being easily-interpreted. For input stimuli that is not

noisy however, this may not be a problem. When designing a network that is to use first spike encoding, it is also possible to overcome the noise-induced spike problem by making it sufficiently difficult to induce a spike in the decision neurons. This can be done by lowering the excitatory conductance of the presynaptic neuron group(s) that is(are) connected to the decision neurons such that for one of the decision neurons to spike, a large amount of repeated stimuli from the presynaptic neuron group(s) is required. This is the method used in this thesis.

The max-output encoding scheme of the Diehl-Cook model has many benefits when compared to other common output encoding schemes. One of these is that unlike the first-spike encoding scheme, information is integrated over time such that the probabilistic nature of the spiking is no longer a problem. Also, it does not require a separate classifier such as is the case for some other models [40]. The max-output encoding scheme as used in the Diehl-Cook model is designed specifically for the case of a fully unsupervised training scheme. The network is first trained on the entire training dataset using STDP. After training is complete, the training dataset is then shown to the network again and each neuron is assigned is classification label based on the images in which it was maximally sensitive. Because this thesis aims to develop a method for supervised learning, the max-output encoding scheme cannot be used as the output neurons are not assigned their respective classes until training is complete.

2.4.2 Supervised Intrinsic Short-term Plasticity

Under the neuromodulated STDP learning algorithm, the only time in which a neuron learns that it has made the correct decision is when it fires and happens to make the right choice. When the network is initialized with random weights, this correct choice is made randomly, as the neuron has not yet learned to fire for any specific stimuli. While the use of anti-STDP can aid in encouraging the correct neuron to fire on the correct stimulus (by decorrelating the other neuron's receptive field weights with said stimulus), this process can still take a long time if the number of output neurons is large; i.e. as the number of neurons (possible decisions) increases, the probability of randomly selecting the correct choice upon network initialization also increases.

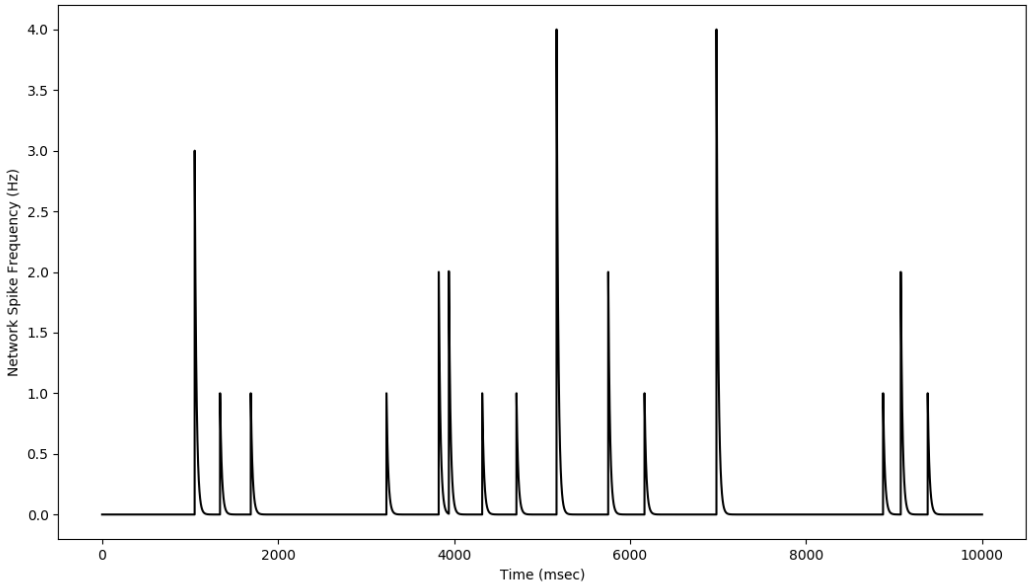


Figure 2.6: Average firing rate of all 16 output neurons estimated using an exponential moving average over a duration of 10 seconds. The network was shown a single image of a vertical bar on an 8x8 pixel grid. The firing rate is clearly not constant, demonstrating the probabilistic nature of spiking neuron firing activity.

In order to speed up the training process, a form of supervised learning is developed using a teaching signal that encourages the correct neuron to fire. This teaching signal is implemented by allowing the firing threshold of the neurons in the outer layer to raised and lowered temporarily through a process called supervised intrinsic short-term plasticity. When presenting each training input to the network, the threshold of the correct neuron/s is lowered by a small value ϵ. In doing so, it becomes easier for the correct neuron(s) to fire which helps to speed up the initial training process of the network. This also helps the decision neurons to develop weights sensitive to the correct features. After each input class is introduced to the network several times (the number of times depends on the learning rate), then this feature can be optionally turned off during training. It is important that each neuron in the output layer not only learn the correct stimuli to activate on, but also to

learn which stimuli to not activate on. With this in mind, it is important to select a value of ϵ that is not so large that only the correct neuron fires.

Chapter 3

Oriented Bar and MNIST Digit Classifcation

This chapter will introduce the process of designing and training spiking neural networks using the neuroscience-inspired processess discussed in Chapter 2 for two separate classifcation problems. The first problem is very simple, consisting of classifying bars in a small image as being either in a horizontal or vertical orientation. This problem was selected as it is easy to understand/interpret and demonstrates a good proof of concept for the methods discussed in Chapter 2.

The second example problem is for the MNIST handwritten digit classification problem [42]. This is a much more challenging tasks but has become a standard in the machine learning community. The network must learn to classify thousands of images of handwritten digits from 0 to 9. This problem is introduced in Section 3.2.

3.1 Example: Oriented Bar Classification

In order to demonstrate and prove the various concepts and principles described in Chapter 2, a toy problem consisting of learning repetitive features in noisy data was modelled in the Ganglion software package that was created for this project. The features in this dataset consist of bars set at different orientations and locations in a noisy image of size 8×8 pixels. The network will be tasked with solving problems related to extracting these bar features

from the noisy data and classifying new images as horizontal or vertical bars. Figure 3.1 shows the bar images that the network must learn to discriminate.

3.1.1 Training the Feature Detector

The first task is to train a network to detect the input data features. We will create a neuron group containing 36 excitatory AMLIF neurons arranged in a 6×6 grid shape. Each of these neurons receives spikes from the input data layer in a densely-connected fashion as shown in Figure 3.2. The input images are unit-normalized and encoded using Poisson encoding into spikes in the sensory layer. The feature layer has inhibitory connections to itself in order to enable both negative feedback and lateral inhibition.

The goal of the AMLIF layer is to learn a sparse representation of the input data, such that each neuron in the layer learns just one or two of the examples in the dataset. The layer is trained by presenting each image in the dataset to the network using the count encoding method, where training for each image is stopped when the cumulative output spike count

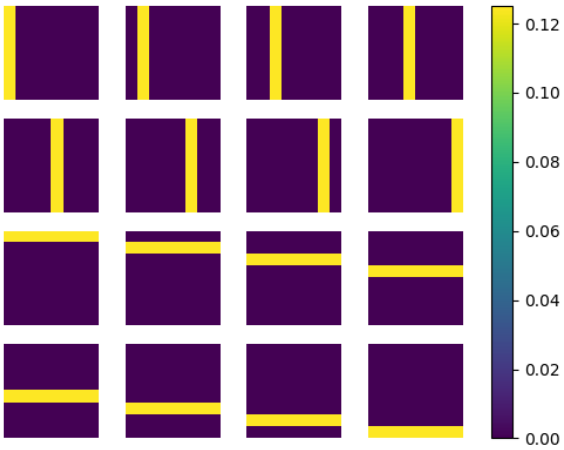


Figure 3.1: The images in the bars dataset. The network must learn to classify each image as either a horizontal or vertical bar.

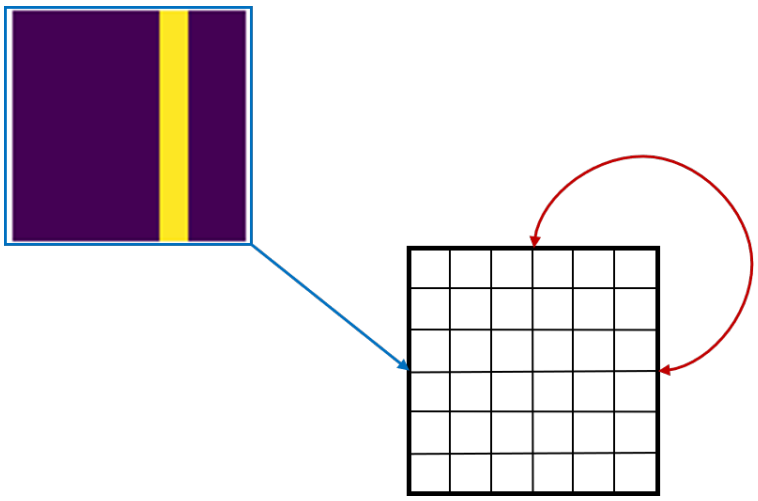


Figure 3.2: Neural network architecture for learning features of the bars dataset. Blue and red arrows represent excitatory inhibitory connections, respectively. The blue squares represent the receptive field of each neuron in the postsynaptic connection.

C_s reaches a certain value. During exposure of each image to the network, the layer learns using the standard online STDP protocol. Each image in the dataset is shown to the network 10 times for a total of 160 episodes (10 episodes for each of the 16 images). The images are alternated after each episode so that the network does not "forget" previously shown images. Every 3 ms during image exposure, Poisson noise is added to the image, with a spiking probability of 1% for each input neuron per time step (0.05 ms). This is to demonstrate the ability for STDP to filter out noise from the input. The inhibitory connections do not learn and are initialized with weight values of 10. The parameters used for the neuron, synapse, and training models are shown in Table 3.1, with non-default parameters in bold.

After all training episodes are complete, the weights of the receptive fields for each of neurons in the AMLIF layer can be viewed in order to observe which features of the input data that each neuron learned. Figure 3.3 shows these weights. As can be seen, 11 of the neurons each learned a single bar orientation/position, 6 learned two bar orientations/positions simultaneously, 2 learned more than 2 bar orientations/positions, and 17 did not learn any features. One neuron in particular learned a total of 6 bars, which actually presents a challenge for orientation classification as if that neuron spikes by itself, it is impossible to

Table 3.1: Parameters used for the feature learning stage of training the bars network. Non-default parameter values are shown in bold.

V_r	V_{thr}	V_E	V_I	C_m	τ_m	t_r	g_e	g_I	τ_C
-70 mV	-50 mV	0 mV	-75 mV	1 pF	10 ms	3 ms	0.2 nS	100 nS	5 ms
$\tau_{\text{syn,E}}$	$\tau_{\text{syn,I}}$	τ_a	τ_b	S_{ab}	S_{ba}	α	β	η	C_s
0.5 ms	0.5 ms	5 ms	5 ms	0.6	-0.3	1	1	0.001	2

determine if it detected a horizontal or vertical bar. Training the model with a lower learning rate fixes this, however it is kept in this example in order to demonstrate an important concept in neural networks, redundancy. Some of the neurons in the layer learned to detect the same bar orientation/position. In fact, correlating to the exposure count C_s of 2, each bar orientation/position has on average at least two neurons that learned each bar. This enables the classification layer (which is about to be discussed) to determine the orientation of the bar even when some neurons are sensitive to more than one bar orientation. Another point of interest is the fact that the neurons did not have any trouble adapting to the noise input into the network, which is a feature of STDP as discussed in Chapter 1.

3.1.2 Training the Classifier

Now that the feature detector has been trained, layers can be added to the network for performing classification decisions. As stated before, the goal of this network is to classify bar images as either horizontal or vertical bars, thus there are two network decisions. One idea is to simply add a single layer after the feature detection layer consisting of two neurons, one for each network decision as shown in Figure 3.4. The feature detector layer is connected to the decision layer in a densely connected fashion. The output neurons use the AMLIF neuron model just like those in the feature detector. Rather than using traditional STDP however, the synapses between the feature detection layer and the decision layer are trained using neuromodulated STDP as described in Chapters 1 and 2. The parameters used for neuromudulated STDP, which are defined in Chapter 2, are shown in Table 3.2. When the

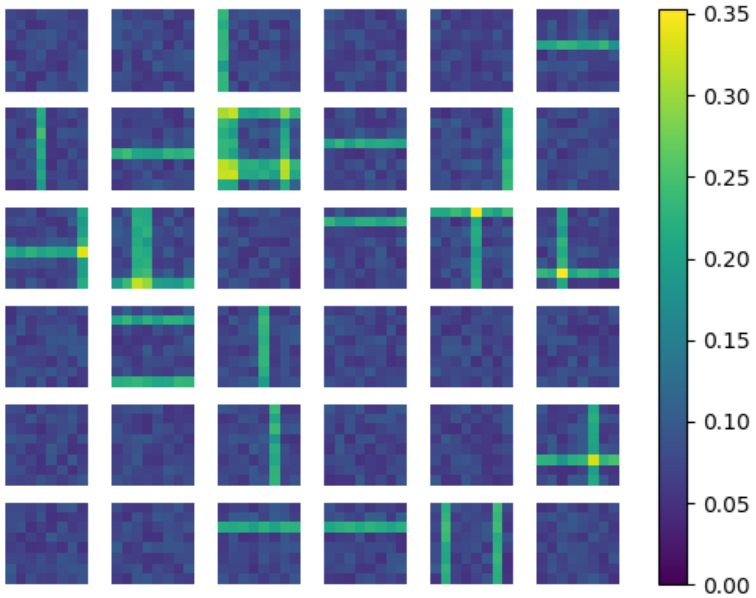


Figure 3.3: Weights learned by the feature detector layer when trained on the bars dataset using standard STDP

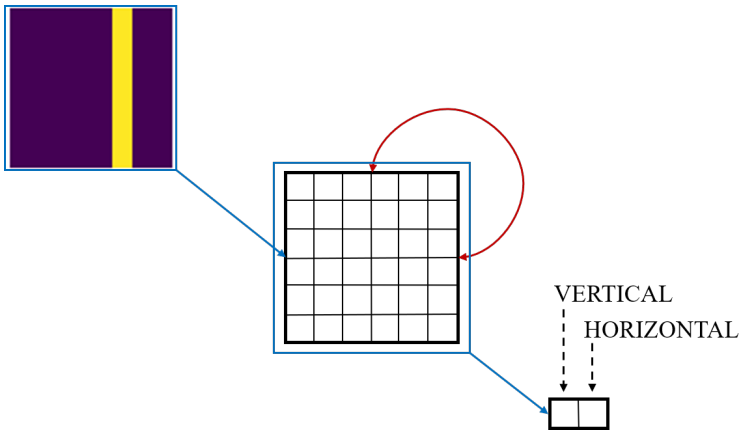


Figure 3.4: First iteration of the neural network architecture for bars dataset classification using pre-trained feature detector weights. Blue and red arrows represent excitatory inhibitory connections, respectively. The blue squares represent the receptive field of each neuron in the postsynaptic connection.

decision layer makes the correct decision, the standard STDP protocol is used to reinforce that decision. Contrarily, when the the network makes the wrong decision, anti-Hebbian learning is used to decorrelate the given stimuli with the wrong decision, to discourage that stimulus/decision pair to occur again in the future. The synapses between the input and the feature detector layer are not trained while training the classifier.

This classification accuracy during training was tracked using a moving average with a window size of 16. For the network as shown in Figure 3.4, the accuracy approached 100% after 58 episodes when trained using the parameters in Table 3.2 as seen in Figure 3.5 which

Table 3.2: Parameters used for training the neuromodulated STDP synapses for the classification learning stage of training the bars network. Non-default parameter values are shown in bold. Unless otherwise stated in this table, non-listed parameters use the values in Table 3.1

$\tau_{\text{syn,E}}$	$\tau_{\text{syn,I}}$	τ_a	τ_b	S^+_{ab}	S^-_{ab}	S^+_{ba}	S^-_{ba}	τ_{ab}	τ_{ba}	α	β	η
0.5 ms	0.5 ms	0.5 ms	0.5 ms	1	-1	-1	-1	10 ms	10 ms	1	1	0.001

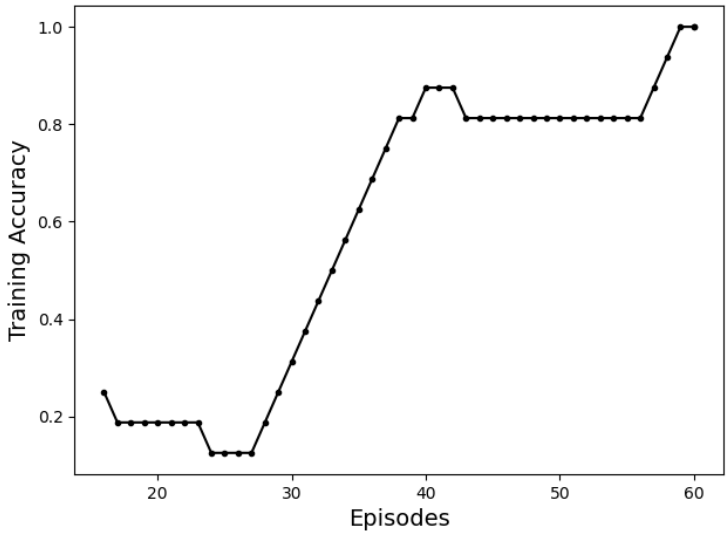


Figure 3.5: Accuracy of the network shown in Figure 3.4 when trained on the bars dataset using the parameters shown in Table 3.2. The accuracy is calculated using a moving average with a window size of 16.

shows the time course of the accuracy during training. One of the most influential parameters during training is the learning rate. As shown in Figure 3.6, just like in backpropagation neural networks, large learning rates lead to very rapid and unstable changes in network behavior, which can cause the network to overshoot its target and prevent it from learning a correct representation of the data.

In order to demonstrate that the algorithm is able to work on deeper networks, another network was designed with another layer placed between the feature and decision layer as shown in Figure 3.7. This extra layer is trained using the neuromodulated STDP method with same parameters as shown in Table 3.2. This network achieved a 100% accuracy after 42 episodes and maintained this accuracy up to the 60 episodes in which it was run.

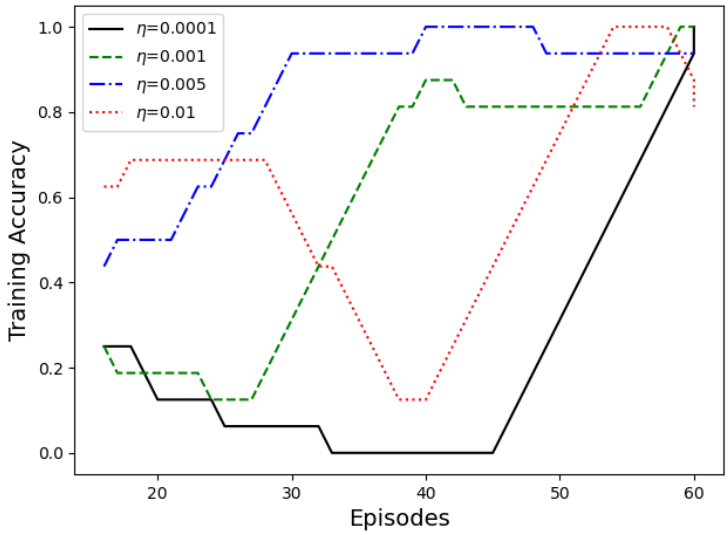


Figure 3.6: Accuracy of the network shown in Figure 3.4 when trained using different learning rates (η). Higher learning rates yield instability in the training. The accuracy is calculated using a moving average with a window size of 16.

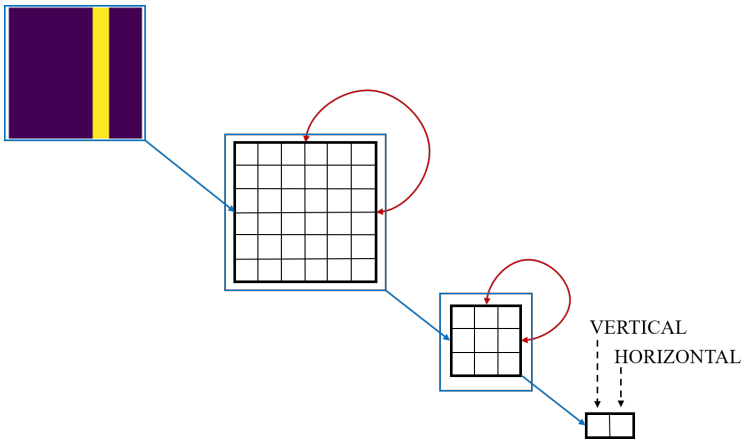


Figure 3.7: The same network as shown in Figure 3.4, with an additional layer between the feature and decision layer. Blue and red arrows represent excitatory and inhibitory connections, respectively. The blue squares represent the receptive field of each neuron in the postsynaptic connection.

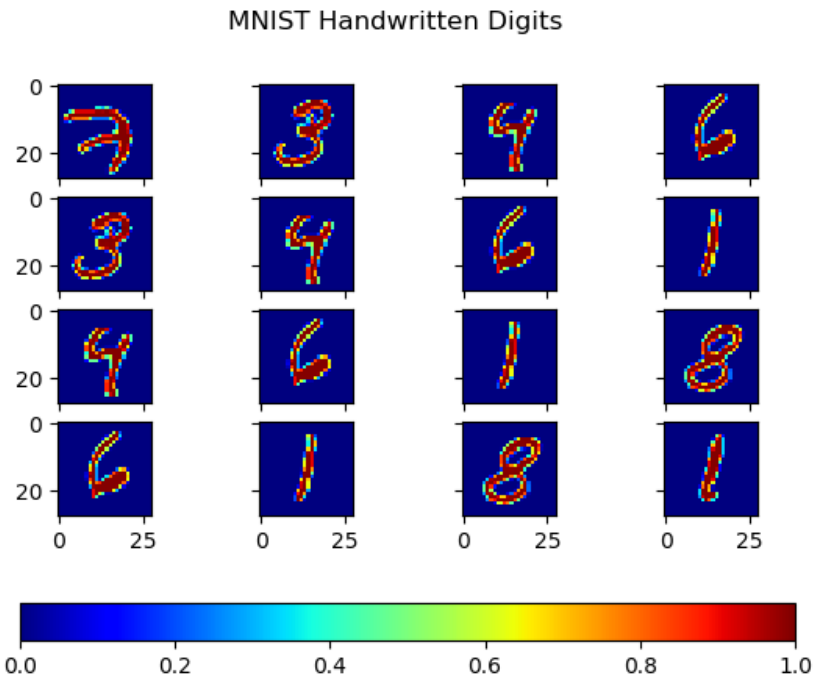


Figure 3.8: An example selection of handwritten digits from the MNIST dataset.

3.2 Example: MNIST Handwritten Digit Classification

The modified national institute of standards and technology (MNIST) database is one of the most influential datasets for machine learning in recent years [42]. The training set in MNIST contains 60000 images of handwritten digits (0 to 9) and the testing set likewise contains 10000 examples. Each of the digits is of size 20×20 pixels centered in a 28×28 pixel grid with a black background. A selection of digits from the database are shown in Figure 3.8.

3.2.1 Network Design

Input Encoding

Straight out of the dataset, the pixel values in these images have values ranging from 0 to 1, however, before presenting each image to the network, the pixel values are normalized to integral unity. This helps the input activity level to be roughly the same across the entire dataset. Without doing this, the network was shown to preferentially learn the digits with more active pixels such as 8 or 3, when compared to digits like 1. Because the computational speed of the feedforward current propagation is dependent on the size of the matrix dot product, smaller network inputs will run faster. For this reason, and also due to the fact that each MNIST image is padded with 4 pixels on each side, the images are cropped to only the center 20×20 pixels.

Inspired by the LGN neurons in the eye, a difference of Gaussian (DOG) filter is applied to each of the digits prior to normalization. This thins the edges out a bit to help them fit within the receptive fields of the feature detection neurons. The MNIST digits are also deskewed using an affine transformation calculated using the image moments. This helps to align the digits in the center of the frame and remove some of the skewing of these digits. Without deskewing, the network was only able to achieve an accuracy of 52%. Deskewing would likely not be as helpful if the network architecture was fully convolutional as such a network would have built-in spatial invariance. Figure 3.9 shows an example of how the cropping, normalization, DOG filtering, and deskewing pre-processing steps change an example input image from MNIST.

Architecture

The final architecture that was settled on is shown in Figure 3.10. The first layer is a simple sensory input layer of size 20×20, the same size as each of the input images. The second layer is a group of 30 separate excitatory locally connected neuron groups using the AMLIF model. Each of the neurons in these groups has a receptive field on the input group of size 6×6 (kernel size, k). The position of the receptive field of each consecutive neuron is spaced by 2 neurons (stride, s). The number of neurons in each of these groups, N_g is given by

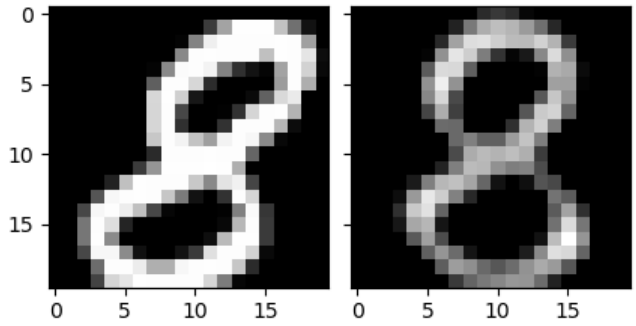


Figure 3.9: An example digit from the MNIST dataset before pre-processing (left) and after pre-processing using cropping, normalization, DOG filtering, and deskewing (right).

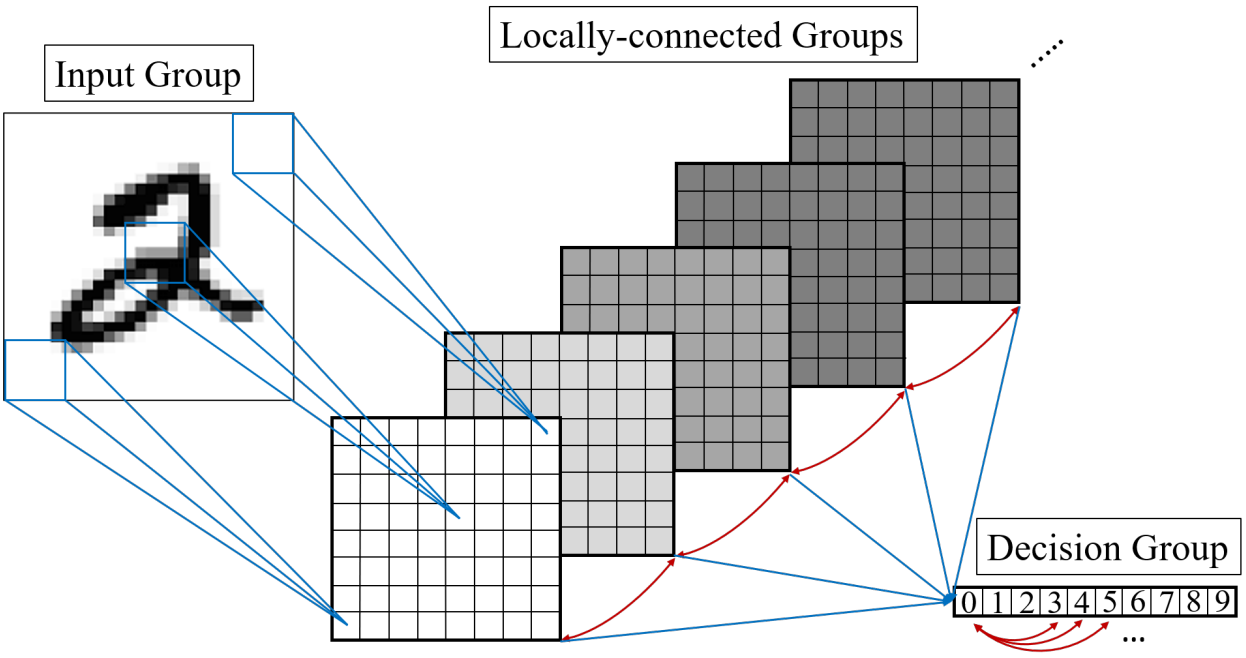


Figure 3.10: The architecture used for the MNIST classification problem. There are 30 locally-connected neuron groups. Blue connections indicate excitatory connections and red indicate inhibitory connections.

Equation 3.1, where N_i is the number of neurons in the input layer. With $k = 6$ and $N_i = 20$, the size of each of the locally-connected groups is 8×8. Equation 3.1 is the same formulation for calculating the output size of a convolutional layer in a traditional CNN, assuming zero padding. The second layer of the network is based loosely on the organization of the V1 part of the visual cortex as described in Section 1.2.5. Coming up with a true convolutional scheme may perform better in terms of classification accuracy, however it is not biologically plausible and is thus not considered in this project.

$$N_g = \frac{N_i - k}{s} \tag{3.1}$$

Many different network architectures were investigated prior to settling on this one. The number of locally-connected groups in the feature detection layer was ranged from just 5 up to 35, with values between 20 and 30 performing best on a limited subset of the testing dataset. With values between 20 and 30 working best, it was decided to settle on 30 in order to encourage the network to build in some feature redundancy in the training phase.

The kernel size and stride were also investigated prior to settling on the final values of $k = 6$ and $s = 2$. Kernel sizes ranging from 5 to 12 were tested, with values between 5 and 8 learning the most generalized features. Larger kernel sizes of 10 or higher did not perform as desired, with the neurons learning to detect more specialized features of specific digits rather than features that are general to all of the digits such as edges, which is what the V1 layer of the visual cortex tends to learn. Kernel sizes smaller than 5 do not work well, as the typical width of the edge features in the input data match or exceed a width of 5 pixels, which causes the neuron weights to learn incomplete features. The selection of the stride value is based primarily on the kernel size and the desired number of neurons in each locally connected group. For $k = 6$, a stride of 2 leads to a locally-connected group of 8×8.

Each neuron in each group in the second layer has a one-to-one lateral inhibition connection to each of the other groups in the second layer, which encourages each neuron to learn a different feature than neurons in the other groups that share the same receptive field. Without this lateral inhibition, all neurons with the same receptive field learn the same

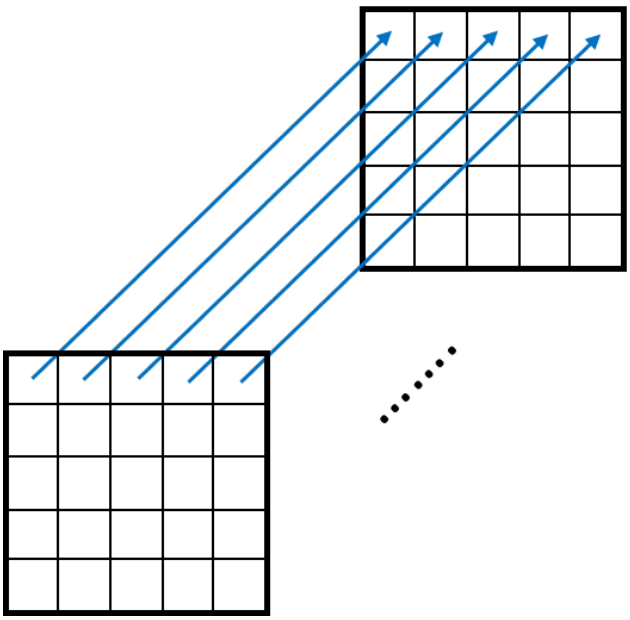


Figure 3.11: Lateral inhibition via 1 to 1 connectivity. Blue arrows indicate inhibitory connections.

features, which negates the purpose of having multiple neuron groups in layer 2. The one-to-one lateral inhibition connection scheme is demonstrated in Figure 3.11 for visualization purposes. Further, the use of lateral inhibition played a very large role in increasing the final classification accuracy, increasing it by over 100%.

The output layer is a group of excitatory STLIF neurons that each have an assigned digit class from 0 to 9. The STLIF neural group uses the methods described in Section 2.4.2 as a form of supervised learning. Using the STLIF neuron model did not result in an increase in final classification accuracy, however it did drastically speed up the training process as it encourages the decision neurons to make the correct decision earlier in the training process. Also, the use of redundant decision neurons (more than one decision neuron per digit class) did not have a measurable effect on the final classification accuracy.

Adding more layers between the feature detection layer and the decision layer did not result in a measurable increase in classification accuracy, but did result in a longer simulation time. It is suspected that if the input data was more complex, i.e. if the input images were not composed of just elementary edge features as the MNIST digits are, then adding more

layers modelled by the visual cortex would likely perform better. This not being the case, the final architecture does not have any layers between the feature detection and decision layers.

3.2.2 Training

Training the Feature Detection Layer

Like the bar orientation classification task demonstrated in Chapter 2, the synapses between the first and second layers of the network are trained using standard unsupervised STDP prior to training the decision layer. Rather than using spike count encoding, each of the 300 training images was shown to the network for a given encoding time t_{encoding}.

Because many of the digits in the dataset have overlapping features, a problem can occur that a neuron may become sensitive to a particular feature and then in the future become sensitive to another separate but similar feature. Over time, the result is that the neuron will tend to learn multiple overlapping features that reduces the specificity of that neuron. In biological networks in the brain, this problem can be overcome by "forgetting" the older feature that is no longer used very often. The concept of forgetting is still not understood in the brain, however some evidence points to LTP decay [29]. In order to simulate this in this network, we can use a concept called weight decay where every time the weights in the network are modified, the weights are multiplied by a weight decay value w_d that is slightly less than 1. This allows the neurons to slowly forget information that is no longer relevant to that neuron. Weight decay is also used in traditional backpropagation artificial neural networks, one of the very few overlaps between both fields.

The parameters used for training the feature layer are shown in Table 3.3.

Training the Decision Layer

Like the bar classification problem demonstrated in Chapter 2, the decision layer is trained using neuromodulated STDP. Unlike the bar classification problem however, positive rewards do not lead to standard STDP, rather they lead to standard Hebbian learning. Likewise, rather than anti-STDP as was used in the bar classification problem, anti-Hebbian learning

is used upon receipt of a negative reward. Using this scheme proved to reduce the time needed for training the network while keeping the final accuracy about the same.

The decision layer is initially trained on 1000 digits of the training set using $\epsilon = -10mV$. It is then trained on another 300 digits using $\epsilon = -5$mV, followed by another 300 digits at $\epsilon = -1$mV, and finally an additional 300 digits at $\epsilon = 0$mV. Each image is encoded using a Poisson encoding with a maximum firing rate of 1kHz, corresponding to a pixel value of 1, and input into the network until the decision layer spikes at least once (first spike encoding). As discussed in Chapter 2, there can be a potential problem in when using first spike encoding combined with Poisson encoding: the first decision neuron to fire may not be the neuron that is most sensitive to the underlying features of the input data. Also discussed in Chapter 2, this problem can be mitigated by making it sufficiently difficult to induce a spike in the decision neurons. This is the approach taken in this network. The excitatory conductance of the synapses between the feature detection layer and the decision layer is set to a very low value such that the decision neurons are forced to integrate many spikes from the feature layer in order to spike.

As with the bar classification problem, the synapses between the input sensory layer and the feature detection layer are not trained when training the decision layer. The many parameters used to train the decision layer are shown in Table 3.4.

3.2.3 Results

The weights of the receptive fields for each of the neurons in the feature layer were plotted after training and are shown in Figure 3.12. As can be seen, the network learned to identify various edges of different size and orientation. This is expected, considering that the digits are composed of edges. It is also interesting to note that each of the weights for neurons with the same receptive fields learn different features. This is important for several reasons: first, it encourages sparsity in the network which both reduces the activity of the network and makes it more computationally efficient and second, it expands the number of features that the network can detect for a given feature layer size. Rather than a single high activity feature dominating a given receptive field, neurons with the same receptive field learn to identify many features.

A zoomed in view of four of the neuron group receptive field weights is shown in Figure 3.13a. This figure serves to show better detail of the learned features and also to better demonstrate the use of lateral inhibition for neuron feature specificity. This figure can be compared to Figure 3.13b which shows the same 4 neuron groups when trained without using lateral inhibition. As can be seen in Figure 3.13b, each neuron with a given receptive field learns the same features when lateral inhibition is disabled, which negates the purpose of having multiple neurons with the same receptive field.

The network classification accuracy was calculated over the entire testing set, consisting of 10000 handwritten digits. The final classification accuracy was **62%**. Using a multi-spike encoding for the testing stage (as opposed to first-spike encoding) did not lead to a statistically significant change in classification accuracy. Figure 3.14 shows the confusion matrix for all digits in the testing set.

3.2.4 Discussion

It is important to note that the primary motivation for testing the network on the MNIST dataset was not to compete with other algorithms, rather, it is to act as a proof-of-concept for the methods described in Chapter 2. As is discussed in the Future Work section, better performance would almost certainly be achieved by doing a meticulous optimization on the many dozens of parameters guiding the dynamics of the network. The architecture itself could likely use some improvement, perhaps by modeling it to be more strictly reminscent of the organization of the human visual processing pathway, as discussed in Chapter 1.

Aside from the potential improvements, this network demonstrates many of the concepts discussed in Chapter 2. First, it demonstrates the use of unsupervised learning through standard STDP for learning statistically significant features in the input space. The features learned in the feature detection layer consist of edges at different orientations and scales, which represent elementary features of the input digits. When looking at the feature maps in Figure 3.13, it can be seen that that each neuron in the second layer was able to learn a different feature than other layer 2 neurons with the same receptive fields. Without lateral inhibition, the network is unable to efficiently encode useful information in the feature detection layer.

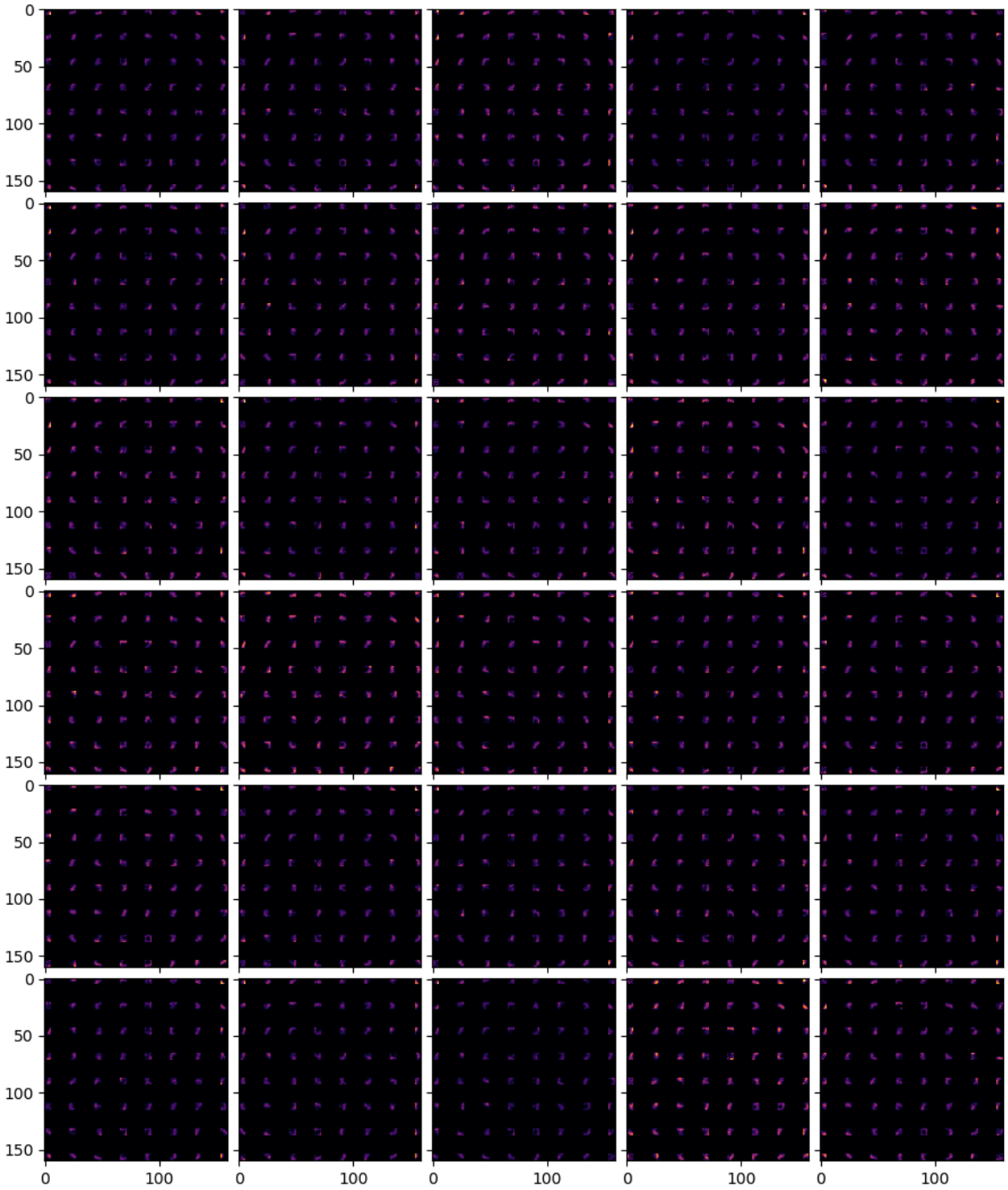
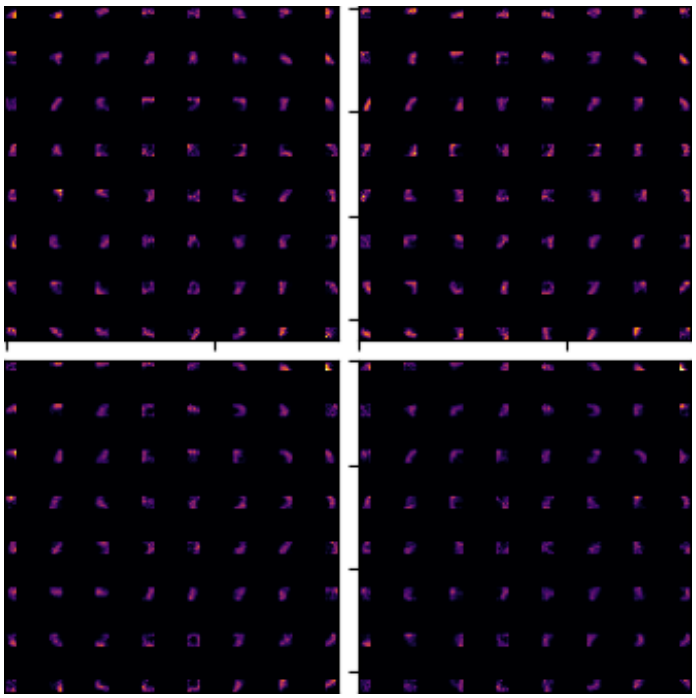
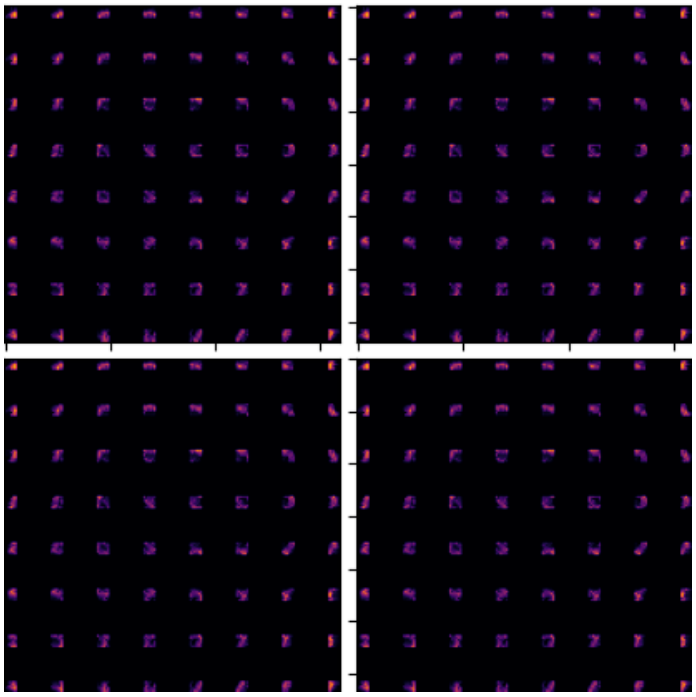


Figure 3.12: Weights of the receptive fields learned by the feature detection layer of the MNIST classification network. The features are composed of various edges at different scales and orientations.



(a) With Lateral Inhibition



(b) Without Lateral Inhibition

Figure 3.13: (top) Weights of the receptive fields learned by the feature detection layer of the MNIST classification network, zoomed in to just four neuron groups from Figure 3.12. (bottom) The weights of the same 4 groups as for the top figure but without using lateral inhibition during training.

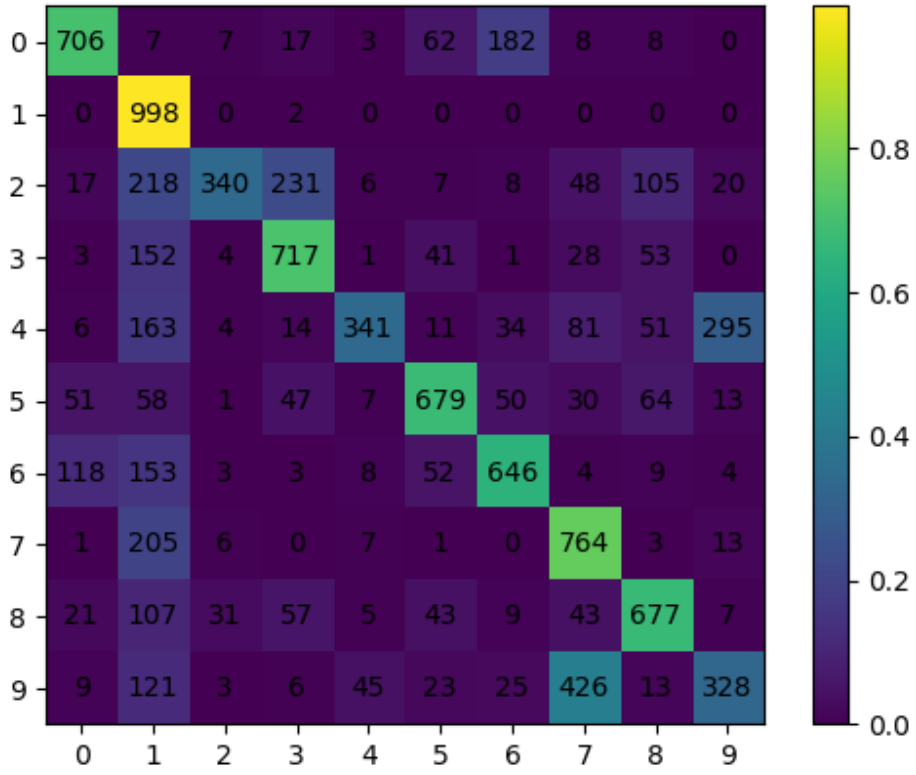


Figure 3.14: Confusion matrix for the algorithm evaluated over the entire testing dataset.

The output decision layer learns to translate the feature maps in the feature layer into digit representations using neuromodulated STDP. There are certainly some areas of improvement with regards to classification accuracy, however, it is important to emphasize that there is no form of gradient-based learning or backpropagated error signal in this network. The weights in the network are adjusted based solely on the relative timing of pre and post-synaptic spikes and neuromodulation. With this in mind, an accuracy of 62% is very reasonable, and at the very least validates further investigation of these topics.

When looking at the confusion matrix, it can be seen that the network performs better on some digits than others. Some of these are not too surprising, such as the confusion between 7 and 9, 0 and 6, and 4 and 9, which have many similar overlapping features. One of the more interesting sources of confusion in the network is for the digit 1. The true positive rate for the digit 1 is nearly 100%, however, it exhibits a high false positive rate as well (classifies other digits as a 1 quite often). When observing the digits in the training set, it can be seen that the digit 1 is very consistent in how it is written when compared to the other digits. As a result, the network develops quite strong weights for the 1 digit, which makes it more likely that other digits will excite the decision neuron assigned to the 1 digit.

Table 3.3: Relevant parameters used for the feature learning stage of training the bars network. Parameters that are not actively used in the simulation are omitted from this table.

Input Neuron Group Parameters

g_e	g_I								
1.5 nS	100 nS								

Feature Neuron Group Parameters

V_r	V_{thr}	V_E	V_I	C_m	τ_m	t_r	τ_C		
-70 mV	-50 mV	0 mV	-70 mV	1 pF	20 ms	3 ms	5 ms		

Input-Feature Synapse and Training Parameters

$\tau_{\text{syn,E}}$	$\tau_{\text{syn,I}}$	τ_a	τ_b	S_{ab}	S_{ba}	α	β	η	t_{encoding}
0.5 ms	0.5 ms	10 ms	10 ms	0.3	-0.4	1	1	0.05	10 ms

w_d	k	stride							
0.9	6×6	2							

Table 3.4: Relevant parameters used for the decision layer training stage for the MNIST network. Parameters that are not actively used in the simulation are omitted from this table.

Input Neuron Group Parameters

g_e	g_I								
1.5 nS	100 nS								

Feature Neuron Group Parameters

V_r	V_{thr}	V_E	V_I	C_m	τ_m	t_r	τ_C	g_e	g_I
-70 mV	-50 mV	0 mV	-70 mV	1 pF	20 ms	3 ms	5 ms	0.1 nS	100 nS

Decision Neuron Group Parameters

V_r	V_{thr}	V_E	V_I	C_m	τ_m	t_r	τ_C		
-70 mV	-50 mV	0 mV	-70 mV	1 pF	100 ms	3 ms	5 ms		

Feature-Decision Synapse and Training Parameters

$\tau_{\text{syn,E}}$	$\tau_{\text{syn,I}}$	τ_a	τ_b	S_{ab}^+	S_{ab}^-	S_{ba}^+	S_{ba}^-	τ_{ab}	τ_{ba}
0.5 ms	5 ms	5 ms	5 ms	1	-1	1	-1	10 ms	10 ms

η	C_s								
0.05	1								

Chapter 4

Conclusion

4.1 Discussion

Biological neural networks, especially in the human brain, are able to process information from thousands of sensors (hearing, vision, pressure, temperature, smelling, tasting, etc.) and use this information to make decisions. The human brain does this simultaneously while also performing complex cognitive tasks such as future decision planning, motor control of hundreds of muscles required to move and maintain essential bodily function, take part in language processing and communication, etc. Achieving the level of intelligence and adaptability that the human brain has achieved is arguably the gold standard in the pursuit of general artificial intelligence. Unlocking the methods in which the human brain achieves such tasks is daunting and cognitive neuroscientists have only so far scratched the surface. This being said, this thesis demonstrated that while the full picture of how learning occurs in the brain is by no means fully discovered, there is still enough knowledge presently to simulate basic cognitive tasks in the artificial intelligence space and demonstrate the viability of the current knowledge for application to the artificial intelligence space.

At its core, this thesis presented a neuroscience-inspired approach to training spiking neural networks and demonstrated this approach on a few simple machine learning tasks. While the results on the MNIST task do not reach the level of accuracy of many other methods used to train spiking networks, the results do indicate that the approach works and is based on a solid theoretical foundation. Spiking neural networks are incredibly complex,

temporally and spatially dynamic, and are notoriously difficult to train. Many prior works have been able to train spiking networks by converting standard artificial neural networks trained using backpropagation into spiking networks. Other prior work achieved success by combining spiking networks with traditional machine learning methods such as support vector machines. While these efforts have been successful and are certainly useful, the work in this thesis does not intend to establish a new state-of-the-art with regards to network performance, rather, it intends to demonstrate a *strictly* neuroscience-inspired approach to training and modelling these networks. As mentioned in the Future Work section, there is reason to believe that an exhaustive optimization of the many tunable hyperparameters in this methodology would result in superior network performance on the MNIST task. Doing this would take a long time and would be very tedious, however, it may reveal insight into how the many parameters in the models can affect network performance in general. The following list gives an overview of the most significant contributions that this thesis made to field of spiking neural networks:

- The design of a new neuron model based on the leaky integrate and fire model that includes the voltage-dependent M-current for spike rate adaptation. The M-current is modeled based on real experimental data and the neuron model demonstrates spike rate adaptation in tune with that observed in real pyramidal neurons, which is important for network activity homeostasis. This model is called the Adaptive M-Current Leaky Integrate and Fire (AMLIF) model.

- Developed a flexible, object-oriented spiking neural network simulation library for Python 3 called Ganglion. The library is designed to be extremely flexible and allows developers to rapidly design and simulate complex spiking neural network architectures using a variety of biologically-inspired methodologies. Included is a graphical visualization environment written in OpenGL for observing spiking networks as they are running.

- Demonstrated the use of synaptic time dependent plasticity for unsupervised training of networks to extract useful features from visual data.

- Demonstrated the use of neuromodulated synaptic time dependent plasticity for converting standard STDP into a variety of alternative forms of Hebbian/STDP learning (including standard Hebbian learning, anti-Hebbian learning, standard STDP, and anti-STDP), as is observed in biological neural networks. The viability of using neuromodulated STDP combined with standard unsupervised STDP for training spiking neural networks on standard machine learning classification tasks is also demonstrated.

4.2 Future Work

There are many opportunities for future work in this project. Research in spiking neural networks and neuromorphic computing is still in the early stages, so the potential list for future work in the subject is nearly endless. I myself intend to continue researching the topic and maintaining the Ganglion simulation library. Some possible areas of future work that directly tie to this project specifically, are as follows:

1. Fine-tune the many hyper-parameters introduced in Chapters 2 and 3. This would be a relatively daunting task as their are many different hyper-parameters and their interactions overlap in many cases. It is very likely that this step would result in better performance on the MNIST task and other tasks as well.

2. Modify the architecture used in the MNIST problem to more closely resemble the visual cortex. This would likely include modeling the lateral geniculate nucleus and V1, V2, and V3.

3. Explore other sensory input encodings. Chapter 3 used the simple Poisson encoding technique, however others such as temporal encoding may perform better given the use of STDP as the learning algorithm.

4. Modify the architecture of the decision-making layer to resemble that of the basal ganglia or the insect olfactory system.

5. Implement other differential equation solvers into the Ganglion software package to potentially improve simulation accuracy and software performance.

6. Explore the possibility of integrating the principles demonstrated in this thesis in neuromorphic hardware.

Bibliography

[1] Abbott, L. F. (1999). Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain research bulletin*, 50(5-6):303–304. 31

[2] Allen, N. M., Weckhuysen, S., Gorman, K., King, M. D., and Lerche, H. (2019). Genetic potassium channel-associated epilepsies: Clinical review of the kv family. *European Journal of Paediatric Neurology*. 14

[3] Baddeley, R., Abbott, L. F., Booth, M. C., Sengpiel, F., Freeman, T., Wakeman, E. A., and Rolls, E. T. (1997). Responses of neurons in primary and inferior temporal visual cortices to natural scenes. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 264(1389):1775–1783. 92

[4] Benda, J., H. A. V. M. (2003). A universal model for spike-frequency adaptation. *Neural Computation*, 15:2523–2564. 9, 13

[5] Benjamin Siddoway, H. H. and Xia., H. (2013). Molecular mechanisms of homeostatic synaptic downscaling. *Neuropharmacology*, 78:38–44. 14, 15

[6] Berridge, M. J., Lipp, P., and Bootman, M. D. (2000). The versatility and universality of calcium signalling. *Nature Reviews Molecular Cell Biology*, 1(1):11–21. 13

[7] Bi, G. Q., P. M. M. (2001). Synaptic modification by correlated activity: Hebb's postulate revisited. *Annual review of neuroscience*, 24:139–166. ix, 10, 12

[8] Bower, J. M. and Beeman, D. (2007). GENESIS (simulation environment). *Scholarpedia*, 2(3):1383. revision #137151. 31

[9] Brown, D. A., A. P. R. (1980). Muscarinic suppression of a novel voltage-sensitive k+ current in a vertebrate neurone. *Nature*, 283:673–676. 13, 14

[10] Brzosko, Z., Zannone, S., Schultz, W., Clopath, C., and Paulsen, O. (2017). Sequential neuromodulation of hebbian plasticity offers mechanism for effective reward-based navigation. *eLife*, 6:e27756. 19

[11] Carnevale, N. T. and Hines, M. L. (2006). *The NEURON book*. Cambridge University Press. 33

[12] Cavazzini, M., B. T. E. N. (2005). Ca2+ and synaptic plasticity. *Cell calcium*, 38:355–367. 9, 10

[13] Clements, J. D., Lester, R., Tong, G., Jahr, C. E., and Westbrook, G. L. (1992). The time course of glutamate in the synaptic cleft. *Science*, 258(5087):1498–1501. 36

[14] Clifton, N. E., Trent, S., Thomas, K. L., and Hall, J. (2019). Regulation and function of activity-dependent homer in synaptic plasticity. *Molecular neuropsychiatry*, 5(3):147–161. 15

[15] Delahunt, C. B., Riffell, J. A., and Kutz, J. N. (2018). Biological mechanisms for learning: a computational model of olfactory learning in the manduca sexta moth, with applications to neural nets. *Frontiers in computational neuroscience*, 12:102. 43

[16] Diehl, P. and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, 9:99. x, 26, 27, 44, 91

[17] Diehl, P. U., Zarrella, G., Cassidy, A., Pedroni, B. U., and Neftci, E. (2016). Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8. IEEE. 28

[18] Dingledine, R., B. K. B. D. T. S. F. (1999). The glutamate receptor ion channels. *Pharmacological reviews*, 51:7–62. 9

[19] Elston, G. N. (2003). Cortex, cognition and the cell: New insights into the pyramidal neuron and prefrontal function. *Cerebral Cortex*, 13:1124–1138. 3

[20] Faghihi, F., Moustafa, A. A., Heinrich, R., and Wörgötter, F. (2017). A computational model of conditioning inspired by drosophila olfactory system. *Neural Networks*, 87:96–108. 43

[21] Fong, M.-f., Newman, J. P., Potter, S. M., and Wenner, P. (2015). Upward synaptic scaling is dependent on neurotransmission rather than spiking. *Nature communications*, 6:6339. 16

[22] Furukawa H., Singh, S. K. M. R. G. E. (2005). Subunit arrangement and function in nmda receptors. *Nature*, 438:185–192. 9

[23] Gerstner, W., Kempter, R., Van Hemmen, J. L., and Wagner, H. (1996). A neuronal learning rule for sub-millisecond temporal coding. *Nature*, 383(6595):76–78. x, 33, 34, 35

[24] Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014). *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press. 33

[25] Gjorgjieva, J., Clopath, C., Audet, J., and Pfister, J.-P. (2011). A triplet spike-timing–dependent plasticity model generalizes the bienenstock–cooper–munro rule to higher-order spatiotemporal correlations. *Proceedings of the National Academy of Sciences*, 108(48):19383–19388. 93

[26] Glimcher, P. (2011-09-13). Understanding dopamine and reinforcement learning: The dopamine reward prediction error hypothesis. *Proceedings of the National Academy of Sciences of the United States of America*, 108. 17

[27] Grienberger, C. and Konnerth, A. (2012). Imaging calcium in neurons. *Neuron*, 73(5):862 – 885. 13

[28] Hangya, B., Ranade, S. P., Lorenc, M., and Kepecs, A. (2015). Central cholinergic neurons are rapidly recruited by reinforcement feedback. *Cell*, 162(5):1155–1168. 18, 19

[29] Hardt, O., Nader, K., and Wang, Y.-T. (2014). Glua2-dependent ampa receptor endocytosis and the decay of early and late long-term potentiation: possible mechanisms for forgetting of short-and long-term memories. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 369(1633):20130141. 61

[30] He, K., Huertas, M., Hong, S. Z., Tie, X., Hell, J. W., Shouval, H., and Kirkwood, A. (2015). Distinct eligibility traces for ltp and ltd in cortical synapses. *Neuron*, 88(3):528–538. viii, 18, 19

[31] Hebb, D. O. (1949). *Organization of Behavior*. 8

[32] Hodgkin, A. L. and Huxley, A. F. (1952a). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117:500–544. 8

[33] Hodgkin, A. L. and Huxley, A. F. (1952b). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544. 42

[34] Hoffman, D. A. and Johnston, D. (1999). Neuromodulation of dendritic action potentials. *Journal of Neurophysiology*, 81(1):408–411. PMID: 9914302. 20

[35] Hu, J.-H., Park, J. M., Park, S., Xiao, B., Dehoff, M. H., Kim, S., Hayashi, T., Schwarz, M. K., Huganir, R. L., Seeburg, P. H., Linden, D. J., and Worley, P. F. (2010). Homeostatic scaling requires group i mglur activation mediated by homer1a. *Neuron*, 68(6):1128 – 1142. 15, 16

[36] Hunsberger, E. and Eliasmith, C. (2015). Spiking deep networks with lif neurons. *arXiv preprint arXiv:1510.08829*. 28

[37] Ibata, K., Sun, Q., and Turrigiano, G. G. (2008). Rapid synaptic scaling induced by changes in postsynaptic firing. *Neuron*, 57(6):819–826. 16

[38] Inoue, Y., Udo, H., Inokuchi, K., and Sugiyama, H. (2007). Homer1a regulates the activity-induced remodeling of synaptic structures in cultured hippocampal neurons. *Neuroscience*, 150(4):841 – 852. 15

[39] Jorge, B. S., Campbell, C. M., Miller, A. R., Rutter, E. D., Gurnett, C. A., Vanoye, C. G., George, A. L., and Kearney, J. A. (2011). Voltage-gated potassium channel kcnv2 (kv8.2) contributes to epilepsy susceptibility. *Proceedings of the National Academy of Sciences*, 108(13):5443–5448. 14

[40] Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., and Masquelier, T. (2018). Stdp-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, 99:56–67. 28, 29, 44, 45

[41] Krogh, A. and Hertz, J. A. (1992). A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957. 37

[42] LeCun, Y., Cortes, C., and Burges, C. (2010). Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2. 26, 48, 56

[43] Letzkus, J. J., Kampa, B. M., and Stuart, G. J. (2006). Learning rules for spike timing-dependent plasticity depend on dendritic synapse location. *Journal of Neuroscience*, 26(41):10420–10429. 18

[44] Lines, A., Joshi, P., Liu, R., McCoy, S., Tse, J., Weng, Y.-H., and Davies, M. (2018). Loihi asynchronous neuromorphic research chip. In *2018 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 32–33. IEEE. 28

[45] Lisman, J., Schulman, H., and Cline, H. (2002). The molecular basis of camkii function in synaptic and behavioural memory. *Nature Reviews Neuroscience*, 3(3):175–190. 18

[46] Liu, D. and Yue, S. (2017). Fast unsupervised learning for visual pattern recognition using spike timing dependent plasticity. *Neurocomputing*, 249:212 – 224. 28

[47] Liu, Y., Cui, L., Schwarz, M. K., Dong, Y., and Schlüter, O. M. (2017). Adrenergic gate release for spike timing-dependent synaptic potentiation. *Neuron*, 93(2):394–408. 20

[48] Luscher, H.-R. and Larkum, M. E. (1998). Modeling action potential initiation and back-propagation in dendrites of cultured rat motoneurons. *Journal of Neurophysiology*, 80(2):715–729. 36

[49] MaBouDi, H., Shimazaki, H., Giurfa, M., and Chittka, L. (2017). Olfactory learning without the mushroom bodies: Spiking neural network models of the honeybee lateral antennal lobe tract reveal its capacities in odour memory tasks of varied complexities. *PLoS computational biology*, 13(6). 43

[50] Markram, H., L. J. F. M. and Sakmann, B. (1997). Regulation of synaptic efficacy by conincidence of postsynaptic aps and epsps. *Science*, 275:213–215. 8, 9

[51] Mather, M., Clewett, D., Sakaki, M., and Harley, C. W. (2016). Norepinephrine ignites local hotspots of neuronal excitation: How arousal amplifies selectivity in perception and memory. *Behavioral and Brain Sciences*, 39. 20

[52] Matias, S., Lottem, E., Dugue, G. P., and Mainen, Z. F. (2017). Activity patterns of serotonin neurons underlying cognitive flexibility. *Elife*, 6:e20552. 21

[53] Merolla, P., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., Jackson, B. L., Imam, N., Guo, C., Nakamura, Y., Brezzo, B., Vo, I., Esser, S. K., Appuswamy, R., Taba, B., Amir, A., Flickner, M. D., Risk, W. P., Manohar, R., and Modha, D. S. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345:668–673. 28

[54] Mishra, R. K., Kim, S., Guzman, S. J., and Jonas, P. (2016). Symmetric spike timing-dependent plasticity at ca3–ca3 synapses optimizes storage and recall in autoassociative networks. *Nature communications*, 7(1):1–11. 20

[55] Mozafari, M., Kheradpisheh, S. R., Masquelier, T., Nowzari-Dalini, A., and Ganjtabesh, M. (2018). First-spike-based visual categorization using reward-modulated stdp. *IEEE transactions on neural networks and learning systems*, 29(12):6178–6190. 28, 29, 44

[56] O'Reilly, R. C., Munakata, Y., Frank, M. J., Hazy, T. E., and Contributors (2012). *Computational Cognitive Neuroscience*. Wiki Book, 1st Edition, URL: http://ccnbook.colorado.edu. ix, 3, 8, 10, 13, 22, 25, 43, 91

[57] Pak, D. T. S. and Sheng, M. (2003). Targeted protein degradation and synapse remodeling by an inducible protein kinase. *Science*, 302(5649):1368–1373. 15

[58] Paoletti, P., N. J. (2007). Nmda receptor subunits: function and pharmacology. *Current opinion in pharmacology*, 7:39–47. 9

[59] Pfister, J.-P. and Gerstner, W. (2006). Triplets of spikes in a model of spike timing-dependent plasticity. *Journal of Neuroscience*, 26(38):9673–9682. 93

[60] Pignatelli, M. and Bonci, A. (2015). Role of dopamine neurons in reward and aversion: a synaptic plasticity perspective. *Neuron*, 86(5):1145–1157. 17

[61] Potjans, W., Diesmann, M., and Morrison, A. (2011). An imperfect dopaminergic error signal can drive temporal-difference learning. *PLoS computational biology*, 7(5). 17

[62] Rossier, M. F. (2016). T-type calcium channel: a privileged gate for calcium entry and control of adrenal steroidogenesis. *Frontiers in endocrinology*, 7:43. 15

[63] Ruan, H., Saur, T., and Yao, W.-D. (2014). Dopamine-enabled anti-hebbian timing-dependent plasticity in prefrontal circuitry. *Frontiers in neural circuits*, 8:38. 18

[64] Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., and Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, 11:682. 28

[65] Schultz, W. (2016). Dopamine reward prediction error coding. *Dialogues in clinical neuroscience*, 18(1):23. 17

[66] Seeburg, D. P. and Sheng, M. (2008). Activity-induced polo-like kinase 2 is required for homeostatic plasticity of hippocampal neurons during epileptiform activity. *Journal of Neuroscience*, 28(26):6583–6591. 15

[67] Sjöström, J. and Gerstner, W. (2010). Spike-timing dependent plasticity. *Scholarpedia*, 5(2):1362. revision #184913. 36

[68] Skou, J. C. (1957). The influence of some cations on an adenosine triphosphatase from peripheral nerves. *Biochimica et Biophysica Acta*, 23:394–401. 4

[69] Soares-Cunha, C., Coimbra, B., Sousa, N., and Rodrigues, A. J. (2016). Reappraising striatal d1-and d2-neurons in reward and aversion. *Neuroscience & Biobehavioral Reviews*, 68:370–386. 17

[70] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press. 17

[71] Turrigiano, G. G. and Nelson, S. B. (2000). Hebb and homeostasis in neuronal plasticity. *Current Opinion in Neurobiology*, 10(3):358 – 364. 14

[72] Yu, S. P., O'Malley, D., and Adams, P. R. (1994). Regulation of m current by intracellular calcium in bullfrog sympathetic ganglion neurons. *Journal of Neuroscience*, 14(6):3487–3499. ix, x, 13, 14, 41

[73] Zhou, Y. and Danbolt, N. C. (1996). Glutamate as a neurotransmitter in the healthy brain. *Journal of Neural Transmission*, 121:799–917. 4

Appendices

A The Ganglion Spiking Neural Network Simulation Package

This section intends to introduce an overview of the Ganglion spiking neural network simulation package, written in Python 3.6. This section does not intend to be a full documentation of the software. This decision was made because the library is still in its early development stages and if the library was documented in this thesis, it would rapidly become outdated. Instead, it is simply a short overview of the motivations behind its development, some of its main features as it currently stands, and a basic tutorial on creating, running, and saving a simple neural network. Ganglion, along with all of the scripts used in this thesis, can be accessed on Github at https://github.com/jghawaly/Ganglion.

A.1 Motivation

While many spiking neural network simulators are available, this project required a very high level of flexibility with regards to neuron models, network architecture, learning algorithms, hardware implementation, etc. For these reasons, and also because writing such a system from the ground up is a great way to become intimately familiar with how these networks work, the decision to write a new SNN simulation library was made. From a software development standpoint, the following design choices were made prior to designing the library:

1. Object Oriented Programming (OOP): Neuron models, synapse models, architecture definitions, and process pipelines will be written in an OOP paradigm with heavy use of multi-level object inheritance. This makes implementation of new neuron models and learning algorithms to be done with minimal effort.

2. Flexible network design and architecture: Ability to easily define complex network structure and connectivity schemes. Allow multiple neuron models to be used simultaneously in single network architectures.

3. Disconnected run-order and network architecture: The order of spike propagation is defined independently of the network's structure, i.e., a neuron group in the 2nd layer

can run before groups in layer 1, in fact, there is no real concept of a "layer" in this library, rather, "layers" are enabled by defining the run order in a layered pattern. This also allows the developer to easily disable certain groups of neurons while allowing them to still be part of the architecture.

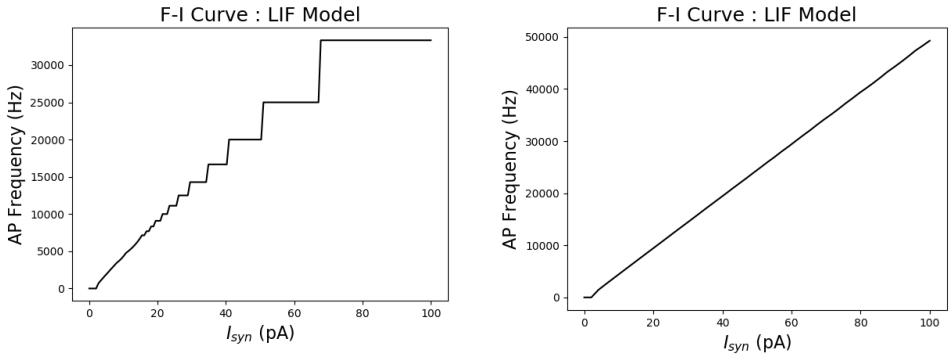
4. Maximal use of vectorized spike propagation: Maximum utilization of the Numpy library for vectorized calculations is done to speed up the simulations. This will also make it easier to transition some of the calculations to the GPU (such as with PyTorch) in future versions of the library if such a route proves to be useful. It is important to note that full vectorization of the network simulations is not possible due to the many conditionals required, an example being the determination of neuron spike times (spike if $V_m > V_{\text{thr}}$).

5. All code is written in Python with the majority of calculations performed using the Numpy library. When using Anaconda, the Numpy library uses statically compiled Fortran routines for the majority of its elementary linear algebra routines, giving it a speed comparable to that of statically-compiled languages. Python is currently the most popular language for machine learning and data science and has a massive plethora of libraries for applications in these fields such as Numpy, Scipy, Tensorflow, OpenCV, etc.

A.2 Solving Differential Equations

As described in Chapter 2, the neuron models, synapse models, and learning models are all described in terms of time-dependent differential equations. The simplest way to solve these differential equations is to use Euler's method, which is also described in Chapter 2. In order to perform Euler's method, the user must define a time step Δt. The Ganglion library uses a class called the TimeKeeper which keeps track of the current simulation time. When running a simulation, the TimeKeeper class is utilized via the TimeKeeperIterator class, which is an interable version of the TimeKeeper class. The accuracy of Euler's method increases at the time step Δt decreases. Likewise, the time it takes to run a simulation increases as Δt decreases. As such, it is the user's responsibility to choose a value for Δt that achieves the

right balance between computational speed and accuracy. In general, the level of fidelity required is dependent on the neuron's excitability (and thus its firing rate). As the firing rate increases, the Δt value needs to be smaller in order for the linear slope approximation to be accurate, as higher excitability means that the neuron's membrane potential will increase more rapidly. The F-I for a leaky integrate and fire neuron with a time step of 0.01 ms (left) and 0.001 ms (right) is shown in Figure B. At higher input currents, the firing rate estimate begins to fail for the lower time step. Luckily, the more biologically-plausible models such as the AMLIF described in Chapter 2 do not easily achieve such high firing rates due to intrinsic plasticity, such that a very small Δt value is not needed. In general, Δt is set between 0.01 and 0.5 for most simulations. The computational speed of the simulations may increase if different, perhaps more accurate, differential equation solvers are implemented such as the Runge-Kutta method, however this was not explored in this thesis and is left for future work.



(a) $\Delta t = 0.01$ms **(b)** $\Delta t = 0.001$ms

Figure 1: F-I curves for the leaky integrate and fire neuron with Euler time step sizes of 0.01 ms (left) and 0.001 ms (right), respectively.

A.3 Neurons

Each neuron model in Ganglion is defined as a new class that must at the very least inherit from the abstract class, NeuralGroup. Every new neuron model must override three functions: *pre_update()*, *update()*, and *post_update()*. The function *pre_update()* is called prior to updating the neuron's membrane potential and spiking state which is performed on calling *update()*. The function *post_update()* is called after the call to *update()* is complete,

which can be used to perform any clean up needed after updating the neuron parameters. The current code for the NeuralGroup class is shown in Listing A.3 to demonstrate the structure of a neuron model.

Listing 1: NeuralGroup source code.

```
class NeuralGroup:
    """
    This class is a base template containing only items that are common among most neuron models. It
    should be overriden, and does not run on its own.
    """
    def __init__(self, n_type: int, num: int, name: str, viz_layer: int, tki: TimeKeeperIterator,
                 field_shape=None, viz_layer_pos=(0,0)):
        self.name = name  # the name of this neuron
        self.tki = tki  # the TimeKeeperIterator instance that is shared amongst the entire network

        self.n_type = n_type  # type of neurons in this group (excitatory or inhibitory)
        self.n_num = num  # number of neurons in this group
        self.shape = (self.n_num,)  # shape/geometry of this neural group
        self.tracked_vars = []  # variables to be tracked throughout the course of evaluation
        self.viz_layer = viz_layer  # layer in the network for display in the GUI
        self.viz_layer_pos = viz_layer_pos  # position of the neural group in the layer for GUI display

        # define the virtual shape of the neural group
        if field_shape is None:
            self.field_shape = (self.shape[0],1)
        else:
            self.field_shape = field_shape

    def pre_update(self, inputs):
        """
        This is called prior to updating the state of the neurons. Inputs would generally be an array of
        synaptic currents

        This method should be overridden
        """
        return None

    def update(self, inputs):
        """
        Update the state of the neurons. Inputs would generally be an array of synaptic currents. This
        should perform all necessary operation required to update self.spiked

        This method should be overridden
        """
        return None

    def post_update(self):
        """
        This is called after updating the state of the neurons

        This method should be overridden
        """
        return None

    def run(self, inputs):
        """
```

```
        This executes the chain of pre_update, update, and post_update commands

        This method should be overridden
        """
        self.pre_update(None)
        self.update(None)
        return self.post_update()

    def reset(self):
        """
        This resets the neuron to its resting state

        This method should be overridden
        """
        return None
```

The code for defining the integrate and fire model is shown in Listing A.3 as an example for how to follow the NeuralGroup abstract class template.

```
class IFNeuralGroup(NeuralGroup):
    """
    This class defines a group of Integrate and Fire Neurons
    """
    def __init__(self, n_type: int, num: int, name: str, viz_layer: int, tki: TimeKeeperIterator,
                 params: IFParams, field_shape=None, viz_layer_pos=(0,0)):
        super().__init__(n_type, num, name, viz_layer, tki, field_shape=field_shape,
                         viz_layer_pos=viz_layer_pos)
        # custom parameters
        self.params = params
        # refractory period for these neurons
        self.refractory_period = np.full(self.shape, params.refractory_period)
        # holds boolean array of WHETHER OR NOT a spike occured in the last call to run()
        # (Note: NOT THE LAST TIME STEP)
        self.spiked = np.zeros(self.shape, dtype=np.int)
        # holds the NUMBER OF spikes that occured in the last evaluated time window
        self.spike_count = np.zeros(self.shape, dtype=np.int)
        # holds the TIMES OF last spike for each neuron
        self.last_spike_time = np.zeros(self.shape, dtype=np.float)
        # holds the time at which the spike count array was last updated
        self.last_spike_count_update = 0.0

        # construct reversal potential matrix
        self.v_rev = np.full(self.shape, params.vrev_e if self.n_type == excitatory else params.vrev_i,
                             dtype=np.float)
        self.vrev_i = np.full(self.shape, params.vrev_i)
        self.vrev_e = np.full(self.shape, params.vrev_e)

        # construct gbar matrix
        self.gbar = np.full(self.shape, params.gbar_e if self.n_type == excitatory else params.gbar_i,
                            dtype=np.float)

        # Parameters from Brette and Gerstner (2005).
        self.v_r = np.full(self.shape, params.v_r)  # rest potential
        self.v_m = np.full(self.shape, params.v_m)  # membrane potential
        self.v_spike = np.full(self.shape, params.v_spike)  # spike potential
        self.v_thr = np.full(self.shape, params.v_thr)  # spike threshold potential
        self.c_m = np.full(self.shape, params.c_m)  # membrane capacitance

        # parameter tracks
        self.v_m_track = []
```

```
42          self.v_thr_track = []
43          self.isyn_track = []
44          self.spike_track = []
45
46          # group behavior
47          self.wta = params.force_wta
48
49      def reset(self):
50          self.v_m = self.v_r.copy()
51
52      def not_in_refractory(self):
53          """
54          return mask of neurons that ARE NOT in refractory period
55          """
56          ir = np.ones(self.shape, dtype=np.int)
57          # set zeros for neurons in the refractory period
58          ir[np.where((self.tki.tick_time() - self.last_spike_time) < self.refractory_period)] = 0
59          # if last_spike_time is 0.0, then the neuron has never fired yet, so we want to exclude this
60          ir[np.where(self.last_spike_time == 0.0)] = 1
61
62          return ir
63
64      def pre_update(self, i_syn):
65          # if we are at a new time step since evaluating the neurons, then clear the spike count matrices
66          if self.last_spike_count_update != self.tki.tick_time():
67              # self.spike_count = np.zeros(self.shape, dtype=np.int)
68              self.spike_count.fill(0)
69
70          self.spiked = np.zeros(self.shape, dtype=np.int)
71
72      def update(self, i_syn):
73          # mask of neurons not in refractory period
74          refrac = self.not_in_refractory()
75
76          # calculate change in membrane potential for neurons not in refractory period
77          dvm = self.tki.dt() * (i_syn / self.c_m) * refrac
78          # print(dvm)
79          self.v_m += dvm
80
81          # find indices of neurons that have fired
82          self.spiked = np.where(self.v_m >= self.v_thr)
83
84      def track_vars(self, i_syn):
85          if "v_m" in self.tracked_vars:
86              # this is a copy of the membrane potential matrix
87              output = self.v_m.copy()
88              # change the output to the spike voltage for each neuron that fired. Note: this does
89              # not affect the actual v_m array, just a copy of it, because
90              # in this neuron model, the voltage of a spike does not really have any specific meaning,
91              #  rather, it is the time of the spikes that matter
92              output[self.spiked] = self.v_spike[self.spiked]
93              self.v_m_track.append(output.copy())
94          if "i_syn" in self.tracked_vars:
95              self.isyn_track.append(i_syn)
96          if "spike" in self.tracked_vars:
97              self.spike_track.append(self.spike_count.copy())
98          if "v_thr" in self.tracked_vars:
99              self.v_thr_track.append(self.v_thr.copy())
100
101     def post_update(self, i_syn):
102         # add a new spike to the spike count for each neuron that fired
```

```
        self.spike_count[self.spiked] += 1
        # update the time at which the spike count array was modified
        self.last_spike_count_update = self.tki.tick_time()
        # modify the last-spike-time for each neuron that fired
        self.last_spike_time[self.spiked] = self.tki.tick_time()

        # Correct for possible overshoot in V_m update resulting from too large of a Euler time step
        hp = np.where(self.v_m < self.vrev_i)
        self.v_m[hp] = self.vrev_i[hp]

        # change the actual membrane voltage to the resting potential for each neuron that fired
        self.v_m[self.spiked] = self.v_r[self.spiked]

    def run(self, i_syn):
        """
        Update the state of the neurons
        """
        self.pre_update(i_syn)
        self.update(i_syn)
        self.post_update(i_syn)
        self.track_vars(i_syn)

        return self.spike_count
```

The integrate and fire model is the base class for nearly all of the other neuron models defined in Ganglion. In order to demonstrate how easy it is to define a new neuron model that inherits from the IFNeuralGroup class, Listing A.3 shows the leaky integrate and fire class source code, which builds upon the integrate and fire model by adding a potassium ion leak channel. The only method that needs to be overridden is the *update()* method.

Listing 2: LIFNeuralGroup source code.

```
class LIFNeuralGroup(IFNeuralGroup):
    """
    This class defines a group of Leaky Integrate and Fire Neurons
    """
    def __init__(self, n_type: int, num: int, name: str, viz_layer: int, tki: TimeKeeperIterator,
                 params: LIFParams, field_shape=None, forced_wta=None, viz_layer_pos=(0,0)):
        super().__init__(n_type, num, name, viz_layer, tki, params=params, field_shape=field_shape,
                         viz_layer_pos=viz_layer_pos)
        # Parameters from Brette and Gerstner (2005).
        self.tao_m = np.full(self.shape, params.tao_m)  # membrane time constant

    def update(self, i_syn):
        # mask of neurons not in refractory period
        refrac = self.not_in_refractory()

        # calculate change in membrane potential for neurons not in refractory period
        dvm = self.tki.dt() * (-1*(self.v_m - self.v_r) / self.tao_m + i_syn / self.c_m) * refrac
        # print(dvm)
        self.v_m += dvm

        # find indices of neurons that have fired
        self.spiked = np.where(self.v_m >= self.v_thr)
```

There are currently ten neuron models implemented in Ganglion. Of these ten models, three are completely custom and the rest are either common models or inspired by previous work on the subject. These models are briefly described below:

- **Integrate and Fire** (IFNeuralGroup): Simple integrate and fire model. This model integrates inputs, has a fixed firing threshold, and does not contain and ion leakage channels.

- **Leaky Integrate and Fire** (LIFNeuralGroup): Inherits from the IFNeuralGroup class. Includes a non-voltage dependent K^+ ion leakage channel that decays the neuron's membrane potential expoentially according to a given time constant.

- **Exponential Leaky Integrate and Fire** (ExLIFNeuralGroup): Inherits from LIFNeuralGroup. As observed in biological neurons, the membrane potential of the ExLIF neuron increases exponentially as the membrane potential becomes very close the firing threshold.

- **Adaptive Exponential Leaky Integrate and Fire** (AdEx): Inherits from ExLIFNeuralGroup. The AdEx model includes an adaptation leakage current that increases as the neuron's firing rate increases. The adaptation current makes it more and more difficuly for the neuron to fire as its firing rate increases. This is a form of spike frequency adaptation and is the model of choice for the Emergent simulation software [56].

- **Floating Threshold Leaky Integrate and Fire** (FTLIFNeuralGroup): Inherits from LIFNeuralGroup: Rather than having a fixed firing threshold, the firing threshold V_{thr} is dynamic, increasing by a small factor every time the neuron fires. The firing threshold decays exponentially according to a given time constant back to its standard value. This is a form of spike frequency adaptation and is inspired by the model used in the Diehl-Cook network [16].

- **Homeostatic Leaky Integrate and Fire** (HSLIFNeuralGroup): Inherits from LIFNeuralGroup. It has been shown that neurons in the feline brain generally fire at a

rate of roughly 3 to 4Hz [3]. Like the FTLIFNeuralGroup model, the firing threshold is dynamic in this model. Every time the neuron fires/does not fire, the firing threshold is increased/decreased by a small value proportional to the difference between the current firing rate and the desired firing rate set by the user. This *encourages* (as opposed to forces) the neuron to maintain an average firing rate defined by the user. This helps neurons that have not fired in a long time to play a role in the network while also making sure that neurons cannot become to active. This is a very computationally-efficient model.

- **Adaptive M-current Leaky Integrate and Fire** (AMLIFNeuralGroup): Inherits from LIFNeuralGroup. This custom model includes an experimental fit of the Ca^{2+} and voltage-dependent M-current which enables spike-frequency adaptation, as described in Chapters 1 and 2. This model is comparable in concept to the Hodgkin-Huxley model and fits the shape of F-I curves of real pyramidal neurons very well.

- **Homeostatic Adaptive M-current Leaky Integrate and Fire** (HSAMLIFNeural-Group): Inherits from AMLIFNeuralGroup. This model uses the AMLIFNeuralGroup but also includes the dynamic threshold as used in the HSLIFNeuralGroup model.

- **Supervised Threshold Leaky Integrate and Fire** (STLIFNeuralGroup): Inherits from AMLIFNeuralGroup. This custom group is described in Chapter 3 and is used for neural groups that are used for decision making. During training, the firing threshold of the neuron/s with the correct label is decreased and is increased for the other neurons. This encourages (but does not force) the network to select the correct choice. This model should only be used in training.

Each NeuralGroup is initialized with a unique name, number of neurons, the shape of the neural group, a set of neuron parameters, and if the GUI will be used, then the visual layer and position in the GUI. An example of initializing an excitatory LIFNeuralGroup with 9 neurons organized in a 3×3 square grid, a leakage time constant of 12 ms, and using a Euler's method time step of 0.1ms is shown in Listing A.3.

Listing 3: Creating an LIFNeuralGroup.

```
# Set neuron parameters
p = LIFParams()
p.tao_m = 12 * msec

# initialize a TimeKeeperIterator with dt=0.1ms
tki = TimeKeeperIterator(0.1*msec)

# create a LIF neural group
g1 = LIFNeuralGroup(n_type=excitatory, num=9, name="g1", viz_layer=1, tki=tki, params=p, field_shape=(3,3))
```

A.4 Synapses

Like the neuron models, the synapse models are defined in an OOP paradigm, beginning with a base class (BaseSynapticGroup) that defines all of the basic methods for generating weights, calculating synaptic current, propagating spikes, etc. All SynapticGroup models are initialized with a presynaptic and postsynaptic NeuralGroup. The Hebbian learning models are implemented in the SynapticGroup models. Currently, four synapse models besides the BaseSynapticGroup (which can be used on its own if learning is not needed) are implemented in Ganglion, which are briefly described below. All models use the alpha synapse for spike propagation.

- **PairSTDPSynapticGroup**: This model uses an online implementation of spike pair synaptic time dependent plasticity as described in Chapter 2.

- **TripletSTDPSynapticGroup**: Instead of using only two spikes to perform STDP, this model uses three spikes in such a way to capture not only the timing dependency of learning but also the spike rate dependency. This model has been shown to resemble the Bienenstock-Cooper-Munro learning rule for certain conditions [25] and can be better fitted to experimental data in the visual cortex [59]. Although it is slightly more biologically plausible than spike pair-based STDP, it is also a bit more difficult to tune.

- **DASTDPSynapticGroup**: This is the neuromodulated version of pair-based STDP as defined in Chapter 2. This model allows the user to define how the presence of a neuromodulator affects the LTP and LTD components of the STDP protocol.

- **InhibitorySynapticGroup**: This model does not perform any learning and equivalent to the BaseSynapticGroup except that the calculated current going into the postsynaptic neuron group from the presynaptic group is always negative. In Chapters 1 and 2, the concept of lateral inhibition was discussed. Lateral inhibition in biological networks is mediated by the neurons called inhibitory interneurons. The excitatory group is connected to the inhibitory interneuron group, which is then connected back to the excitatory group (in the case of a negative feedback loop) or other excitatory neurons in the same layer (lateral inhibition). This is one way in which lateral inhibition can be implemented in Ganglion, however, rather than creating an entirely separate inhibitory neuron group, it is more or less computationally equivalent to allow the excitatory group to also have inhibitory connections to other excitatory groups. That is the purpose of the InhibitorySynapticGroup model.

A.5 Neural Networks

Currently, a NeuralNetwork class is initialized with a list of NeuralGroup objects and an instance of the TimeKeeperIterator class, along with a variety of other optional parameters. The NeuralNetwork class is responsible for handling the creation of synaptic groups between neuron groups, running the network according to a given group order, saving and loading weights, setting network trainability, injecting neuromodulators into the networks, and many other network-level functions. The best way to understand how it works is to demonstrate an example.

Building on the code in Listing A.3, we can create another neuron group. Let's say we want to create an AMLIF model with default parameters. This is shown on line 13 of Listing A.5. We can then define a new NeuralNetwork containing these two neurons, shown on line 17 of Listing A.5. In order to connect the LIFNeuralGroup to the AMLIFNeuralGroup, we must create a synaptic group between them. Line 24 shows the process of connecting these two groups together using densely connected synapses with pair-based STDP enabled, using a custom learning rate of 0.01. As can be seen, the neuron groups are referenced in the *fully_connect()* method by their given names not by their variable names. The options for synapse type (*s_type*) are "base", "pair", "triplet", "da", and "inhib" for each of the model

previously discussed. We can also set the minimum and maximum weights to be randomly generated for the synapses, 0.1 and 0.9 in this case, respectively.

In order to run the network, we must iterate over the TimeKeeperIterator instance that we created on line 6 as shown on line 27. For demonstration purposes, we will just input random noise into the LIFNeuralGroup using Poisson encoding with a maximum firing rate of 10 Hz as shown on line 29. We can then perform the spike propagation by using the *run_order()* method of the NeuralNetwork which takes a list or tuple of neuron names in the order that we wish for them to run, in this case we run the LIFNeuralGroup before the AMLIFNeuralGroup as shown on line 32. We can then stop the simulation at 100 ms as shown on lines 35 and 36.

The weights learned in the network via STDP can be saved as shown on line 39, which shows them being saved to the current working directory. They can then be reloaded into another network as shown on lines 42 and 43. The weights are saved in compact binary files using the ".npy" extension.

Listing 4: Creating a NeuralNetwork.

```
1  # Set neuron parameters
2  p = LIFParams()
3  p.tao_m = 12 * msec
4
5  # initialize a TimeKeeperIterator with dt=0.1ms
6  tki = TimeKeeperIterator(0.1*msec)
7
8  # create a LIF neural group
9  g1 = LIFNeuralGroup(n_type=excitatory, num=9, name="g1", viz_layer=1, tki=tki, params=p,
10                      field_shape=(3,3))
11
12 # create an AMLIF neural Group
13 g2 = AMLIFNeuralGroup(n_type=excitatory, num=36, name="g2", viz_layer=2, tki=tki, params=AMLIFParams(),
14                        field_shape=(6,6))
15
16 # create a new NeuralNetwork with these two groups
17 nn = NeuralNetwork(groups=[g1, g2], name='my_network', tki=tki)
18
19 # create some custom learning parameters
20 stdp_p = PairSTDPParams()
21 stdp_p.lr = 0.01
22
23 # connect g1 to g2
24 nn.fully_connect("g1", "g2", trainable=True, stdp_params=stdp_p, minw=0.1, maxw=0.9, s_type='pair')
25
26 # iterate over TimeKeeperIterator
27 for step in tki:
28     # inject random values into g1 using Poisson encoding with a maximum firing rate of 10Hz
```

```
29      g1.run(poisson_train(np.random.random(g1.n_num), tki.dt(), 10))
30
31      # run g1 then g2
32      nn.run_order(("g1", "g2"))
33
34      # stop at 100 ms
35      if tki.tick_time() >= 100 * msec:
36          break
37
38  # save weights
39  nn.save_w(".")
40
41  # create another neural network, copied from the first one
42  nn2 = NeuralNetwork(groups=[g1, g2], name='my_network_copy', tki=tki)
43  nn2.load_w(".")
```

A.6 Network Visualization

It is very useful to be able to visualize the network while it is running. This allows the user to see which neurons are firing and how the membrane potential of each neuron evolves over time. Because the networks can be quite complex, it is useful to be able to do this in 3D space, thus a GUI was created using OpenGL via the pyglet Python library. The GUI is currently in its earliest stages but is at the point to where it is useful for the purposes described above. The positioning of each neural group in the GUI is defined by the *viz_layer* and *viz_layer_pos* parameters in the NeuralGroup initialization.

The left, right, up, and down arrow keys are used to rotate the network in space. The W, A, S, and D keys are used to pan the network in space. The scrollwheel on the mouse can be used to zoom in and out of the network. By default, the neurons are colored grey and turn yellow when they spike. In order to visualize the membrane potential, the user can press the V key. The closer the membrane potential is to the firing threshold, the brighter the red color of the neuron. Likewise, if the neuron has a membrane potential near rest, it will be black. Neuron spikes are still colored yellow. In order to go back to the plain spike view, the user can press P.

Vita

James Ghawaly Jr. was born and raised outside of New Orleans, LA. After high school, James moved to Knoxville, TN to study Nuclear Engineering at the University of Tennessee Knoxville. After graduation, he continued on at UTK to pursue a Ph.D. in Nuclear Engineering, where his dissertation research is on the development of a neural network for radiation anomaly detection. Given his interest in the field of machine learning, specifically with regards to spiking neural networks, neuromorphic computing, and computational cognitive neuroscience, James is also pursuing a M.Sc. degree in Computer Engineering. After graduation, James plans to continue research and development in the field of machine learning, especially with regards to its application to radiation detection, identification, and localization.