University of Colorado Boulder

National Institute of Standards and Technology

Doctoral Dissertation

Neuromorphic Intelligence

Bio-Inspired Algorithms for Hardware-Native Neural Computing

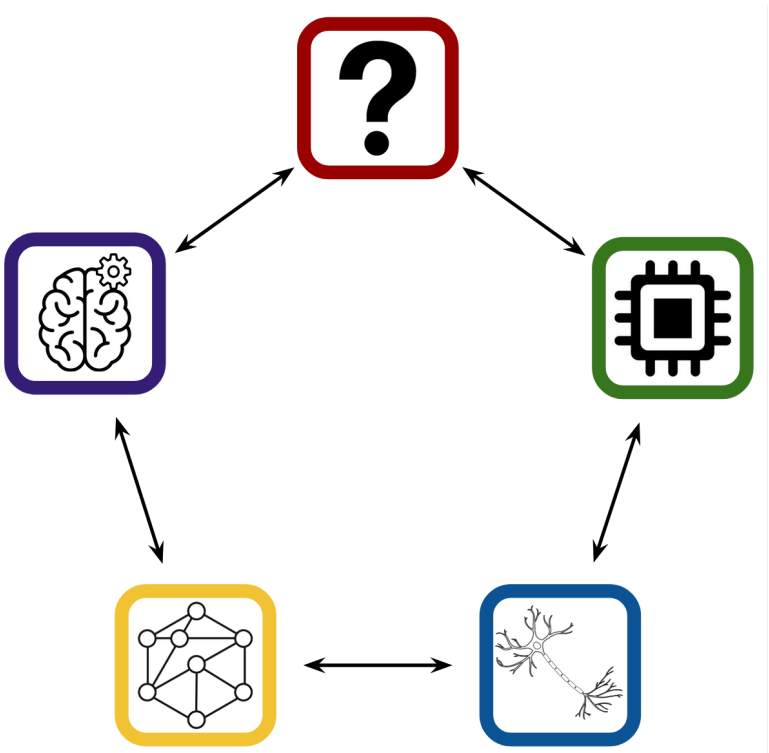
Author: Ryan O'Loughlin

CU Supervisor:	Bradley Hayes
CU Reviewer:	Nikolaus Correll
CU Reviewer:	Sriram Sankaranarayanan
NIST Supervisor:	Adam McCaughan
NIST Supervisor:	Sonia Buckley

Completed work toward the fulfillment of a
Doctoral Degree in Computer Science as assessed by
The University of Colorado Boulder *and*
The National Institute of Standards and Technology

March 31, 2026

Abstract: The brain is efficient, and modern AI is not. This mismatch is a matter of design. The brain utilizes analog computations distributed across sophisticated organic network topologies, whereas modern AI implements the backpropagation algorithm on digital GPUs supported by Von Neumann architectures. There are a growing number of emerging hardware that implement brain-like networks and operations, but there is not yet a clear neuromorphic algorithm to match the backpropagation-level performance required for impactful intelligence. This dissertation explores a number of novel and known candidates to support intelligence in neuromorphic hardware, and culminates in a bio-inspired learning algorithm that consistently performs within 1% of backpropagation using only hardware-friendly operations. This result may inform advancements in machine learning, specialized hardware design, and even neuroscience.



To my parents, who never wavered in their belief in my education.

Acknowledgments

I thank my NIST advisors, Adam McCaughan and Sonia Buckley, who seemed to possess a perfect understanding of how to best cultivate my scientific potential.
I thank Bradley Hayes at CU for going out on a limb to accept transfer applicant into his well-organized lab.

I thank my committee, including Nikolaus Correll and Sriram Sankaranarayanan in addition to my above-named advisors, for their accommodation of my timeline and their valuable feedback.

I thank the MGD group at NIST, especially Bakhrom Oripov, Nick Skuda, Andrew Dienstfrey, Adam McCaughan, and Sonia Buckley, for the many meetings, discussions, and support that were integral to this work.

I thank the neuromorphic study group at NIST. To be a scientist, one must read papers, and there is no better way to do so than in group.

I thank the SOENs project for giving me my start at NIST, and in particular my officemates, Bryce Primavera, Abhi Saxena, Ravin Chowdhury, and Saeed Khan.

I thank the TREM team at NIST for allowing me to participate in such a successful case of technological development. Although that work is not represented in this dissertation, my involvement with the TREM project was nevertheless integral to my development as a scientist.

I thank the Quantum Nanophotonics group at NIST for so readily welcoming a nonstandard background into their physics and engineering group. Their engagement with my (sometimes lengthy) updates was invaluable to my cultivation as a scientist and a presenter. I especially thank Rich Mirin, Marty Stevens, and Sae Woo Nam for allowing me to be a part of their exemplary scientific community.

I thank the many friends and collaborators from my Master's program in The Netherlands, who were essential in my early development as a researcher, Marleen Schippers,

Jelmer Borst, Thomas Tiotto, Guillaume Pourcel, Steven Abreu, Jesper Kuiper, Leander van Boven, Madhur Boran, Aurélien Adriaenssens, Oscar Tanis, Abe Brandsma, Ivo Steegstra, Chandan Seedhara, and Ludi van Leeuwen.

I thank the National Institute of Standards and Technology for funding this work and providing me with a scientific home.

I thank the University of Colorado Boulder for affording my work the opportunity to earn a PhD. I further thank the CU Department of Computer Science, and its key organizer, Rajshree Shrestha, for facilitating this collaboration across institutions, to my great benefit.

I especially thank my family for believing in my education and supporting the long journey that has lead to this doctoral dissertation.

I thank my wife, Anna, who so often dissolved my academic stress with her kindness and support.

Contents

1

An Introduction to Neuromorphic Computing

How does natural intelligence arise in the brain? We know that biological systems comprise billions of interconnected elements whose locally realized functions are asynchronous, analog, distributed, and massively parallel. We also know that natural intelligence is realized at a fraction of the energy cost of artificial intelligence (AI), where large numbers of digital circuits execute discrete operations in lockstep to implement even simple functions. Neuromorphic computing applies neural computing principles to dedicated brain-inspired hardware, enabling rich computation and dramatic efficiency gains in emerging systems for artificial intelligence.

This introduction will motivate the pursuit of neuromorphic computing, provide an overview its many approaches, and contextualize the contributions of this thesis within the field at large. For general background, we rely heavily on more comprehensive treatments such as [1–4], while framing this introductory material to prepare the reader for the original works of this dissertation [5–8].

We begin by introducing and motivating neuromorphic computing in Sec. 1.1 according to its intersection with **machine learning**, **hardware design**, and **neuroscience**. We then define the three fundamental features of any neuromorphic system as **neuromorphic functions** (Sec. 1.2) organized into a **neuromorphic network** (Sec. 1.3) that may be driven toward intelligent computations according to some algorithm for **neuromorphic learning** (Sec. 2.4). In our problem statement (Sec. 1.5), rather than asking how intelligence arises in the brain, we instead ask *how intelligence might arise in brain-inspired systems*. In Sec. 1.6, we introduce our contributions toward solving this problem, namely *Bio-Inspired Algorithms for Hardware Native Neural Computing*.

1.1 Why Neuromorphic?

Neuromorphic computing stands to advance artificial intelligence in terms of speed, energy efficiency, footprint, adaptability, and fault-tolerance [3]; all of which may further permit access to novel forms of neuro-inspired artificial intelligence. These advantages are enabled by a number of properties that distinguish neuromorphic computing from conventional computing, most notably those of neuro-inspired functions interconnected by organic topologies, driven to learn by bio-plausible mechanisms, and implemented on specialized neuromorphic hardware.

The rise of modern computing has scaled Turing-complete machines [9] in the form of von Neumann architectures (VNAs) [10] on silicon transistors according to Moore's law [11,12]. This trajectory has enabled the steady amassing of arbitrary compute and has been applied to a staggering diversity of real-world problems, producing unprecedented value to the human condition at large. It has therefore been subject to unprecedented investment. On the other hand, natural (human) intelligence has enjoyed no such abounding support and has rather been constricted by millennia of evolutionary pressure, pulled taught by two competing evolutionary advantages, intelligence and efficiency. Natural systems thus tend toward a balance of intelligence and efficiency [13], while artificial systems have instead realized supernatural intelligence [1] at the asking price of supernatural inefficiency [14]. Neuromorphic computing seeks to recouple intelligence with the real world by requiring that computational primitives match hardware primitives such that a circuit, like a neuron, is not an abstraction but rather intrinsically computational (see Fig. 1.1).

By exploiting physical properties of devices to perform the sorts of operations that coalesce into intelligence in the human brain, neuromorphic systems have begun to show substantial gains in efficiency for the kinds of information processing associated with cognition [15–20]. However, brain-like efficiency does not guarantee brain-like intelligence. A circuit might closely replicate the generic function of a neuron, but a brain is comprised of billions of neurons (with thousands of subtypes) and trillions of topologically intricate synaptic connections, subject to the dynamical symphony of human senses immersed in the interactive environment that forms a lived experience. Neuromorphic computing makes no claims to pursue such levels of sophistication, but rather targets the implementation of neural computing principles on dedicated hardware to advance artificial intelligence. Embedded within this objective are three lenses through which to view and motivate

[1] Intelligence here means the advantages accommodated by modern computing generally, including artificial intelligence.

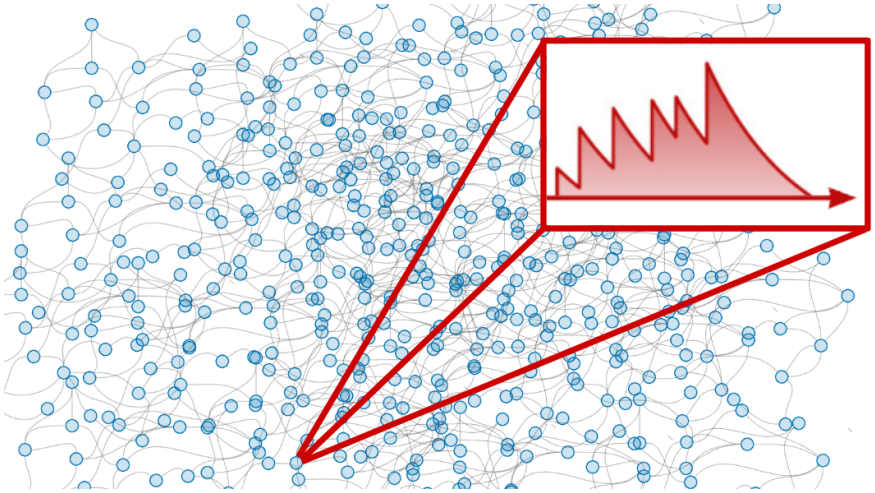


Figure 1.1: A simplified depiction of networked elements responsible for local computations. Each element might compute its own unique function according to its own intrinsic physical properties. We define a neuromorphic system as a network of neuro-inspired functions whose parameters are subject to some form of bio-inspired learning over time. This definition is captured above, with the understanding that the function and network are subject to intelligent adaptations over time.

neuromorphic computing, *machine learning*, *hardware design*, *and neuroscience*. These perspectives help constrain the problem space of intelligence to more tractable goals.

1.1.1 Machine Learning

Abstractly, an intelligent system would like to learn some mapping $\mathcal{F}$ of an input u to an output $\hat{y}$

$$\mathcal{F} : u \mapsto \hat{y}, \tag{1.1}$$

where this input-output relationship might involve a number of modalities across an arbitrary window of time T. The application space, then, might serve any of the following relationships over whatever interval of time

$$\mathcal{F} : \begin{bmatrix} \text{image} \\ \text{sound} \\ \text{signal} \\ \text{visual} \\ \text{text} \\ \text{state} \end{bmatrix}_{0:T} \mapsto \begin{bmatrix} \text{image} \\ \text{sound} \\ \text{signal} \\ \text{visual} \\ \text{text} \\ \text{action} \end{bmatrix}_{0:T.} \tag{1.2}$$

Mapping one or many of the above input elements to the above output elements should be self-explanatory in terms of utility, defining arbitrary application space. The greater question is how to learn such mappings?

We thus introduce a general (and commonplace) notation for defining any neuromorphic system as a state machine that learns to map some input u to an output $\hat{y}$ as governed by state transitions

$$x_{t+1} = f(x_t, u_t; \Theta) \tag{1.3}$$

where x_{t+1} is the state of the system at time $t+1$ as produced by the set of functions $f(\cdot)$ (usually a network of functions) that transition the system according to the the previous system state and input at time t, given the set of all learnable parameters Θ. The output of the system is given by the readout map $g(\cdot)$ such that

$$\hat{y} = g(x). \tag{1.4}$$

While there are some unsupervised methods that may learn useful mappings $\mathcal{F}$ naturally, neuromorphic computing (and AI generally) tends toward some form of task supervision in order to enforce the sorts of useful things a system might learn. Performance on some intelligent task can be defined as

$$\mathcal{L}(\hat{y}, y; \Theta), \tag{1.5}$$

where $\mathcal{L}$ is a loss function, such as L2-norm, and y is some target output value, such as a correct label given some input image u. Supervised methods allow a system to be guided iteratively toward some desired mapping by algorithmically adjusting learnable parameters Θ. We examine neuromorphic algorithms and loss minimization via gradient descent in greater detail in Sec. 2.4. Suffice it to say here that these common machine learning framings of desirable input-output mappings being realized through learnable parameters within a network of functions all translate directly to neuromorphic computing, and therefore the goals and possibilities of ML are considerably united with those of neuromorphic computing.

1.1.2 Hardware Design

As aforementioned, dedicated circuitry that exploits physical hardware properties for native computations and communication offers considerable advantages over traditional Von Neumann architectures. The degree to which a specialized hardware may leverage these principles over traditional methods varies and thus neuromorphic hardware is usually parsed into three categories accordingly: *digital*, *analog*, and *mixed signal*.

1.1.2.1 Digital Hardware

Moore's law [11] has seen unparalleled resource attention toward the advancement of increasingly compact transistor-based digital computing devices. While we have already introduced some motivation for dedicated analog circuits in neuromorphic computing, it is reasonable to expect that leveraging the existing infrastructure for laying out digital hardware toward specialized neuro-inspired digital devices is a path toward tangible near-term neuromorphic results. Indeed, some of the most scalable and accessible neuromorphic systems are based on digital transistors with specialized systems for low-bit, parallelisable communication [15–18]. By limiting communication overhead to sparse low-bit spiking events and communicating according to address event representation (AER), where specialized routing networks serve up spiking events according to neuron addresses in a graph, neuromorphic benefits can already be realized without needing to adopt novel computing substrates. AER avoids the von Neumann bottlenecke by communicating asynchronously, promoting speed, parallelism, and energy efficiency. These gains are moreover supported by local computations, and the sparsity of spiking events in time. A number of useful applications on such systems have already been realized [15, 16, 18, 21–28].

1.1.2.2 Analog and Mixed-Signal Neuromorphic Hardware

Analog hardware [19, 20, 29, 30] employ time-continuous physical dynamics to model neural and synaptic operations rather than digital abstractions. Many such systems still incorporate digital infrastructure for communication and management, such as AER communication, which are then referred to as *mixed signal hardware*. Transistors, capacitors, and resistive circuit elements all approximate neuron-like computing operations by virtue of their natural physical properties. Therefore, a neuron can be realized as a direct physical simulation rather than as an approximation in digital space. This permits considerable gains in speed and energy efficiency for certain operations. In particular, leaky-integrator computing elements (see Sec. 1.2.2) naturally lend themselves to physical simulation by LR circuits, allowing for temporal dynamics to be realized directly rather than according to a computationally cumbersome simulation by discrete-time numerical integration.

Some systems such as BrainScaleS [19] and Neurogrid [20] utilize considerable digital intermediaries to realize neuromorphic computations. Other, in development, systems such as superconducting optoelectronic networks (SOENs) [31, 32] propose little no digital intermediaries, relying instead on analog circuits for integration and photonic events for communication. By departing from the proven reliability of digital systems, such ambitious

hardware projects stand to incur unprecedented gains in speed and efficiency by forgoing digital latencies entirely and fully exploiting the exotic nature of novel circuit elements for computation.

1.1.3 Neuroscience

Machine learning and computing hardware are some of the biggest concepts in technology today, their link to neuroscience only tenuously bound by the name "neural network," but for neuromorphic computing, inspiration drawn from neuroscience, and insights lent to neuroscience, are fundamental to the field. A human brain and a large language model (LLM) both retain convincing language abilities. One consumes the yearly energy budget of a small city to train, and the other about one fifteen-thousandth of that [14]. We are thus motivated to ascertain what biology has mechanistically discovered to implement intelligence. Neuromorphic computing adopts the key computational primitive of the brain, such as synaptic weights and discontinuous spiking activations, as well as key principles, such as parallel in-memory distributed compute over large networks of heterogeneous connectivity patterns, to design novel computing systems that offer brain-like gains in energy efficiency. Moreover, the task of engineering a neural computing system sheds considerable light on the capabilities of the brain and can even inform on the sorts of intelligent computing mechanisms we might find there within.

We formalize biological neural computation in terms of four fundamental components: state dynamics, event-based communication, network connectivity, and synaptic adaptation, all of which are realized as instantiations idiosyncratic to the brain, but which are adaptable to hardware and computational design. We have already hinted at the dynamical properties of neuro-inspired systems in Equation 3.1, which in neural computation might be characterized as

$$\frac{dv_i(t)}{dt} = F(v_i(t), I_i(t)) \tag{1.6}$$

where $v_i(t)$ corresponds to the *membrane potential* of the i^{th} neuron in a network at time t, evolving according to the influence of its own state and incoming ionic current I. The functions, network topology, and temporal adaptations remain to be introduced, and will be addressed in the next three sections respectively as the defining characteristics of a neuro-inspired computing system. We will see how each of these features can be defined in terms of their role in machine learning, hardware design, and neuroscience simultaneously.

1.2 Neuromorphic Functions

Abstractly, a neuromorphic function is defined by the input/output relationship occurring at some node in a network, such that for any input x, we have the output $a(x)$. In the machine learning context, we can simply think of this as an activation function. In hardware terms, this will be a physical circuit with some I/O relationship, and in neuroscience, this would usually be a neuron (or a component of a neuron, like a dendrite). See Tab. 1.2 Generally, we will think of neuromorphic functions as those which are not sensibly reducible in any of these framings. For example, in the SOENs hardware context, fluxons contribute to an accumulation of flux in a in an LR circuit, but are not treated individually with respect to neural computation, because they are subsumed into an integration process. Therefore, the rate of flux quanta production defines the activation function rather than the JJs that produce them. Hardware tend to implement some physically constrained, noisy version of the ML abstraction of an activation function, and some over-simplification of the biological sophistication of a neuron. The final product may fall on a wide scale of complexity, and we will here list a typology of the most common neuromorphic functions.

Framing	**Neurmorphic Function**	**Example (LIF neuron)**
Machine Learning	$x_{i_{t+1}} = f(x_{i_t}, u_{i_t}, \Theta_i)$	$\frac{dV(t)}{dt} = ...$
Hardware Design	physical computing circuit	RC circuit + thresholding element
Neuroscience	Corresponding neural element	Leaky-integrate and fire neuron model

1.2.1 McCulloch-Pitts

Early success in neural networks for machine learning implemented McCulloch-Pitts (MP) neurons [33] in the context of Rosenblatt's multilayer perceptron (MLP) [34]. The idea underpinning this implementation is that the activity of a neuron in time might be reducible to a single scalar value for a given input x such that

$$a_j(x) = a_j(z_j) = f\Big(\sum_{i=0}^{N} w_{ij} x_i\Big) \tag{1.7}$$

where N is the number of input weights, j is the neuron index in question, a is that neuron's activation, $f(\cdot)$ is some (usually nonlinear) function. Time is not encoded in MP neurons, unless explicitly treated in the context of recurrent neural networks (see Sec. 1.3). It is instead assumed that the scalar value a is the activity that a network of settle to for a given input and parameter set. Without time, linear activations are insensible because a deep

network of neurons could be collapsed to a single layer [35–37]. It is therefore the nonlinear activation that give a deep network meaningful texture. Many activation functions have been adapted by MP neurons for context-dependent reasons like computing efficiency or computing sensitivity. MLPs commonly employ rectified linear units (ReLu) [38], given by

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{1.8}$$

for intermediate layers because this function is efficient to compute. Sigmoid activations, given by

$$a_j(z) = \frac{1}{1 + e^{-z}}, \tag{1.9}$$

are commonly used in output layers to produce probabilistic predictions that pair naturally with cross-entropy loss, improving gradient sensitivity during learning [35, 39, 40]. The hyperbolic tangent (tanh) [41] function is a less common activation for MP neurons, but we will make mention of it here because it is later utilized in this dissertation's original $\vec{\delta}$-MGD algorithm [8] for its characteristic symmetry about zero (see Fig. 1.2) given by

$$a(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}. \tag{1.10}$$

This symmetry will be required by an absolute thresholding operation in $\vec{\delta}$-MGD that should treat excitation and inhibition equally, and so we encounter a case where an activation function might be selected for its unique response features. Other common activations for MP-neurons include Leaky ReLu [42], expontial linear unit (ELU) [43], Gaussian Error Linear Unit (GELU) [44], all visualized in Fig. 1.2.

In machine learning, these activation functions are simply abstract mathematical functions. In hardware, these tend to be efficiently realized by vector-matrix multiplications (MatMuls) on GPUs. In the neuroscience context they are meant to approximate neurons that have reached some steady activation state over some input.

1.2.2 Leaky-Integrate and Fire

Increasing in complexity, and (marginally) toward biological realism, we now move on to those activation functions that naturally encode time in the form of *spikes*, which can be thought of as an instantaneous impulse arriving to, or emitting from, the neuron at a particular moment in time. The leaky-integrate and fire (LIF) neuron integrates incoming spikes into a membrane potential V that is itself leaking according to some time constant τ_m, and fires [45] spiking events [46] to downstream neurons whenever some threshold V_{th}

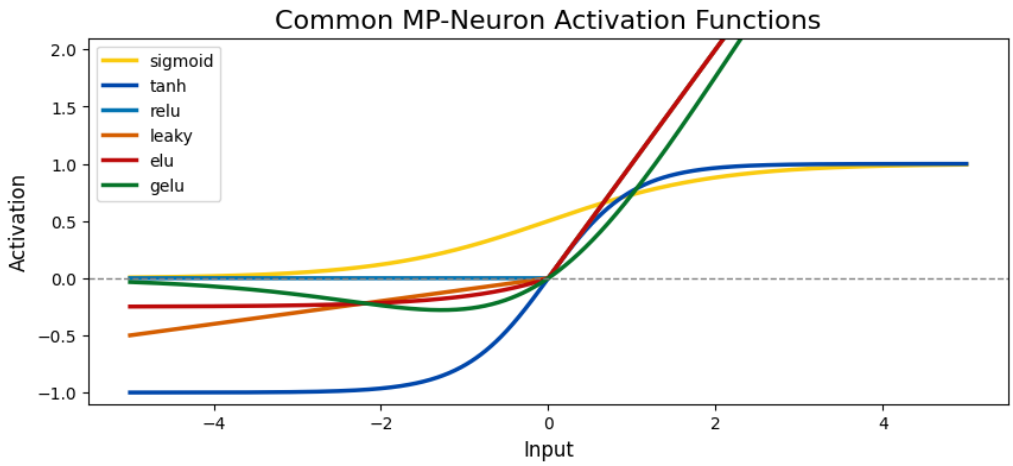


Figure 1.2: Common MP-neuron activation functions for like inputs. Different functions display different sensitivities over different input ranges. For example, the ReLu functions treats all values below zero equivalently, while sigmoid function flips inflections at the zero point. Only the tanh activation function preservers symmetry about zero across both input and output.

is reached before resetting its membrane potential to zero. We are thus given typical LIF neuron dynamics (excluding hyperparameter variations) according to

$$\tau_m \frac{dV(t)}{dt} = -V(t) + R_m \cdot I(t)) \tag{1.11}$$

$$s(t) = \Theta(V(t) - V_{th}) \tag{1.12}$$

$$\Theta(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{1.13}$$

$$s(t) = 1 \implies \text{spiking event}; V = 0 \tag{1.14}$$

where membrane potential V natural leaks by way of the $-V(t)/\tau_m$ term, integrates input current $I(t)$ according to $R_m/\tau_m = C_m$, and spikes when $V = V_{th}$ by way of the Heaviside function Θ (not to here be confused with learnable parameters), which produces 1 only when its input is nonnegative, resulting in a spiking event where V is reset to zero and all downstream neurons receive an integrateable impulse of current. This superposition of perspectives is indicative of elements like the LIF neuron that so naturally traverse these three domains.

Populations of spiking events can encode arbitrary information, such as speech, vision, and abstract signal values, through temporal, rate-based, or population coding mechanisms [46, 47]. These information-rich impulses are integrated into a neuronal activation value according to some weight associated with the unique path of that spike source with the

H

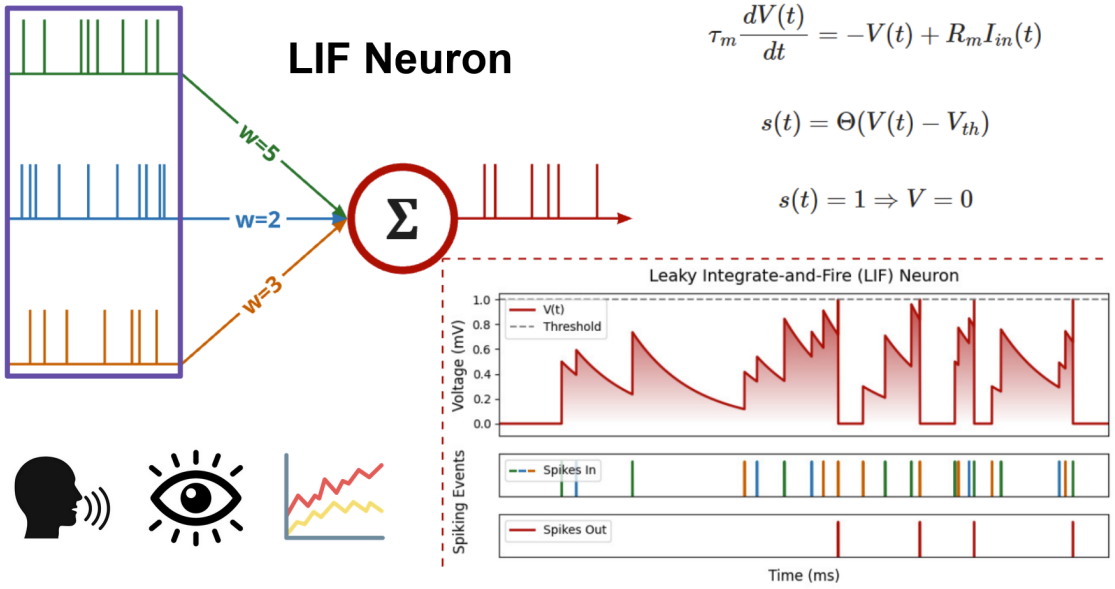


Figure 1.3: Leaky integrate and fire neuron model: A neuron receives synaptically weighted input spikes (upper left), that may represent arbirtray information domains (lower left) and are integrated over time as membrane potential V against some leak rate and subject to fire at some threshold V_{th} (lower right) according the LIF update equations (upper right).

neuron in question (see Sec. 1.2.5). This integrated value is associated with the *membrane potential* in terms of biology, a *real number* in terms of computation, and a *voltage* in terms of hardware (see Tab. 1.2). In all cases, this value is subject to a continuous leak in time according to some leak term. The result of these mechanisms together is a functional element which integrates weighted input encoded in time in contention with a temporal decay.

There is some debate about which of these encoding mechanisms is most key in terms of computation. The difference in spike time arrival [48], the delay between spike source and integration [49], the rate at which spikes arrive [47], all encode information about input and will affect the nature and time of neuron output. Here we see the capabilities of LIF neurons sharply rise when considered in a networked setting. The output of a neuron is itself a spike whose timing is determined by the moment integrated membrane potential reaches a threshold value V_{th} (see Fig. 1.3). Upon meeting with this value, the neuron (usually) depletes all membrane potential and sends a spiking event to all downstream neurons to which it is connected according to some network topology (see Sec. 1.3) which themselves will receive the spiking events accordingly to uniquely weighted synaptic connections. Thus

the neuron receives time-encoded spiking information with numerous possible key sources of processable information (timing differences, delays, rate, etc) before itself passing along this processed information to a network of other such elements, some of which may even feed back onto the source neuron by way of recurrence. The surprising complexity which may consequently emerge form a spiking neural network (SNN) gives rise to considerable interest in the question of how to realize intelligence in neuromorphic systems. We will shortly examine more biologically plausible neurons that harbor greater sophistication, which are of considerable significance to fields like computation neuroscience, but neuromorphic computing often focuses first on LIF neurons [18, 24], for their already powerful complexity in networks, and efficiently realizable operations in hardware.

1.2.3 Hodgking-Huxley

At the far end of common neuron models in terms of biological realism, the *Hodgkin-Huxley* neuron [50] has remained the gold standard in computational neuroscience since its conception in 1952, whereby measurements were taken of a giant squid axon (both giant squid and giant axon) to derive a system of equations roughly governing neuron dynamics. Where the leaky-integrate and fire neuron is described by one ODE, the Hodgkin-Huxley neuron is implemented according to four coupled ODE's. Like the LIF, we assume some external driving current I_{ext} integrating into a membrane potential V, save for now there are three additional current sources to consider (I_{Na}, I_K, I_L), each with their own dynamical properties, and a membrane capacitance C_M, such that membrane potential is given by

$$C_M \frac{dV}{dt} = I_{ext} - (I_{Na} + I_K + I_L), \tag{1.15}$$

where current terms correspond to *ionic channels* that determine the polarization of the cell. While sodium current (I_{Na}) depolarizes the neuron, potassium current (I_K) repolarizes, and some leak current (I_L) maintains the resting potential of the cell. Polarization here corresponds to the negativeness of the cell membrane potential. Intuitively, sodium ions are flowing into the cell while potassium ions flow out, both affecting total charge, and in the presence of a general current leakage. When an external stimulus drives the membrane potential to a certain threshold, the sodium channel suddenly floods, and a runaway increase in membrane potential drives the cell to a high voltage before the cell realizes a spiking event across its network of downstream neurons (see Fig. 1.4). This mechanism is called the *action potential* and is the biological process by which a neuron fires.

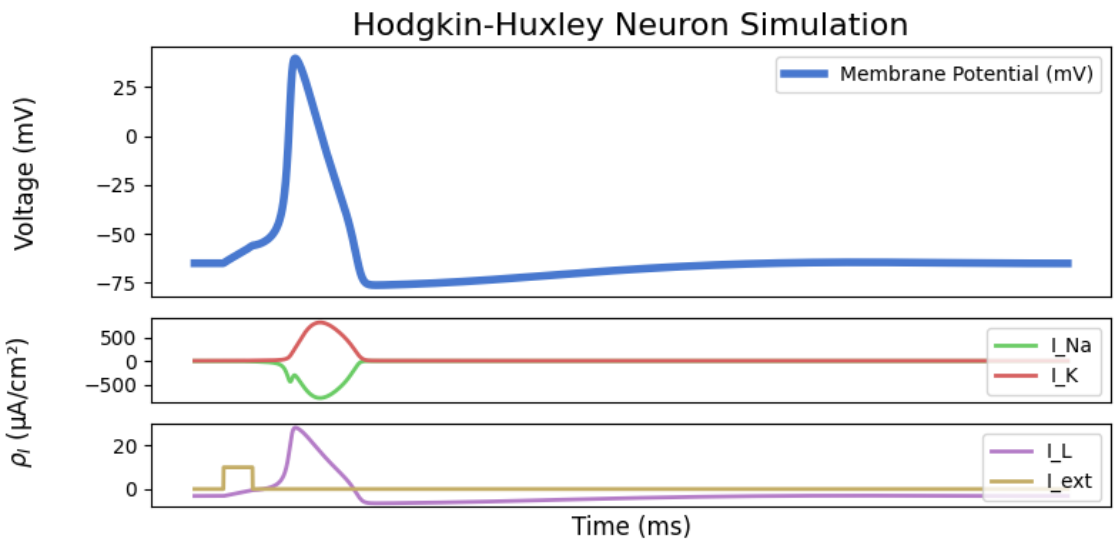


Figure 1.4: *Hodgkin-Huxley neuron dynamics*: We see here that integration and action-potential dynamics (upper plot) are more sophisticated than in LIF neurons due to the influence of multiple ionic channels (lower two plots) on membrane potential according to Eq. 1.15.

As aforementioned, the Hodgkin-Huxley model surpasses commonly targeted levels of sophistication for neuromorphic computing. It does, however, have a corresponding circuit model that ensures of the same strong overlap between biological neural mechanisms and operations realized in physical circuitry, all of which may be distilled to simple computations given by coupled ODEs [51, 52]. We have therefore included this model in our literature review to highlight the fundamental coherence between machine learning computations, hardware design, and neuroscience. It is also worth making mention that such biological sophistication is extremely expensive to simulate digitally (another reason why ML may have avoided such comprehensive models), whereas physical circuits may naturally implement these function to realize the same computations, again motivating systems of dedicated circuits which may illuminate previously infeasible simulation methods in computational neuroscience.

1.2.4 Dendritic Models

Thus far, we have treated neurons as point neurons which execute functions given some underlying equations, and have ignored the distinct morphologies that may decompose the neuron-wide functions into smaller submodules. *Dendrites* [51, 53–55] refer to branches of

neurons that may receive input from numerous weighted synapses, and perform their own leaky-integration operations, before feeding into the true cell-membrane potential of the *soma* (the cell body of the neuron that is governed by the threshold-and-fire dynamics as discussed in Sec. 1.2.2) according to some local weight. Dendrites, among other spatiotemporal operations, preserve information even after the neuron fires due to their own local integration values. This may afford interesting opportunities for computations across time because a history of neuron activity (with encoding corresponding to unique synaptic pathways) is preserved even after the action potential event.

1.2.5 Synaptic Models

We now arrive to the connective tissue of a neural network that straddles the role of a function and a topological entity. In its simplest form, a synapse can be thought of as a simple linear operation where $f_{synaptic}(x) = w_{synaptic} \cdot x$. This trivial computation, *en masse* and on the order of *trillions*, form the astounding complexity that is information flow in the human brain. Without synapses, information does not flow from one element to the next, and thus the brain does not compute. In fact, in artificial neural networks (ANNs), ranging from simple MLPs all the way up to gargantuan transformers, it is synapses through which model training is encoded. This is because it is this weighted information flow that permit the computational uniqueness of these often otherwise static architectures toward representing an array of solutions to a high-dimensional objective function, contributing significantly to model capacity of the network [35–37]. We will address these properties further in Sec. 2.4 (on learning in neuromorphic networks), but will first preface learning by a further biologically-observed extension often implemented to a synaptic model in both simulation and hardware.

While a synapse may compute an instantaneous linear operation, it has been shown that in biological systems this weight may also be subject to evolution over time according to synaptic activity [56] (see Fig. 1.5). Spike-timing dependent plasticity (STDP) [57,58] realizes a straightforward synaptic process over time by modulating synaptic strength (weight w) according the correlation of firing between the pre-and-post-synaptic neuron. If the presynaptic neuron fires just before the postsynaptic neuron, the stimulus-driven correlation is assumed to be strong and the weighted connection is strengthened. If firing is out of order, the connection strength is weakened. This simple mechanism has been accounted to contribute to homeostasis, network sparsity, and unsupervised learning, posited biologically [57], demonstrated in simulation [59], and implemented in hardware [17, 18]. Thus we see that intelligently adjusting network connection strengths over time can underpin

intelligence. Modulation of synaptic excitability is also subject to processes such as short-term synaptic plasticity (STSP) [60], neuromodulation [61, 62], and even network-activity informed modulations moderated by astrocytic subnetworks [63], all of which segue well to our subsequent sections on networks (Sec. 1.3) and learning (Sec. 2.4).

1.3 Neuromorphic Networks

The arrangement by which functional elements connect to each other determines the flow of information through a network, and therefore the system-wide input/output relationship. In machine learning, this is referred to as topology, in hardware design it is a physical structure (though sometimes abstracted, as we have seen with AER communication), and in neuroscience it is usually defined according to synaptic connectivity.

Network form gives rise to the nature of computation flow as well as information locality, both of which have considerable impact on hardware design and learning methodology. We will here introduce a typology of topologies and consider their implications on hardware design, machine learning, and neuro-plausibility.

1.3.1 Feedforward Neural Networks

1.3.1.1 Multilayer Perceptrons (MLPs)

Early machine learning success in neural networks resulted from the application of MP-neurons organized into a layered network (often called MLPs or ANNs), whereby information flows according to a directed acyclic computational graph (DAG) from the input u to some output $\hat{y}$ across layers $\{L_0, L_1, ..., L_K\}$. Unlike most neuromorphic networks, this information flow and neuron type results in a system that does *not* depend on previous states of the network. This is by virtue of that MP-neurons encode no temporal information, and that there are no cycles in the graph through which past information may recur. We therefore contrast the time independent mapping $\mathcal{F} : u \mapsto \hat{y}$

$$\mathcal{F}(u; \Theta) = \hat{y} \tag{1.16}$$

with our general neuromorphic state machine

$$\mathcal{F}(u_t, x_t; \Theta) = \hat{y}_t. \tag{1.17}$$

The two are equivalent in the case where time $t = 1$ and x_t is time-invariant. Functionally speaking, we can decompose $\mathcal{F}$ into layerwise functions such that each set of hidden

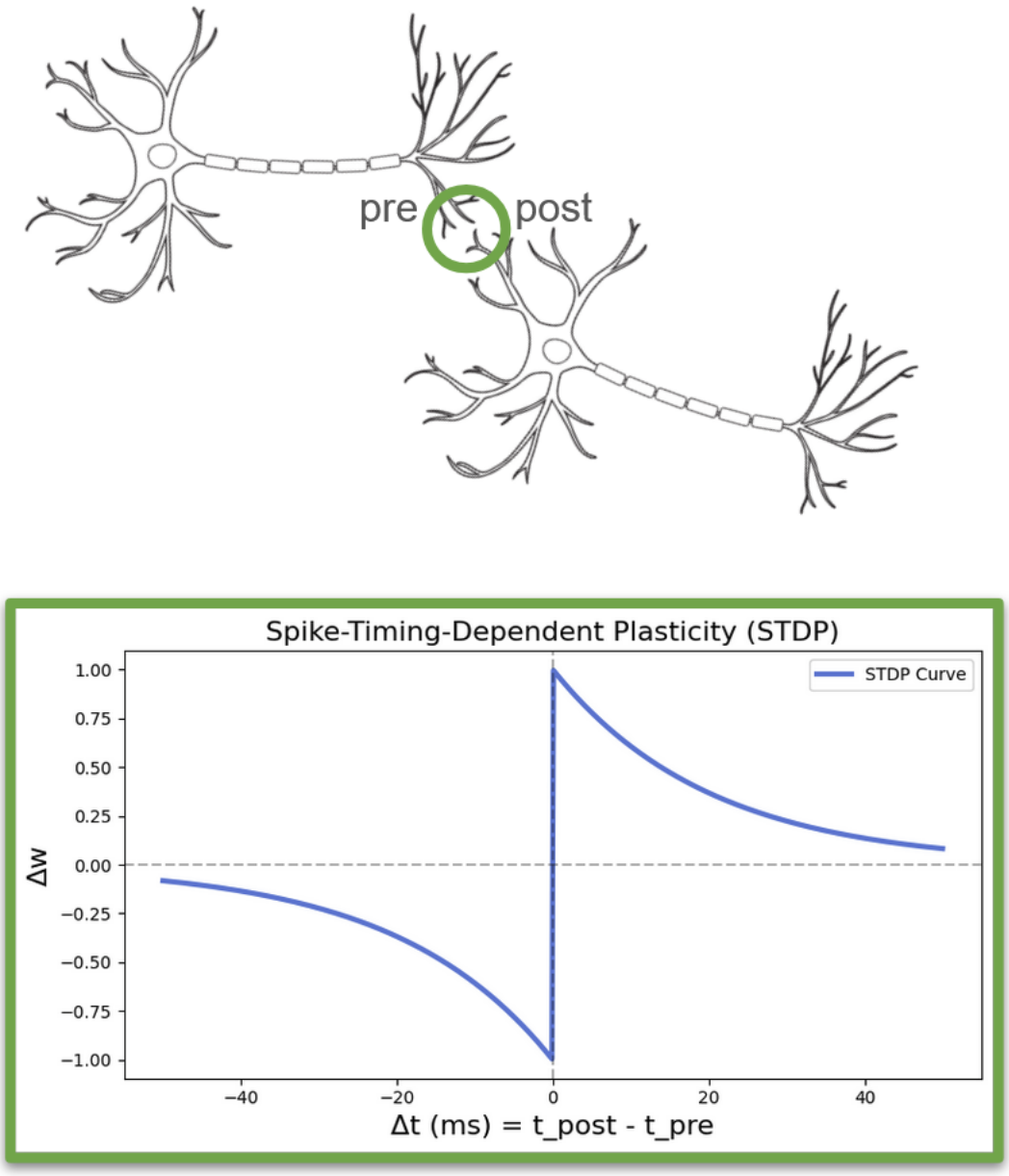


Figure 1.5: Spike-timing dependent plasticity (STDP), utilizes only information locally available to the site of synaptic connection (upper diagram) to invoke an update to synaptic weight w according to correlation in the temporal ordering of spiking events given by $t_{post} - t_{pre}$ to realize unsupervised learning that improves representational quality over the network input distribution.

activation values $\mathbf{h}^{(k)}$ at some later L_k is computed according to a layer specific activation function $\phi^{(k)}$

$$\mathbf{h}^{(k)} = \phi^{(k)}\big(\mathbf{W}^{(k)}\mathbf{h}^{(k-1)} + \mathbf{b}^{(k)}\big) \tag{1.18}$$

where the set of layer-specific learnable parameters $\Theta^{(k)}$ is given by weight matrix $\mathbf{W}^{(k)}$ and bias vector $\mathbf{b}^{(k)}$

$$\Theta^{(k)} = \{\mathbf{W}^{(k)}, \mathbf{b}^{(k)}\}, \tag{1.19}$$

with the first layer being equivalent to the input

$$\mathbf{h}^{(0)} = u \tag{1.20}$$

and the output being given by activations at the final layer L_K.

$$\hat{y} = \mathbf{h}^{(K)}. \tag{1.21}$$

Thus we have a set of functions organized in a feedforward graph that map input-to-output $u \mapsto \hat{y}$.

1.3.1.2 Dendritic Trees

A feed forward network need not be temporally independent. By definition, leaky-integrators always encode some history of past states according to their leak rate. Dendrites can be computationally defined as temporally dependent leaky-integrators (see Sec. 1.2.4) with weighted feedforward connections within a tree structure. Tree graphs are also DAGs, but with the additional clause that no element has more than one parent element. This tree will eventually terminate into a single root node from any amount of leaf nodes (all of which will contain one path to the root). In dendritic arbors, information flows backwards from leafs (synapses) to root (soma). In this configuration, a dendrite will not only depend on the state weighted sum of upstream activation values, but also on its own previous state, such that for a k^{th} dendrite with some N parent dendrites, we have

$$a_{i_{t+1}} = f\left(\sum_{n=0}^{N} \mathbf{W}_n \cdot a_{n_t}, a_{i_t}\right) \tag{1.22}$$

where $f(\cdot)$ here can be a simple summation or even a nonlinear interaction, such as with SOEN's dendritic elements.

1.3.1.3 Spiking Feedforward Neural Networks

Further temporal nuance in feedforward neural networks can be realized through discontinuous spiking functions. Many performant spiking neural networks (SNNs) adopt a simple MLP structure [64], needing only to adopt equation 1.18 with the added requirement that $f(\cdot)$ is now a discontinuous spiking function. Such networks have managed reasonably strong performance on temporal tasks such as the spiking Heidelberg-digit dataset [65], with temporal encoding being subject to only feedforward operations and past-state dependence.

1.3.2 Recurrent Neural Networks

Another natural way of temporal processing in neural networks is the application of temporal input $u_{0:T}$ to a network defined by a computational graph that contains cycles. The key insight here is that a network state will then depend on previous network states because information may now flow from one neuron back onto itself. This mechanism for representing temporal features may be coupled with temporally independent activation functions (MP-neurons) as well as temporally dependent activations (leaky-integrators, LIFs, etc). We address these network types in order of activation complexity. Note that the computational graph defining a recurrent network of N neurons is typically given by an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ where some matrix element $\mathbf{A}_{ij}$ defines a connection between the i^{th} and j^{th} neuron.

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} & ... & \mathbf{A}_{0,N} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & & \\ ... & & & \\ \mathbf{A}_{N,0} & & & \mathbf{A}_{N,N} \end{bmatrix} \tag{1.23}$$

Adjacency matrices are also referred to as transition matrices because they define how system evolves from one state to the next.

1.3.2.1 State Space Models SSMs

Recurrent temporal processing can be idealized over some sliding window using only linear activations by projecting input onto a system of basis vectors defined by high order polynomials. The state can then encode a lossless representation of recent input history under ideal linear system assumptions. Both Legendre memory units (LMUs) [66] and high-order polynomial projection (HiPPO) [67–69] employ this trick. For example, LMUs realize a

Legendre representation of past signal in state space activations by defining input mapping $\mathbf{B}$ and transition matrix $\mathbf{A}$ according to Legendre polynomials

$$\mathbf{A}_{ij} = (2i+1)\begin{cases} -1 & \text{if } i < j \\ -1^{i-j+1} & \text{if } i \geq j \end{cases} \tag{1.24}$$

$$\mathbf{B}_i = (2i+1)(-1)^i, \tag{1.25}$$

resulting in the repeated application of these terms over time, and therefore realizing polynomial orders at specific neurons. Activations produced are thus equivalent to coefficients of a Legendre expansion over past input up to that order. It is the repeated application of linear multiplication terms over input and past states that results in temporal processing, permitting some simple readout map $\mathbf{C}$ on state activations $\mathbf{x}_{0:T}$ to be learned accordingly for prediction and classification tasks given input signal $u_{0:T}$. Thus the entire system can be defined by

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}u \tag{1.26}$$

$$\hat{y} = \mathbf{C}\mathbf{x}. \tag{1.27}$$

evolving only according to simple networks of linear operations.

1.3.2.2 Reservoir Computing

A recurrent network need not be structured according to some principled scheme. Reservoir computing rather utilizes randomly initialized transitions matrices that permit the projection of input to higher dimensions (similar to a kernel trick), enabling more trivial and even linear readout maps to learn prediction and classification tasks on temporal signals. Echo state networks (ESNs) [70] utilize MP-neurons, while liquid state machines (LSMs) [71] use LIF neurons. It is worth noting that such networks may in particular benefit from neuromorphic hardware, which can physically instantiate nonlinearities and recurrence across time without necessarily having to compute $N \times N$ matrix multiplications.

1.3.3 Other Networks

Convolutional neural networks (CNNs) [72] and **transformers** have emerged as the gold standard of modern AI. While the CNN may have some correspondence to the visual cortex, neither is strongly neuro-inspired and both would merit lengthy introductions themselves. The final work of this dissertation makes use of both network types for preliminary results [8] as proof of concept of the presented algorithm, but neither networks are essentially neuromorphic and will thus not be introduced here.

Astrocytic networks are only recently emerging in the neuromorphic literature [63], but can be thought of as auxiliary networks that intelligently modulate synaptic excitability of some other network (whatever the architecture), in something like a three-factor learning scheme [63,73–75]. Astrocytes can interact with millions of synapses simultaneously [76,77] and are more ubiquitous than neurons in the brain.

1.4 Neuromorphic Learning

A neuromorphic system, composed of a network of neuromorphic functions, realizes some global input-output relations $u \mapsto \hat{y}$, where a network stimulus u produces some externally discernible affect $\hat{y}$ on network activity. At initialization of this network of functions, this mapping is very unlikely to represent any meaningful connection, and yet neural networks, both artificial and biological, are known to yield extremely intelligent behavior. What do we here mean by intelligence? As stated at the outset of this review, the goal of neuromorphic computing is to realize gains in artificial *intelligence*. To achieve an end such as this, it best be well defined, and indeed a well-founded target for intelligent behavior is necessary for a neuromorphic system (or any AI) to produce any useful results. Learning to represent an intelligent input/output mapping $u \mapsto \hat{y}$ with a set of learnable parameters Θ by way of some algorithm $\mathcal{A}$ presupposes a few things.

1. The mapping task must be *well-posed*, such that shared information I between input and target is greater than zero,
$$I(u; y) > 0 \tag{1.28}$$
implying that there should exist some (set of) function(s) f^* that can reliably map the input into some range of the output. Thus,
$$\exists f^* \text{such that } y = f^*(u) + \epsilon. \tag{1.29}$$

2. That the intelligent system in question has sufficient *model capacity* to represent this mapping, which is to say that within the space of mappings realizable by the set of learnable parameters $\mathcal{H}_\Theta$, there exists this mapping,
$$f^* \in \mathcal{H}_\Theta. \tag{1.30}$$

3. Only if conditions 1. and 2. are met, might we endeavor to determine if there exists an *algorithm* $\mathcal{A}$ that can navigate parameter space toward a set of parameters Θ^* that realize an acceptable solution to the learning task, such that
$$\mathcal{A} : \Theta \to \Theta^*. \tag{1.31}$$

It still remains to articulate what is meant by a solution to an intelligent task and what is meant by an algorithm. The only way to measure (and therefore inform toward improvement) the intelligence of a system is to define a problem that is (1) well-posed for a system that (2) has sufficient model-capacity, and (3) observe its ability to *learn* solutions to this problem. *Learn* here must imply improved solution-finding ability over time given experience with the problem. We therefore might distill a measurement of intelligence to the ability of a system to learn. Borrowing terminology and phrasing from [78], we might more generally state that intelligence is the measure of a system's ability to improve **performance** on a learning **task** as a function of **experience**.

A learning task can take on many forms. In biological systems, the most fundamental learning task is to survive and reproduce. This task is implicit in living systems, because only those that accomplish these ends will continue to exist. This broad task of course decomposes into more tractable sub-tasks, *how to fuel the metabolic demands of this body* $\rightarrow$ *how to feed this body* $\rightarrow$ *how to attain nourishment* $\rightarrow$ *how to identify edible berries* $\rightarrow$ *how to classify different berries found in a forest*? While designers of AI might like to realize such hierarchical reasoning in their systems, AI tends to first confine itself to the tip of this iceberg – a well defined input-output mapping, learned through experience; i.e. learning to classify berries through experience (visually, olfactory, gustatory).

Common machine intelligence tasks are classification, regression, prediction, generation, clustering, representation, detection, segmentation, compression, and control, which may span across arbitrary input-output modalities. Fundamentally, the question is, can we induce *experience* of some input u to improve performance measured by some output $\hat{y}$ with respect to a given learning task such as those listed above.

In most cases, these tasks are best accommodated through a *supervised learning* scheme whereby a performance measure is straightforwardly realized in a loss function (Eq. 1.5) that punishes dissimilarity between the measured system output $\hat{y}$ (edible berry) and desired output y (inedible berry), which might be measured along a number of dimensions (goodness of taste, ensuing belly ache, L2-norm against a one-hot-encoded edibility label, etc). However, in an unsupervised scheme, performance on a task might correspond to the quality of representations over an input distribution, which may involve direct measurement (k-means clustering) or not (STDP). In either case, this experience of input u ought to inform some update over learnable parameters

$$\Theta_{t+1} = \mathcal{A}(\Theta_t, x, u) \tag{1.32}$$

according to a learning algorithm $\mathcal{A}$ such that expectation of performance (whatever the measure) is improved

$$\frac{d}{dt}\mathbb{E}[\text{performance}] > 0. \tag{1.33}$$

Neuromorphic systems employ both supervised and unsupervised learning algorithms, and we will introduce example of both here, but modern AI is dominated by supervised gradient descent which we will also posit as an essential asset to neuromorphic computing. We will claim, however, that the foremost of modern AI learning algorithms for gradient descent, backpropagation, is ill-suited for neuromorphic systems because it is fundamentally mismatched for implementation in dedicated physical circuits (biological, electrical, and otherwise). Thus, we will introduce the primary aim of this dissertation; *Bio-Inspired Algorithms for Hardware-Native Neural Computing.*

1.4.1 Unsupervised Learning

As we have stated, intelligence requires learning, and learning is an improvement of performance on some task as a function of experience in some environment. In an unsupervised setting, this can most generally be defined as *an improved representation of the environment.*

By representation, we mean the sorts of states $\mathbf{x}$ (and thus outputs $\hat{y}$) corresponding to inputs u as produced by the mapping of Eq. 1.17. The key property of an unsupervised scheme is that updates to learnable parameters Θ, according to Eq. 1.32, are not driven by target values y, but rather only inputs, internal states, and sometimes outputs.

1.4.1.1 Hebbian Learning and Local Plasticity

The most common neuromorphic mechanism for unsupervised learning is *Hebbian learning* [49, 56, 58, 79–82], which has already been prefaced as local synaptic mechanisms in Sec. 1.2.5, but we will here linger on a few properties integral to the intelligence of these methods. By realizing long term potentiation (LTP) and depression (LTD) according to correlations of spiking activity with input, increasingly distinct representations are naturally learned as function of fundamental features encoded in input [57], thereby increasing the separability of input in representation space. Famously, [59] leverage this to perform classification of MNIST digits using only unsupervised STDP on a network of spiking neurons, and an additional (also unsupervised) clustering readout.

A favorable feature of Hebbian learning methods is that information required for the parameter update is available *locally* at the site of that parameter. A synapse is definitionally

connected to its pre-and-post-synaptic elements, and therefore is well disposed to make use of pre-and-post-synaptic information for a local update. Local information thus lends itself well to circuit design, where update circuits can be collocated with learnable parameters, never requiring the transport of information globally. This feature promotes distributed, in-memory operations that enable powerful parallelism in hardware design. Hebbian methods are indeed well-realized across neuromorphic implementations [15, 19, 59, 83]. Local, unsupervised learning methods are also often paired with some supervision given by a global broadcast of reward signal, akin to neuromodulation. When a learning algorithm moderates local pre-and-post-synaptic-defined updates with this third supervisory influence, we have what is called a *three-factor rule* [61, 62]. This can be a favorable hybrid learning approach in hardware because a global broadcast from some supervised loss circuit to all learnable parameters is topologically more trivial than feedback pathways through a network.

1.4.2 Supervised Learning and Gradient Descent

As we have already introduced, a supervised learning objective is most commonly defined according to a loss function, also known as an objective function. For some intelligent task that wishes to maximize expected performance on *unknown* data, it is appropriate to minimize loss on *known* data (training data), notwithstanding the phenomenon of overfitting [35]. By defining target values $\mathbf{y}$ (such as one-hot-encoded classification labels) for all N training samples, there is some set of learnable parameters Θ^* that minimizes error across all training examples, such that

$$\Theta^* = \arg\min_{\Theta} \mathcal{L}(\Theta, u, y). \tag{1.34}$$

The job of the learning algorithm, and the crux of intelligent information processing, becomes *how to find* Θ^* *among all possible* Θ (Equation 1.31). For systems with high-dimensional sets of learnable parameters, which are essential to sufficiently represent solutions to complex learning tasks, this space is computationally intractable to search randomly. Finding Θ^*, therefore, requires some bearing on the solution landscape. Knowledge of the local gradient in parameter space with respect to the objective function provides directional information toward better solutions given a specified point in parameter space $\Theta \in \mathbb{R}^N$. The gradient element ∇_i with respect to loss for any given parameter is given by

$$\nabla_i = \frac{\partial C}{\partial \Theta_i}, \tag{1.35}$$

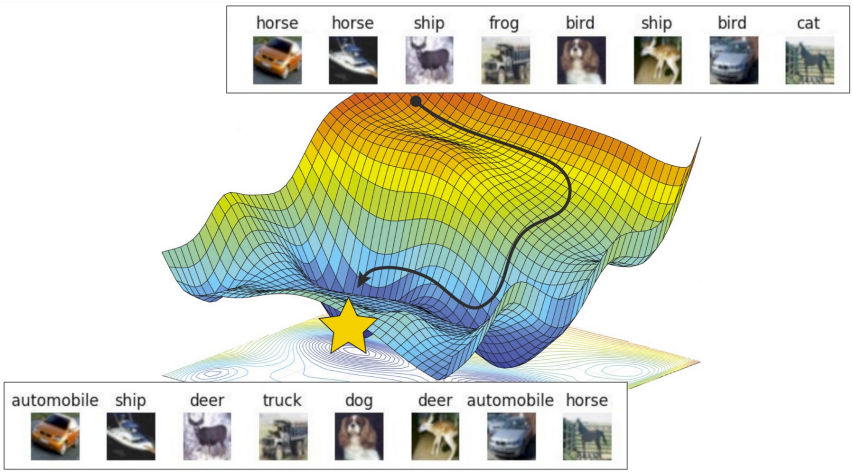


Figure 1.6: Stochastic gradient descent udpates parameters (visualized here in the flat x-y plane) to minimize some loss on an objective function (here the vertical axis), which should result in improved performance on some intelligent task (here image classification).

which answers the question, *how do small changes in parameter* Θ_i *affect the result of the global loss function*, where $C = \mathcal{L}(\Theta, u, y)$. Parameters may thus be updated according to

$$\Theta_i \leftarrow \Theta_i - \eta \nabla_i. \tag{1.36}$$

where η is a learning rate hyperparameter. This simple update drives parameters in the direction of steepest descent on the loss landscape, and toward loss minimization. Known as stochastic gradient descent (SGD) [84] (see Fig. 1.6), this update is the foundation for much of supervised learning in AI and ML. The key value in this scheme, however, is non-obvious to attain. The change in cost C with respect to each of the learnable parameters in Θ, is the *gradient* of gradient descent, and it does not come for free. We next outline various methods for calculating the gradient, both analytically and empirically, which match physical computing operations with varying degrees of efficiency and feasibility.

1.4.2.1 Gradient Descent Off-Chip

A Turing complete VNA can commit arbitrary computation so long as the expense of digital abstraction is readily paid. Thus, if a specialized hardware designer is willing to compute the gradient of a learning objective *off-chip* (*ex situ*) and instead on a traditional computer, then arbitrary computations are available to avail themselves of. This is the argument for weight-transfer and hardware-in-the-loop learning schemes, both of which calculate the gradient and appropriate parameter updates with traditional computing methods, thus exposing themselves to the limitations of the von Neumann bottleneck during training in

the hope of fully capitalizing on specialized neuromorphic hardware alone in the inference phase.

Why make this trade off? The answer is simple: *backpropagation* [85]. The backpropagation algorithm $\mathcal{A}_{bp}$ is the backbone of all learning in modern AI, across network architectures, neuron types, and task domains. The reason for its might is straightforward. For any continuous set of networked functions, backpropagation explicitly calculates the *exact* analytic solution to the gradient with respect to the objective function. For high-dimensional learning problems, this exactitude proves to be paramount in supervised learning schemes, and is achieved simply by repeated application of the chain rule backward across the computational graph, starting from the cost C and working its way through every parameter element Θ_i.

Remembering that for a simple MLP network, we have

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$
$$\mathbf{a}^{(l)} = \phi^{(l)}(\mathbf{z}^{(l)}).$$

The local chain-rule derivatives with respect to cost C are thus

$$\frac{\partial C}{\partial b^{(l)}} = \frac{\partial C}{\partial z^{(l)}} := \delta^{(l)} \tag{1.37}$$
$$\frac{\partial C}{\partial W^{(l)}} = \delta^{(l)} \left(a^{(l-1)}\right)^T \tag{1.38}$$

where bias parameter gradients are equivalent to the local derivative $\delta^{(l)}$ at the neuron, and weight parameter gradients are equal to upstream activations multiplied by local (bias) gradients. If memory and computation diversity are non-limiting, as is ideally the case with VNAs, these values can be computed for any continuous network, even across time. Backpropagation through time (BPTT) [86] simple treats time-continuous networks, such as RNNs, as a sequence of functions unrolled in time, and likewise readily computes the gradient with the chain rule.

If backpropagation can compute arbitrary gradients analytically, then is our search for neuromorphic algorithms complete? There are a number of properties in backpropagation that render it irredeemable in dedicated physical-compute systems, sequestering it as only viable by traditional computing methods, which at best may be hybridized with physical compute.

Problem 1: Backpropagation is widely considered biologically implausible because transporting changing weights backward through the graph is topologically intractable. This

challenge is coined as the *weight transport problem* [87]. Unlike local learning methods such as STDP, backpropagation requires dynamical meta network information, like weight values, to be transported backward across the network, which is fundamentally unnatural according to dedicated physical functions and communications.

Problem 2: Not all functions are continuously differentiable. Spiking activation functions, namely, are discontinuous and thus backpropagation immediately falls away from many neuromorphic advantages in terms of speed and energy savings.

Problem 3: For every computation in a system that leverages distributed, physical, in memory compute, there must be circuits that can naturally implement this computation as a function. The greater the diversity of required computations, the less tractable the system is to design. Backpropagation requires global memory transport, MatVec operations, nonlinear activations, differentiation, forward and backward pathways, all for every training example. No dedicated physical system has ever implemented all of these functions together natively in physical circuits and it is unlikely to ever be realized with neuro-inspired computing primitives [2, 88, 89].

Notwithstanding all of these strict caveats, backpropagation, along with its many bells and whistles (momentum, ADAM optimizer [90], etc), can be computed by traditional means and transferred onto a specialized system. When a specialized system is digitally modeled, trained entirely on traditional hardware (off-chip) the parameter results of which are transferred onto the specialized neuromorphic hardware, we call this the *weight transfer* method, or simply *off-chip training*.

Alternatively, one or many forward passes might be made on the specialized hardware itself, the results of which are measured out onto a traditional computer, updates calculated according over a digital twin of the hardware, and the new parameters transferred back onto the specialized, programmable chip. We refer to this case as *hardware-in-the-loop training*.

For spiking networks, both approaches usually perform *surrogate gradient descent* [64], whereby a differentiable surrogate function is used to approximate the dynamical properties of a discontinuous spiking function. Surrogate gradients enable training despite non-differentiable spike functions, but are limited as biased approximations. Moreover, for this method, and even those involving systems with continuous activation functions, modeling a physical (analog or otherwise) system is never perfect, and therefore computed

gradients are always no more than approximation. Specialized physical compute hardware often suffers from device variation, stochasticity, and measurement noise. The point of the digital system is to compute with discrete, perfectly communicable values, while a specialized hardware must do without this luxury.

Even given these detriments to learning neuromorphic parameters on traditional computers, there is moreover the price to pay of the von Nuemann bottleneck, here becoming the *neuromorphic bottleneck of off-chip learning algorithms* whereby the many advantages of neuromorphic computing, such as distributed, real-time, in-memory, dedicated compute, are sacrificed in the name of backpropagation.

It is here we remind the reader that it is not backpropagation specifically that is required to perform supervised gradient descent, but rather the gradient. Thus we posit an essential question toward the advancement of neuromorphic computing. How to compute the gradient *on-chip* using only neuromorphic operations and still achieve backpropagation-like performance?

1.4.2.2 Gradient Descent On-Chip

A number of successful efforts have been made toward *in-situ* (on-chip) neuromorphic learning, none of which stand as a definitive solution to the problem of self-training neuro-inspired physical computing systems, but all of which offer necessary insights on the nature of on-chip neuro-computing.

These learning algorithms typically start in theory and simulation on simple networks/neurons before eventually being implemented in hardware toward meaningful applications. What is key in algorithm development of this kind is to implement a learning scheme that can be feasibly realized with only neuromorphic functions and networks; that is to say, according to circuit-realizable operations, local memory, distributed asynchronous compute, and modest communication requirements such as local communication or global broadcast.

Feedback Alignment

The local gradient neuron-wise gradient $\delta^{(l)}$ at layer $l$from Eqs. 1.37 expands to

$$\delta^{(l)} = \left(\mathbf{W}^{(l+1)}\right)^T \delta^{(l+1)} \odot \phi'^{(l)}\left(\mathbf{z}^{(l)}\right) \tag{1.39}$$

where the glaring violation of neuromorphic principles is the required communication of the transposed weight matrix $\left(\mathbf{W}^{(l+1)}\right)^T$ from downstream layers (we have already made mention of the weight transport problem). This weight matrix exactly apportions credit

assignment or error to specific parameters, allowing for well-informed updates according to Eq. 1.36.

Feedback alignment (FA) [91] presents the novel insight that feedback matrices need not be symmetric to feedfoward weights in order for learning to occur. In fact, [91] shows that random fixed feedback matrices $\mathbf{B}^{(l)}$ are sufficient to approximate the gradient and learn to minimize the objective function. Instead of depending on downstream weights, local gradients are redefined such that Eq. 1.37 becomes

$$\delta^{(l)} = \mathbf{B}^{(l)}\delta^{(l+1)} \odot \phi'^{(l)}\big(\mathbf{z}^{(l)}\big). \tag{1.40}$$

That gradients communicated via random weights may still be applied toward parameter updates that will realize gradient descent may be a surprising result, save for the insight that the learning scheme is still iteratively subjected to filtration by the loss function, which will nevertheless penalize bad updates and reward good ones, resulting in a drift in parameter space that suites the available feedback matrix with respect to loss-minimization.

Feedback alignment has been implemented in a number of interesting learning contexts, over different network types, and on neuromorphic hardware [91–94].

Equilibrium Propagation

Rather than redefining the gradient equations of backpropagation, equilibrium propagation [95] instead computes the gradient according to differences in nearby equilibria of an energy model.

Continuous-time, energy-based state dynamics can be defined as

$$\frac{dx_i}{dt} = -\frac{\partial E}{\partial x_i}, \tag{1.41}$$

such that state dynamics x are governed by a minimization of energy E across time t. More simply put, that a system will settle to its lowest energy state (such as a marble placed at the inner rim of a bowl will eventually settle at the exact bottom of the bowl).

These lowest-energy points can be thought of equilibria and exist as function of network parameters and input. Thus for some particular input, we can measure the resultant equilibrium of the energy model. By input, we here mean values clamped to some subset of nodes in a fully connected neural network, and by energy model, we mean a fully connected neural network [96]. Let's say a system relaxes to some equilibrium $\mathbf{x}^0$ only as a function of input to some energy model. It is then sensible to measure the gradient of an objective function according to a nearby equilibria provoked by "nudging" another set of neurons, associated with some readout mechanism, toward a desired target value (weighted

by nudge factor β) to produce a new equilibrium x^β. Thus the gradient for some weight $\mathbf{W}_{ij}$ in this fully connected network between the i^{th} and j^{th} neuron is given by

$$\frac{\partial C}{\partial \mathbf{W}_{ij}} = \frac{1}{\beta}\Big(x_i^\beta x_j^\beta - x_i^0 x_j^0\Big), \tag{1.42}$$

thereby producing a method for estimating the gradient of an energy model by comparing relaxed and nudged equilibria given the same input. From here we can perform our SGD update once again over samples, batches, objective functions, and all the normal supervised learning metaparameters.

There are a number of limitations to EP and computing with energy models generally. Spiking networks are not directly applicable because small changes in the network evoked in the nudged phase do not necessarily correspond to a smooth measurable change in the equilibrium point. Deep networks are may also respond to small changes in output nudging in dissipating ways. EP has nevertheless seen numerous follow up work and hardware implementations [95, 97–99].

Perturbative Methods and Multiplexed Gradient Descent

Perturbative methods measure the gradient by comparing the difference in network cost with respect to small changes (perturbations) in network parameters. Importantly, these methods are *model-free*, meaning that no knowledge of the neurons/network/hardware is required, only empirical measurements of output. In the case of finite difference [100], a parameter is simply perturbed by some value ϵ in the positive and negative direction. In hardware, these are trivial local operations. In both cases, the cost is measured, and a gradient estimation with arbitrary precision according to ϵ is given by

$$\nabla_i = \frac{\partial C}{\partial \Theta_i} \approx \frac{C_{\Theta_i+\epsilon} - C_{\Theta_i-\epsilon}}{2 \cdot \epsilon} \tag{1.43}$$

which accomplishes a straightforward local measurement of the change in cost C with respect to the i^{th} parameter Θ_i as induced by $\pm\epsilon$ perturbations. This procedure can be repeated for all N_Θ parameters in the network and for all N_D samples in a training dataset, thereby enabling an entirely empirical implementation of SGD. Given that a dataset may contain tens of thousands to even millions of samples, and that it may need to be iterated over some number of epochs, the scaling of this methods for any sizable network, dataset, or learning task becomes impractical.

Simultaneous Perturbation Stochastic Approximation (SPSA) [101] instead perturbs all parameters at once with a random Bernoulli vector of length N_Θ multiplied by some

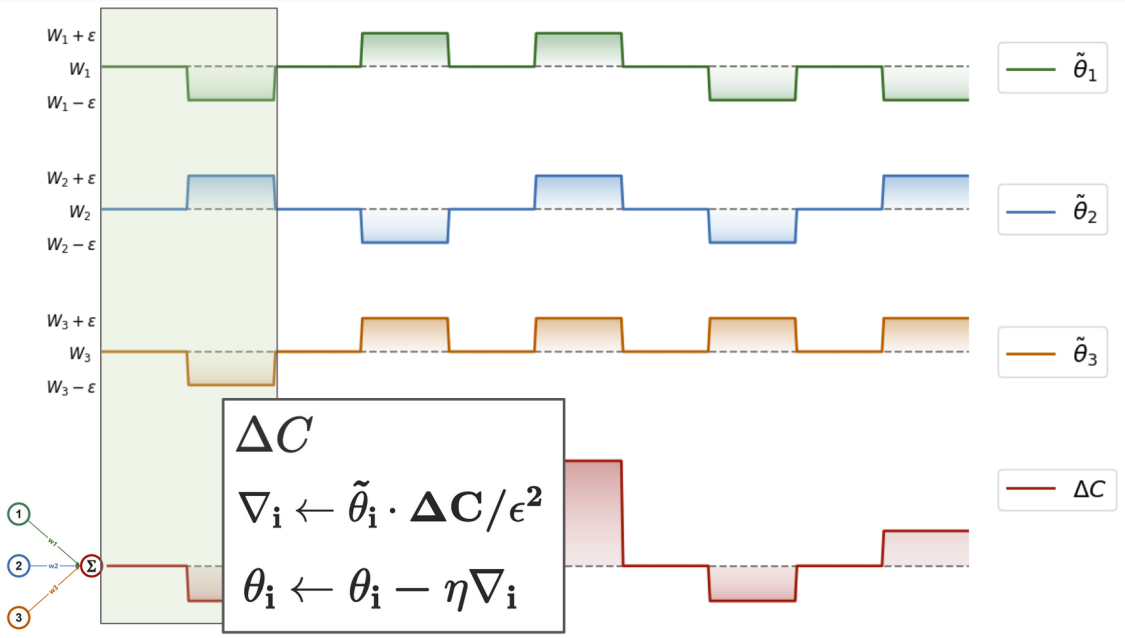


Figure 1.7: An example of a perturbative update for $\tau_x = \tau_\theta = \tau_p = 1$ within the MGD framework. We see perturbations to parameters $\mathbf{w}$ according to Bernoulli vectors $\tilde{\theta}$ occurring over different input samples (here not visualized). Updates are made locally according to the estimated gradient given by the value of the local perturbation $\tilde{\theta}_i$ as weighted by the measured change in global cost ΔC and some normalization factor $\frac{1}{\epsilon^2}$.

perturbation size ϵ. The idea here is that

$$\nabla_i = \frac{\partial C_i}{\partial \Theta_i} \approx \frac{\Delta C}{\Delta \Theta_i} \tag{1.44}$$

where ΔC_i is what the measurable change in cost would be if only the i^{th} parameter Θ_i were perturbed and ΔC is the measurable change in cost when all parameters are perturbed simultaneously. This is only an approximation of the gradient because local parameter change only correlates with global change in cost *most* of the time when weighted by gradient magnitude, as we will explore at length in Chapter 5. To improve correlation of local changes with measured global cost, one might choose to integrate a number of gradient measurements into a single gradient estimate, and indeed [101] shows that as the number of integrations approaches the number of parameters, an estimation fidelity to the true gradient achieved that is equivalent to that realizable with finite difference, but at the same cost in scaling time with parameters linearly.

Managing the balance of gradient estimation accuracy with scaling and other hardware considerations results in a few key hyperparameters to track. Multiplexed Gradient Descent

(MGD) [102, 103] introduces a framework to organize these considerations with respect to hardware design and algorithm constraints. Three main hyperparameters are proposed: τ_x, τ_θ, and τ_p track integration time for data samples, parameter updates, and perturbation vectors respectively. The optimization of these hyperparameters is not uniform across datasets or networks.

MGD offers a path for perturbative methods to be optimized for hardware and application considerations. Importantly, perturbative methods only require simple physically-realizable operations such as random number generation (RNG), local memory, and global broadcast. Though definitive methodology for SNNs remains to be seen, perturbative methods will scale to any problem size so long as sufficient integration time is allotted and the network is differentiable [101]. Thus powerful architectures such as CNNs and transformers may be trainable on specialized hardware should the problem of scaling estimation time linearly with parameter count be overcome. Indeed, perturbative methods are associated with a plethora of performant results across domains and network architectures in the context of hardware, machine learning, and neuroscience [7, 102–107].

1.5 Problem Statement:

Having introduced neuromorphic systems as a *network* of *functions* that *learn* some intelligent input-output relationship, all of which ought to be neuro-inspired and implementable in-part or entirely on neuromorphic hardware, which itself employs computing principles such as in-memory asynchronous distributed physical computations, we now endeavor to highlight where the potential for a neuromorphic advantage over traditional computing may be the greatest.

To fully utilize the known hardware advantages of neuromorphic computing, we must realize all the requirements of a neuromorphic system on such hardware. Neuromorphic functions and networks are readily implemented on physical neuromorphic circuits, but neuromorphic learning is far more challenging to realize with strictly neuromorphic hardware operations. For this reason, we have seen that much of the neuromorphic state-of-the-art involves partially digital hardware and/or integration with conventional computing. These efforts importantly expand on strictly digital VNAs, and we only here note that much opportunity remains in the exploration of fully neuromorphic self-learning circuits that do not rely on any traditional methods. Fully harnessing dedicated neuromorphic hardware will therefore require algorithms that can be implemented entirely *in-situ* and we term the absence of such a scalable, general backpropagation-competitive learning mechanism

as the *neuromorphic bottleneck* that currently bounds the field to what may be only its most nascent stage of development. We thus hope to best serve the advancement of neuromorphic computing by address the following problem statements: *How might intelligence arise on brain-inspired hardware?* and the more immediate, tractable *How to implement supervised gradient descent using only neuromorphic operations?*

1.6 Toward On-Chip Gradient Descent: Bio-Inspired Algorithms for Hardware-Native Neural Computing

This dissertation interests itself with solving the neuromorphic bottleneck by way of advancing on-chip gradient descent methods for neuromorphic learning. Chapters 2 and 3 introduce the *arbor update rule* [5, 6], which is a learning algorithm explicitly designed for those hardware that adopt dendritic arbors as neural morphologies for computation. Results for this method are the first of their kind to be simulated on SOENs and the first insight into training a hardware that is designed to approach the limits computation in terms of speed and scalability. Chapter 4 abstracts away from specific hardware conditions and realizes gradient descent in RNNs using only neuromorphic operations on SSM networks [7], offering a novel integration of the MGD framework for perturbative learning with recurrent networks. Finally, Chapter 5 extends MGD to the novel $\vec{\delta}$-MGD [8], where the scaling issues of perturbative methods are overcome by employing astrocytic subnetworks to implement gradient-informed three-factor hebbian updates on MLPs, CNNs, and even transformers, realizing equivalent performance as backpropagation with orders of magnitude fewer gradient component computations, using only hardware-friendly operations. We thus present a set of earnest contributions (across hardware, applications, functions, and networks) toward the effort of *bio-inspired algorithms for hardware-native neural computing.*

1. AN INTRODUCTION TO NEUROMORPHIC COMPUTING

2

The Arbor Update Rule for Image Classification

Citation and author list: [108]

Paper Abstract

Superconducting Optoelectronic Networks (SOENs) combine photonics and superconductors to instantiate computing systems that approach the fundamental limits of information processing in terms of speed and scalability. Overcoming the engineering challenges of integrating these technologies into one system has consistently been aided by using neuro-inspired architectures. SOENs are natively Spiking Neural Networks (SNNs) of *loop neurons*, which themselves are comprised of many subsequent dendrites organized into intricate morphologies. Therefore, SOENs at scale may embody highly complex superstructures that demand commensurate learning methods. To that end, we here propose a simple activity-based update rule (the *arbor-update*) for dendritic learning that is found to successfully classify nine-pixel images with a single neuron. We test two amendments to the arbor-update on a winner-take-all (WTA) mutually inhibitory three-neuron network. Both *elastic weight collision* with dynamical boundaries and *intermittent validation* are found to improve convergence time and conditions. The arbor-update and its variants are scalable with SOENs and may even map to other systems. Importantly, all proposed learning methods are expected to be entirely implementable for *on-chip* learning in SOENs.

2.1 Introduction

Like many emerging neuromorphic systems [15, 16, 19, 83, 109], Superconducting Optoelectronic Networks (SOENs) break away from traditional Von Neumann architectures, and instead compute with asynchronous spiking neurons. SOENs utilize superconducting cell bodies, silicon light sources, and optical communication [31, 110, 111]. These components come together to form the *loop neuron*, an example of which can be seen in Fig. 2.1. Loop neurons can host dendritic arbors which allow for more complex within-neuron processes and have been shown to be useful for learning and computation [112–118].

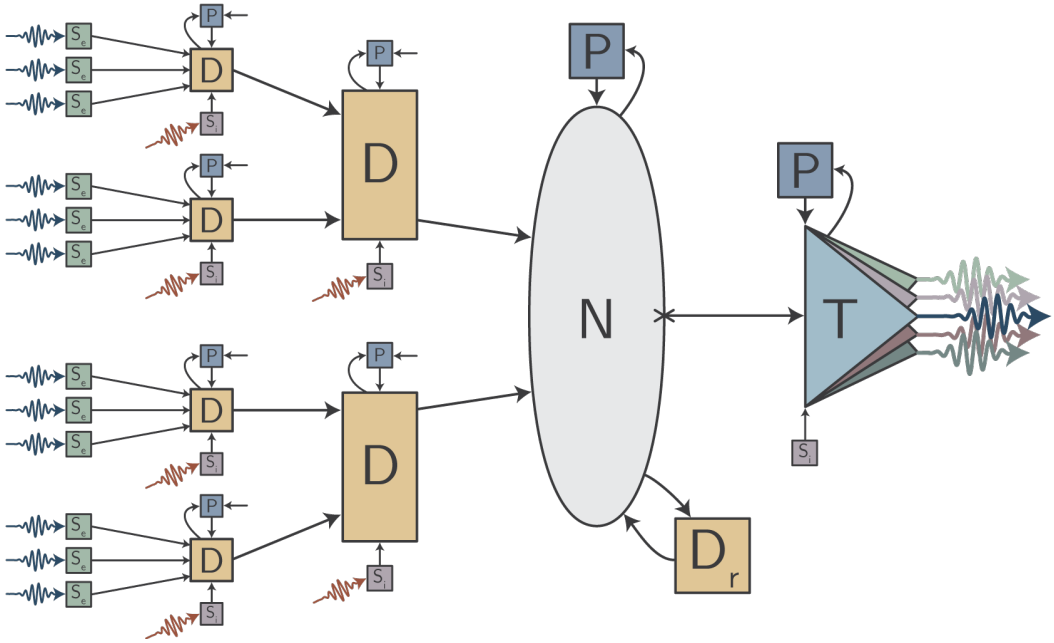


Figure 2.1: An example of a loop neuron with non-point morphology. S_e = excitatory synapse, S_i = inhibitory synapse, D = dendrite, N = soma, T = transimtter circuit, and P = plasticity modules.

In Fig. 2.2, we can see a simplified (monosynaptic) point neuron in circuit form. Here a brief overview of circuit dynamics is given, but only the flow of information across components is necessary to understand the computational properties of SOENs. Starting at the synapse, light pulses are received in the few-photon domain by a superconducting single photon detector (SPD). Current in the SPD is inductively coupled as flux with a coefficient (analogous to a synaptic weight) to a superconducting quantum interference device (SQUID). The flux coupled into the SQUID must reach a bias-current-dependent threshold before the time-average voltage across the circuit becomes non-zero, after which point

fluxons are emmitted and captured by an adjacent L-R loop. Current is driven, integrated, and leaked by the L-R loop and thus the SQUID and L-R loop together form a mechanism that is qualitatively similar to a dendrite such that information from upstream synapses or dendrites is integrated and leaked (with a definable L/R time constant), without ever spiking. A soma is a dendrite with a transmitter circuit that may, upon reaching a defined threshold, trigger a superconducting switch (called a tron) to provoke a silicon diode light source to fire into a dielectric waveguide, routing photons to all downstream neurons. The triggering of the tron also flushes the somatic dendrite of all integrated current (but not the current of pre-somatic dendrites), while simultaneously adding signal into an attached refractory dendrite that will inhibit the soma with a given L/R time constant. Together these superconducting optoelectronic components form a *loop neuron* (named for the many receiving and integrating loops involved). To summarize, a loop neuron is a structure with synapses that receive signals in the form of light, dendrites that integrate and leak current, and somas which integrate and leak current, firing photons to all downstream synapses if a given threshold is reached. These operations and components are demonstrated in a growing body of experimental work [110, 119–121]

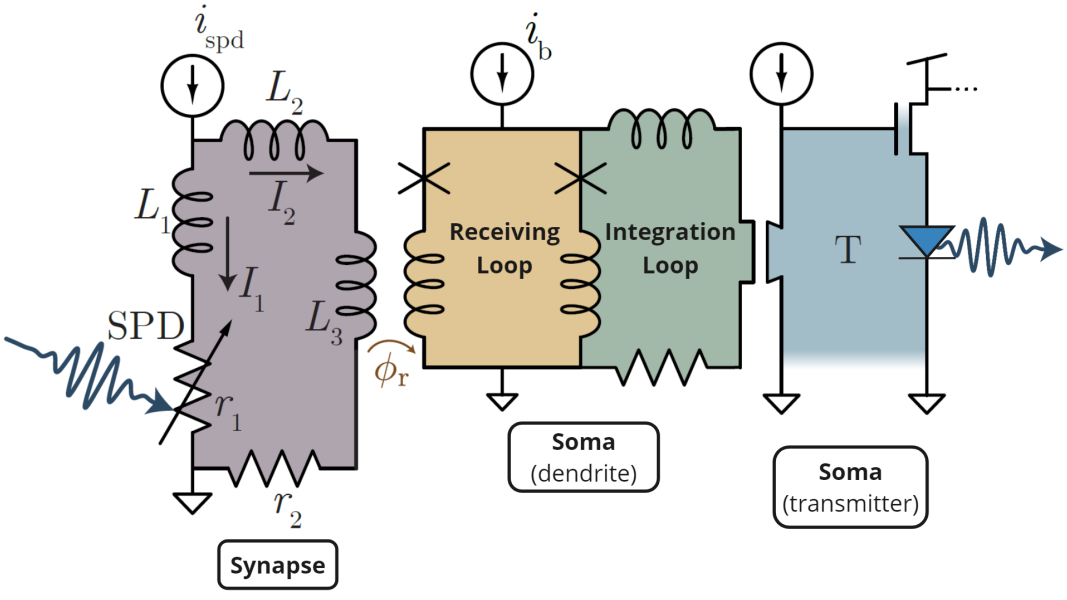


Figure 2.2: Circuit diagram for monosynaptic loop neuron.

Section 2.2 overviews the simulation methods used to experiment with SOENs. Section 2.3 investigates the computational primitives of loop neurons. Readers familiar with SOENs, or at least with spiking neural networks, may wish to start directly with Section

2.4, which explains plasticity and learning mechanisms at the single neuron level. Section 5.7 outlines the nine-pixel problem (with and without noise) and gives experimental results. Finally, Section 5.9 interprets results, discusses limitations, and speculates on future implications for the development of SOENs.

2.2 Simulation

In order to simulate the picosecond dynamics of SOENs, a phenomenological model is used [122], as solving circuit equations for such a system would run prohibitively slow. The model nonetheless maintains a $10^{-4}\chi^2$ accuracy to the circuit equations and executes ten thousand times faster. All units in this model are normalized to be dimensionless. The principal ordinary differential equation for dendritic current integration

$$\beta\frac{ds}{dt} = r(\phi_r, s; i_b) - \alpha s \tag{2.1}$$

is solved for all relevant components (anything with a dendrite) with the forward Euler method, where β and α are physical constants determined in fabrication, relating to loop inductance and resistance respectively. Signal, s, refers to the amount of current integrated into an integration loop in dimensionless units normalized to critical current I_c. i_b denotes the biasing of the SQUID circuit (greater values of which essentially lower the flux threshold required to begin signal integration). ϕ_r is the flux incident on the receiving loop. Together, ϕ_r, s, and i_b define the rate of flux quantum production r which is the source of signal, s, at each time step. The function r is obtained prior to simulation and subsequently accessed as a lookup-table for each time step in the phenomenological simulation. The forward Euler thus becomes:

$$s_{t+1} = s_t\left(1 - \Delta t\frac{\alpha}{\beta} + \frac{\Delta t}{\beta}r(\phi_r, s; i_b)\right) \tag{2.2}$$

where t is the time step of the simulation. Integrated signal decays exponentially with a time constant $\tau = \alpha/\beta$ and integrates as a function of received flux ϕ_r, bias current i_b, and signal s from the previous time-step. Because α and β are constants set in fabrication, i_b and ϕ_r become the key dynamic parameters of the system, both of which can be influenced in supervised and/or local fashions. Modulating bias current can change the effective 'weight' of a dendritic or synaptic connection in real-time. For the same quantity of received flux and a constant inductive coupling strength, the impact on integrated signal can be changed. Another method for changing the effective weight between two components is to offset the received flux with some ϕ_{offset} such that the total received flux will be more or less by some defined quantity. This can also be done dynamically and will be exploited to

facilitate the arbor-update in Section 2.4. Also relevant is that an SPD can only respond to a new incident photon once every $\sim 35ns$ and a tron can only be triggered once every $\sim 10ns$, which define an absolute refractory period for synaptic events and somatic firing respectively. The SPD response to a photonic event is a steep jump in flux ($\sim 10ps$), followed by an exponential decay on the order of $\sim 35ns$. All parameters and architectures are deployed over the phenomenological model using *sim_soens* (Simulator for Superconducting Optoelectronic Networks). Code is available at: *https://github.com/ryangitsit/sim_soens*.

2.3 Computational Primitives

To demonstrate basic properties of loop neurons, we first constrain ourselves to the monosynaptic neuron, as seen in Fig. 2.2, before expanding to more complex morphologies. All dynamics of the soma apply to *any* dendrite, save for firing and refraction.

2.3.1 Leaky Integrators

Loop neurons respond to synaptic events by integrating SPD flux as current into the dendritic integration loop via the dendritic receiving loop. In Fig. 2.3 we see three synaptic events integrated into the somatic dendrite of a monosynaptic point neuron for different parameter settings, where firing threshold is set sufficiently high to only elicit leaky-integration behaviour. In accordance to equation 3.1, we see that bias current i_b is positively correlated with the amplitude of signal integration in the somatic dendrite, the time constant τ determines signal leak rate, and the inductance parameter β informs the shape of integration. Together, these terms allow for significant tailoring of dendritic signal integration shapes.

2.3.2 The Flux Threshold ϕ_{th}

When flux arrives at the receiving loop of a dendrite, this does not immediately result in signal accumulating in the integration loop. Some bias-dependent threshold must first be reached. In Fig. 2.4 (top), it can be seen that when received flux is below the flux threshold ϕ_{th+}, no signal is integrated. In Fig. 2.4 (middle), received flux just passes the threshold and some current is integrated. This threshold can be raised and lowered by changing the bias current–the greater the bias, the lower the threshold. As seen in Fig. 2.3 (top), for greater bias currents, more signal is integrated given equivalent synaptic events. This is also evident from Fig. 2.5, where it can be seen that no fluxons are produced for certain applied flux values at given current biases. Due to the symmetry of fluxon production

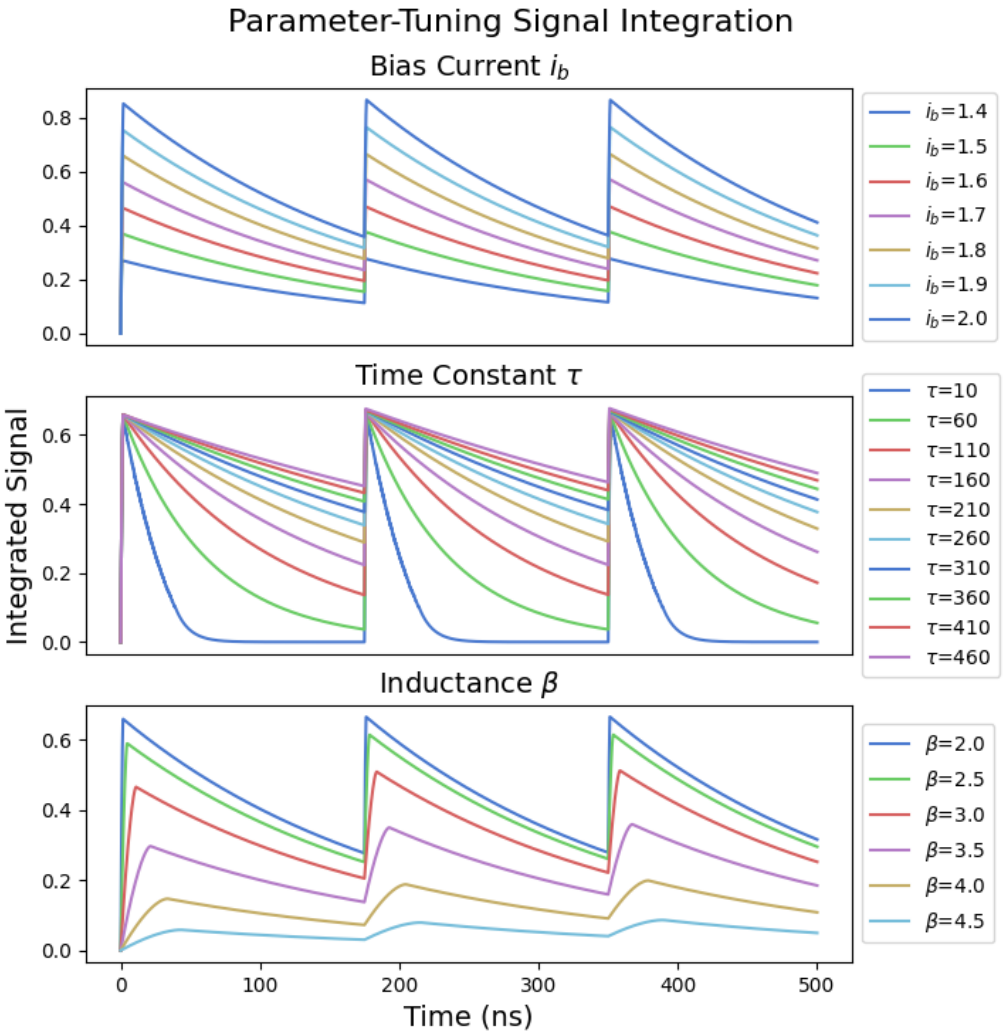


Figure 2.3: Leaky-integrator shaping with parameter tuning for incoming synaptic events at 0, 175, and 350ns.

over applied flux about zero, there is a negative flux threshold ϕ_{th-} that when crossed, will also integrate *positive* signal, such as seen in Fig. 2.4 (bottom). Negative excitation is sometimes observed in the brain [123] and may serve computational purposes, but can just as well be limited by remaining within a certain activity range using deliberate parameter selection or bounding mechanisms on inhibitory inputs to a dendrite.

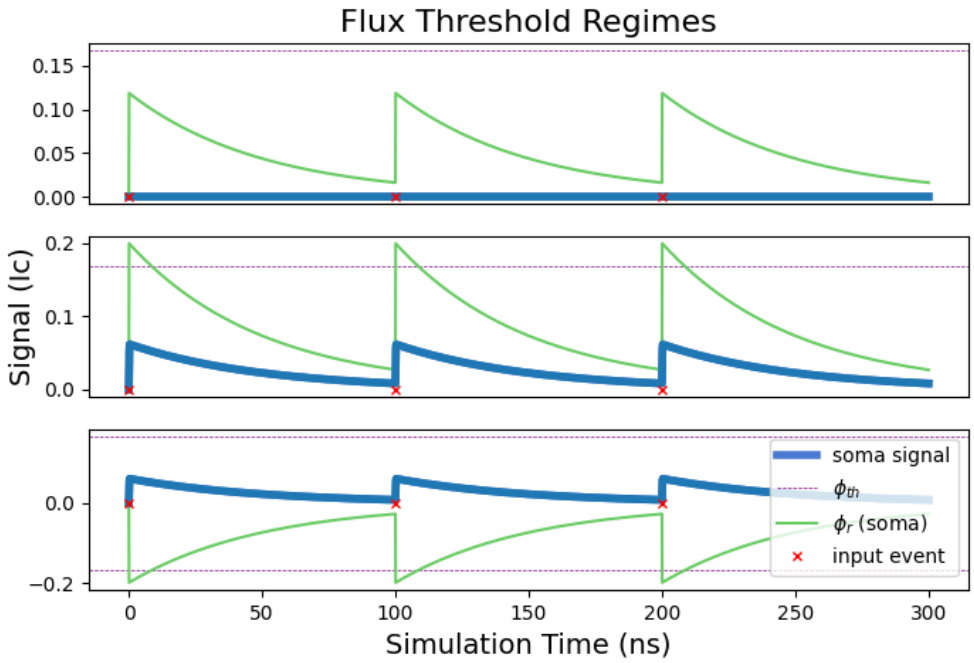


Figure 2.4: Integrated signal for different amounts of received flux ϕ_r.

2.3.3 Rollover and The Dynamical Range

SOENs offer a large and diverse space of biologically plausible behaviors and also include many dynamics outside the realm of typical neurons. The role these non-biological features may play in computation is to be determined and for the time being SOENs can simply be constrained to certain ranges in parameter space such that only biologically plausible behaviour is manifested. One such constraint is defined by *rollover* whereby when a certain amount of flux is received for a given bias current, the signal integrated into the dendrite via this flux no longer increases, but rather declines along a curve that is symmetric to the one it has just ascended. This cycle may continue periodically for increasing values of flux and is a result of the periodic nature of SQUID responses to applied flux. A potential trajectory of integrated signal in response to a strongly coupled synaptic event can be

observed in the sequentially lettered phases (A-G) of Fig. 2.5.

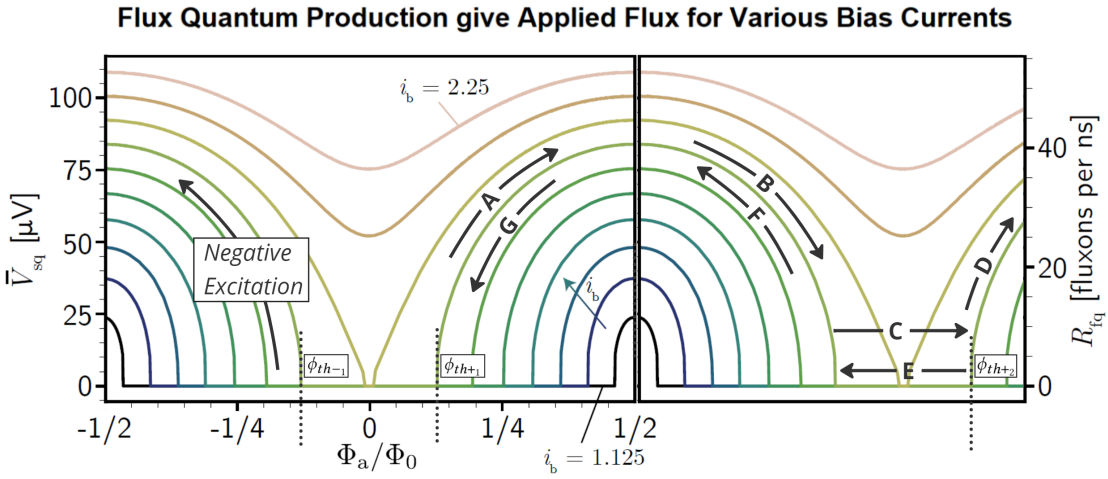


Figure 2.5: Rate of fluxon product (R_{fq}) as a function of applied flux ϕ_r. Following the fourth path from the top, where $i_b \approx 1.8$, applied flux may cross over several thresholds that initiate flux quantum production (and therefore signal integration), labeled ϕ_{th}. Different phases in a potential applied flux trajectory (in reponse to a strongly connected SPD) are labeled sequentially with letters A-G.

To demonstrate the relationship of received flux and integrated signal, Fig. 2.6 plots different coupling strengths between the SPD and the somatic dendrite. In doing so, the amount of *peak* flux received is altered. For all values of received flux less than 0.5 (the point of rollover for applied flux), somatic response remains proportionate to the SPD response (plots 1-3). However, the rate of fluxon production r decreases as applied flux exceeds the rollover point at 0.5 (phase B in Fig. 2.5) and therefore the rate signal accumulation is diminished (plots 4-7). For very high values of applied flux (subplots 8-10), the rate of fluxon production reaches the second trough (phase C in Fig. 2.5) in the periodic cycle thereby integrating zero flux, before sliding back over phases B and then A.

While this signal trajectory might seem atypical in terms of neuron dynamics, it may serve as a useful homeostatic mechanism [123] should a SOEN designer choose to exploit it as such. However, this moment of decreasing signal for increasing flux may just as well be avoided by arranging related neuron parameters such that this amount of flux is never reached. This can be achieved by simply limiting coupling strength into any receiving loop so that for no reasonable incoming values of current will received flux surpass this rollover point. There are also feasible circuit design solutions to constrain activity to a selected *dynamical range*, like $\in [\phi_{th-}, \phi_0/2]$ where there is no negative excitation and only

monotonically increasing positive excitation. The computational value of these approaches will be explored in Section 5.7.

2.3.4 Firing, and Refraction

Without firing behavior, activity would be confined to a single neuron. It is the photonic communication that allows SOENs to negotiate information at the large network scale. Once the somatic threshold is reached, the signal in that integration loop is purged, and a refractory dendrite is driven, which temporarily forces the soma into a quiescent state. The signal now in this refractory dendrite will negatively inhibit the soma's own signal integration for some time defined by the τ_{ref} time constant associated with that refractory dendrite. Upon reaching threshold, a light source is triggered which delivers photons to downstream synapses.

In Fig. 2.7, firing behavior for two different neuron configurations are shown. In both cases, incoming synaptic events create an SPD-driven received flux response, though because of a greater coupling strength and lower inductance parameter, only the top neuron fires (black **x** mark) in response to the first synaptic event, despite having a lower bias value. In this upper-plot case, it can be seen that the refractory signal immediately weighs down the total received flux due to inhibition of the soma (coupled negative flux). Even as the next synaptic event arrives, less signal is integrated in total due to residual refractory inhibition. By the third incoming synaptic event, the neuron is ready to fire again and the process repeats itself.

The lower-plot neuron in Fig. 2.7, however, requires two incoming synaptic events to reach firing threshold, and then takes another four events to reach threshold while refraction is in play. This combination of greater inductance values, longer time constants, and smaller coupling strengths (or alternatively, bias values), demonstrates how parameter selection can result in qualitatively different neuron firing behavior. In this case, the essential difference between the top and bottom neurons is the number of incoming events required to fire, and the dynamics of how this number changes over time due to refraction. Firing rate is a key quantity in computational behavior at scale, and these are just two of many possible configurations that comprise the diversity of neuronal behaviors of which loop neurons are capable. A key contribution to firing rate here is coupling strength because of its role in the effective weight of the synapse to the soma. In Section 2.4, we will explore how changing the effective weights via flux offset can result in different firing rates for more complex neurons. In the remainder of this section, we expand this behavioral space by not

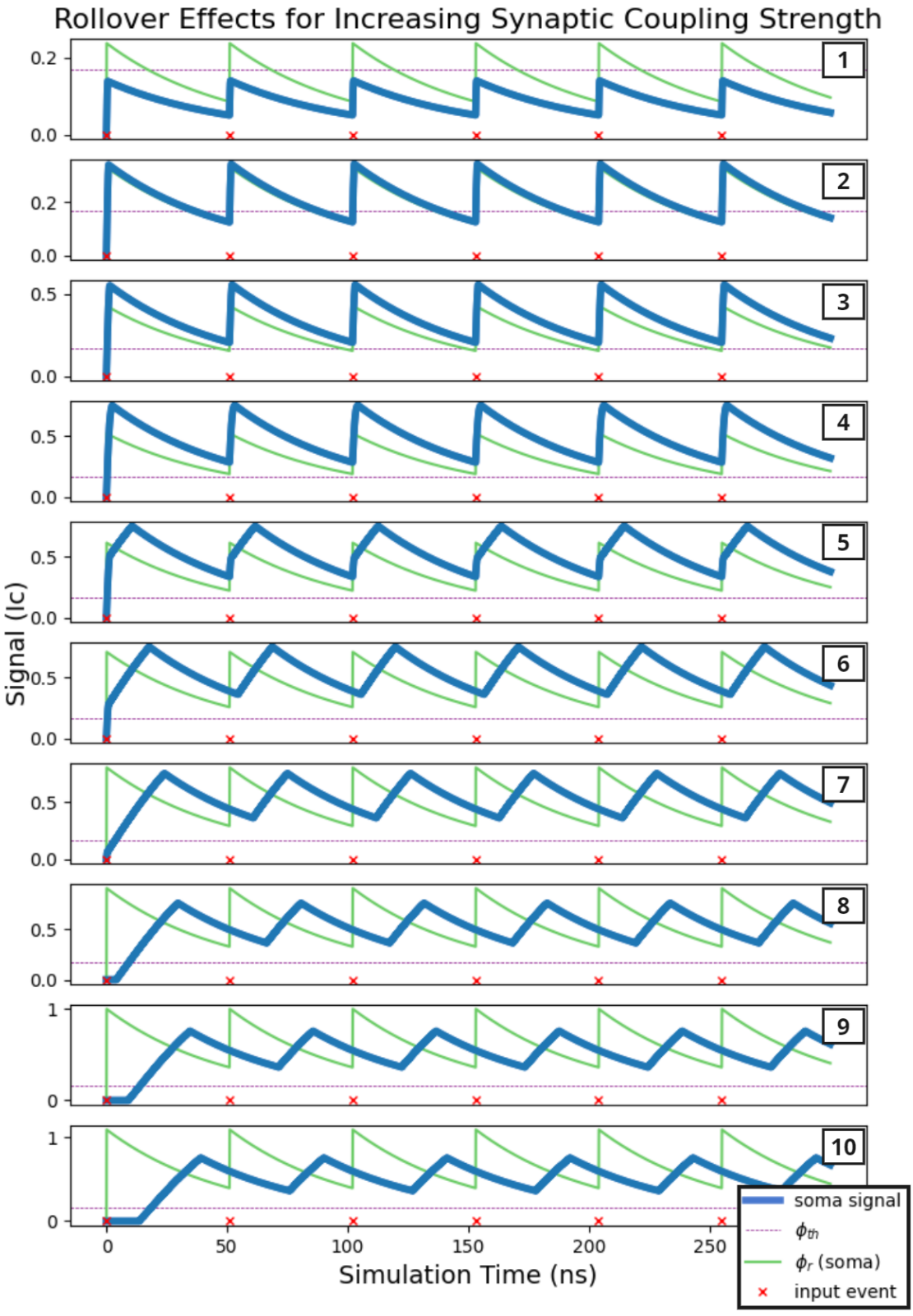


Figure 2.6: From top to bottom, coupling strength from synapse to somatic dendrite is increased in unitless steps of 0.2 from 0.5 to 2.5.

only exploring a diversity of parameters, but also of neuron structures–beyond the point neuron.

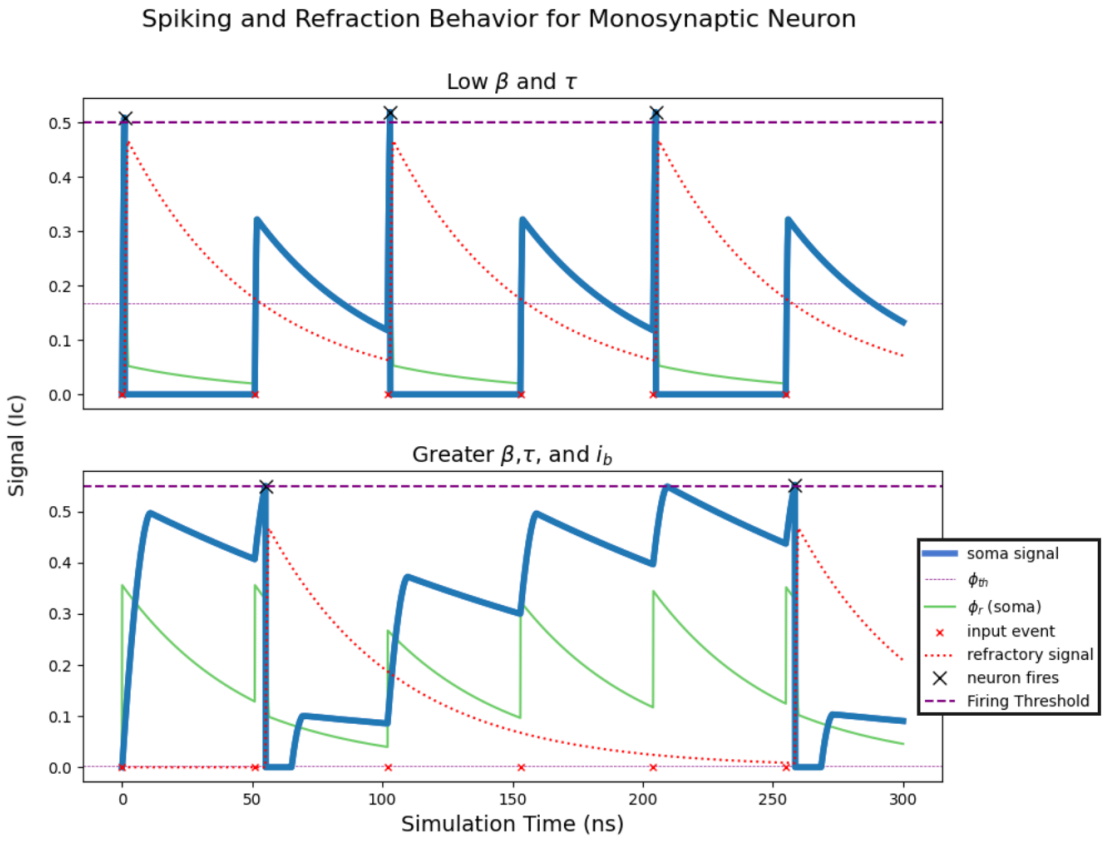


Figure 2.7: Two neurons respond to same input for different combinations of parameter settings.

2.3.5 Dendritic Arbors

All of the above lessons with respect to signal integration in the somatic dendrite (other than firing) are directly applicable to *any* dendrite in the complex dendritic structures that are realizable in SOEN neurons. For more tractable information flow, easier electronic fan-in [124], and more biological plausibility, dendritic arbors are often constrained to a tree-structure, but in practice can adopt arbitrary morphology. An example structure is plotted in Fig. 2.8, and this same structure will be recycled for the subsequent experiments of this paper. In addition to flexible dendritic allocation, so too can synapses be distributed in an arbitrary fashion–one or many to any dendrite, including the somatic dendrite.

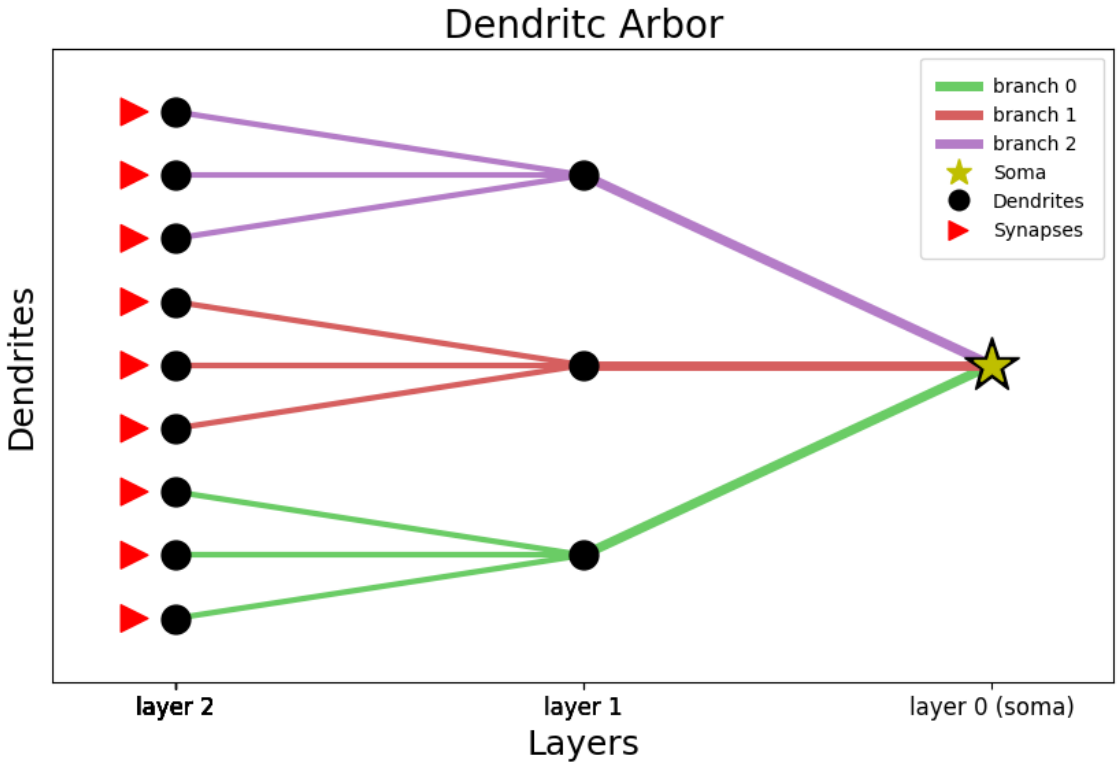


Figure 2.8: Arbor Plot - A simple dendritic tree structure.

2.4 Plasticity and Learning

Given the above computational primitives, and the desired scale of networks, the complexity realizable in SOENs clearly calls for a system of management that goes beyond explicit human oversight. We therefore turn our attention to the preliminary plasticity mechanisms that enable learning in loop neurons at the single and few-neuron scale.

2.4.1 Flux Offset

An advantage to using a tree structure for the dendritic arbor of a loop neuron is that modulating the *effective weight* between two dendrites is then a trivial operation. By effective, we here mean the weight that defines the influence of one dendrite on another, which is the result of both inductive coupling strength and *flux offset*. While coupling strength is determined in hardware fabrication, flux offset can be determined by any mechanism that can couple flux. Simply by adding flux in the receiving loop of the upstream dendrite, any influence of that dendrite unto the downstream dendrite is increased or decreased linearly by this amount. This offset can be in either the positive or negative direction (see Fig. 2.9) and indeed a connection can be 'disconnected' with sufficient negative offset such that no signal will be coupled from one dendrite to another.

Employing flux offset for effective weight moderation allows the use of *memory loops* for storing and applying the offset values. The offset-defining currents in these memory loops can be discretely raised or lowered with ten-bit precision in response to synaptic events, dendritic couplings, or direct electrical input. Therefore, flux offset becomes a viable plasticity mechanism for both local-unsupervised and top-down-supervised learning on chip.

2.4.2 Dendritic Learning: The Arbor Update Rule

To apply flux offset toward a learning task, an update rule must be implemented. While there are many possible strategies, we here posit a simple and effective update plan that exploits the time-continuous information retained in dendrites (even after the soma spikes). In a supervised learning setting, input can drive a certain amount of spiking output from a single loop neuron. Given the label associated with this input, the output can be compared to some *desired* output assigned to that label. We may call the spike difference the error, and straightforwardly multiply this value with the retained signal (or just as well the trial-averaged signal) for each dendrite in the neuron, weighted by some learning rate η. The

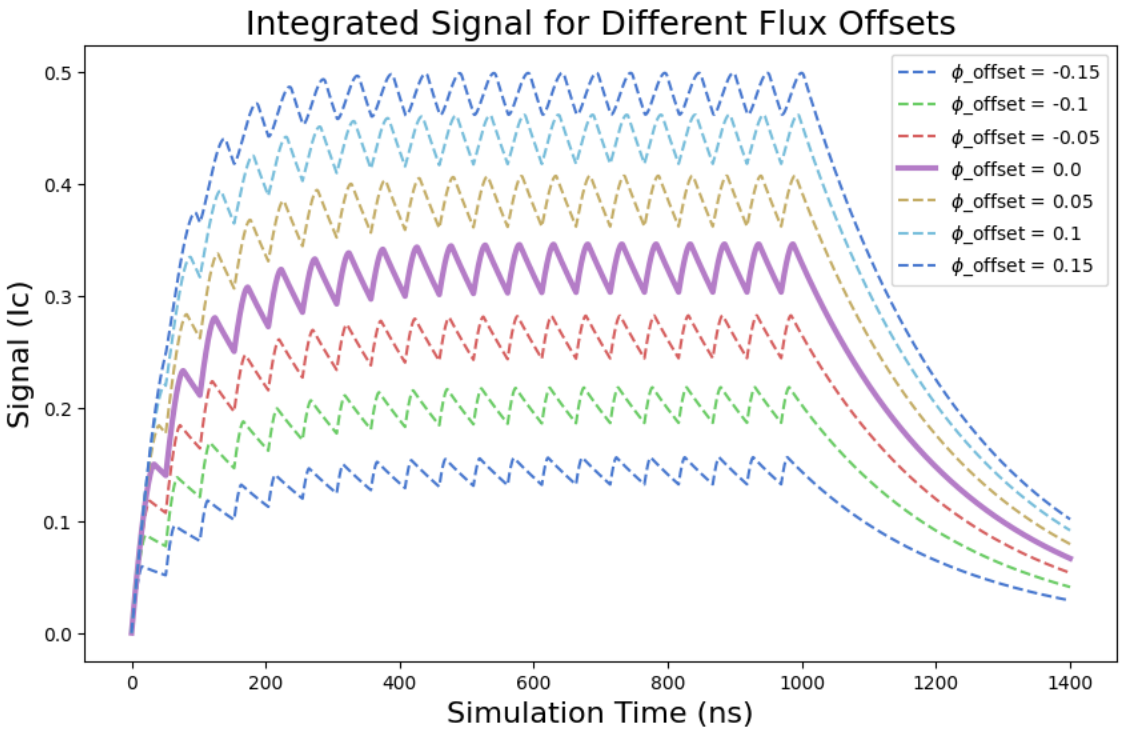


Figure 2.9: Dendritic signal integration for different flux offsets.

arbor-update rule is thus as follows:

$$\phi_{\text{offset}_{i_{t+1}}} \leftarrow \overline{s}_{i_t} \cdot \Delta y_t \cdot \eta \tag{2.3}$$

where i refers to the i^{th} dendrite in the neuron, $\overline{s}_i$ the trial-averaged signal, t the trial in question, Δy the difference between actual and desired spikes, and finally η the learning rate. This procedure may be repeated over a number of iterations until the desired number of output spikes for a given input is achieved. Notably, a single neuron can learn several input-spike relations by iterating over this update sequence for different inputs in a cycle or at random.

There are a number of considerations to implement this rule in an effective way, and these will be investigated in the Section 5.7. All operations of this learning rule are realizable in hardware. Spike differences for the error can be calculated with comparator circuits, mean signal can be approximated with long decay rates, and offset values can be stored and updated with memory loops. Error can be communicated to these memory cells with photons and the remaining update contributions would be accessible locally. We therefore here propose a learning rule intended be fully implementable *on-chip*.

2.5 Experiments and Results

To demonstrate the computational efficacy of single loop neurons, and to explore their dynamical range, we take on a nine-pixel classifier task. This task has already been solved with loop neurons in [122] by analytically choosing a dendritic structure to implement specified logical operations that parse input data into correct classes. We therefore know the task is solvable for this hardware, but would now like to implement the on-chip arbor-update to accomplish the classification task. This simple test case can help explore the implementation of this learning rule and inform on the more nuanced challenges of scaling this approach to problems of greater complexity. Lessons are immediately applied toward the non-linear noisy nine-pixel problem, using a small winner-take-all (WTA) network of mutually inhibitory loop neurons. On this latter task, amendments to algorithm 1 (elastic weight collision and intermittent validation) are as well tested.

2.5.1 Learning The Nine-Pixel Classifier

As a most distilled image classification task, the nine-pixel dataset allows for a tractable problem to be understood by human reason, which can aid in algorithm design. Fig. 2.10 gives an overview. There are three classes to differentiate *z, v,* and *n*, each corresponding

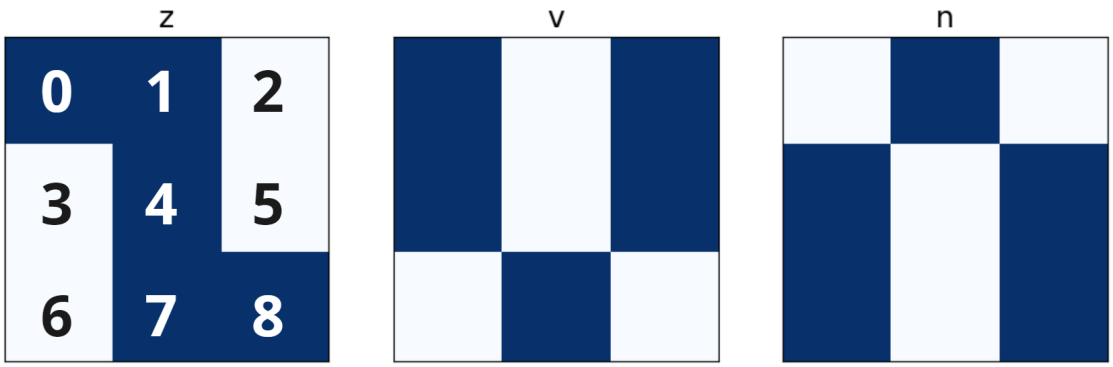


Figure 2.10: The nine-pixel image dataset with letter-labels. Pixel indices are included for the letter z.

to a unique configuration of nine binary pixels that can take the values $\mathbb{W} \in \{0, 1\}$. In our experiment, these images are translated to spiking input simply by having nine-input channels, any of which spikes at some given time value (20ns in this experiment) if the pixel value associated with that channel is 1 (blue in Fig. 2.10) and does not spike otherwise.

To solve this problem, we follow the learning paradigm outlined in Algorithm 1 using the neuron shown in Fig. 2.8, which has an outer layer of nine dendrites, each with a synapse, an intermediate layer of three dendrites, and finally the soma–arranged in a fractal structure. This allots significantly less resources than the logic-neuron used in [122], which has a total of 24 synapses and 22 dendrites (of two types). Neuron parameters are initialized within a dynamical range that results in one output spike for any of the three letters, (see Table 2.1). W_i refers to weights in the i^{th} dendritic layer.

parameter	i_b	τ	β	W_1	W_2	s_{th}	ϕ_{offset}
value	1.8	50ns	$2\pi \times 10^2$	0.5	0.3	0.5	0.0

Table 2.1: Neuron initialization parameters.

Our learning implementation is straightforward. A letter is shown (in the form of spikes) to the neuron. Output spikes are measured, and the error is given by the difference of this to the desired number of spikes for the letter. We assign desired spike-values [0,2,4] to letters [z,v,n] respectively. Every dendrites' flux offset (effective weight) is updated as a function of this error multiplied by its time-averaged signal for that trial and some learning rate ($\eta = 0.02$). All letters are cycled over in order repeatedly until there is no error for any of the three letters. The neuron is then once more shown each of the letters, and its output determines its classification success.

Algorithm 1 The arbor-update Algorithm

```
while convergence==False do
    total error = 0
    for letter in letters do
        run trial with letter as input
        err = desired spikes - actual spikes
        total error ← err
        for dendrites in node do
            φ_offset_{i_{t+1}} ← s̄_{i_t} · Δy_t · η
        end for
    end for
    if total error == 0 then
        convergence = True
```

The output on the test-letters is shown in Fig. 2.11. It can be seen that the number of spikes associated with each letter is completely correct in terms of desired output. It took 32 loops over all three letters (96 iterations in total) to converge on this solution. If the learning rate was too high, or the neuron initialization was at the upper end of the dynamical range (spiking many times for all three inputs in the first trials) the system did not necessarily converge.

A sample of dendritic activity (superimposed on the arbor morphology) is shown for the correct n letter classification in Fig. 2.12. It can be seen (as expected given the convergence values from Fig. 2.9) that the final flux offsets, visible in the pre-input (before 20ns) dotted lines at each dendrite, are not large and it is rather the combination of small offsets that guides the neuron to the correct solution. No rollover or saturation activity is visible (in which cases flux would be greater than signal). However, rollover is an issue for initialization on the periphery of the dynamical range, and for larger learning rates.

2.5.2 The Non-Linear Nine-Pixel with Noise Task

The nine-pixel problem is linearly separable, and in fact information from only two pixels (1 and 4 for example) is sufficient to correctly classify the set. However, when expanding the data set to ten samples for each class by including the original triplet and all possible one-pixel noise permutations, the task is no longer linearly separable and therefore a system must learn more complex data features to achieve correct classification. To meet this challenge, and to further explore the computational space of SOENs, we now implement

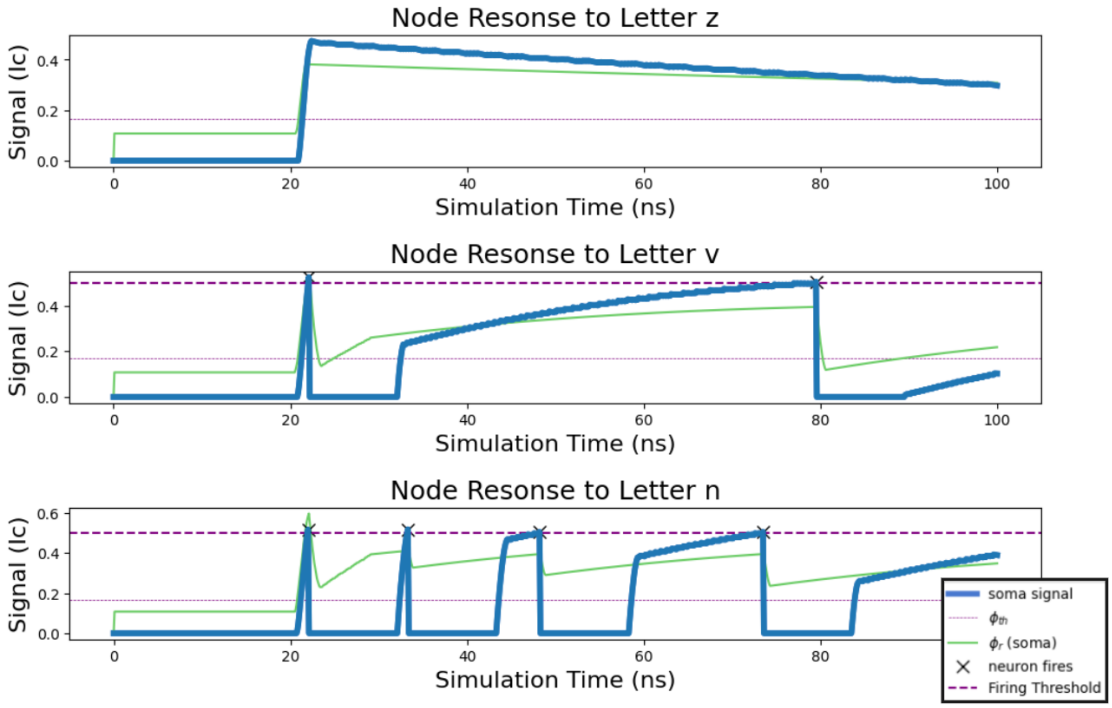


Figure 2.11: Classifier Output for each nine-pixel class.

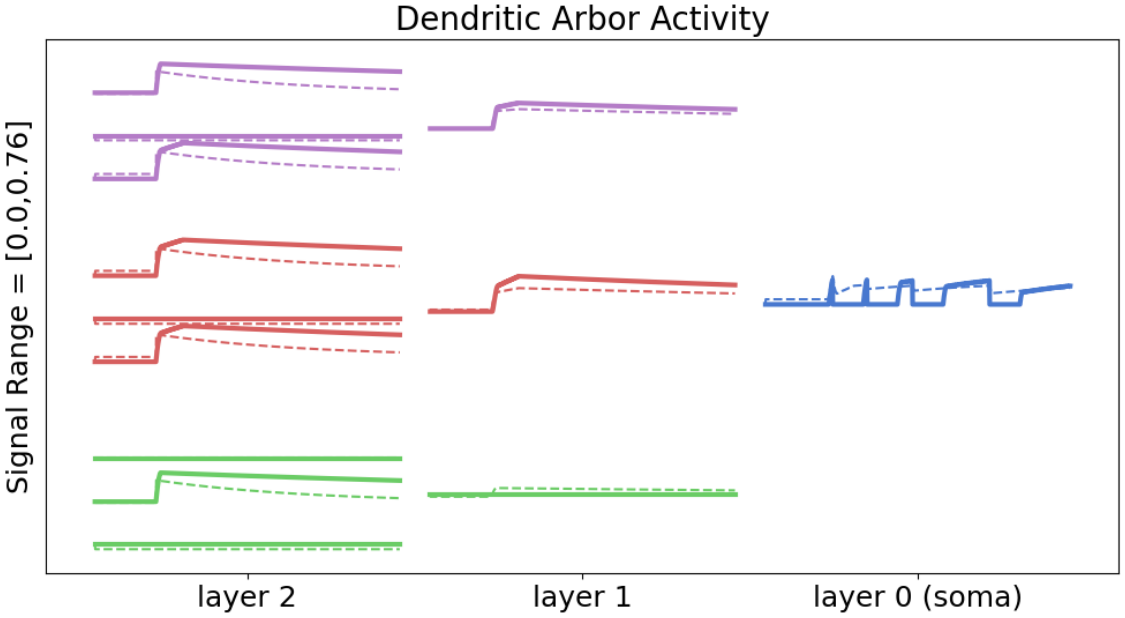


Figure 2.12: Arbor activity for correct n classification. Solid lines = signal, dotted = ϕ_r

a three-neuron mutually-inhibitory (inhibitory synapse on each soma, receiving spiking output from all other neurons) winner-take-all (WTA) network.

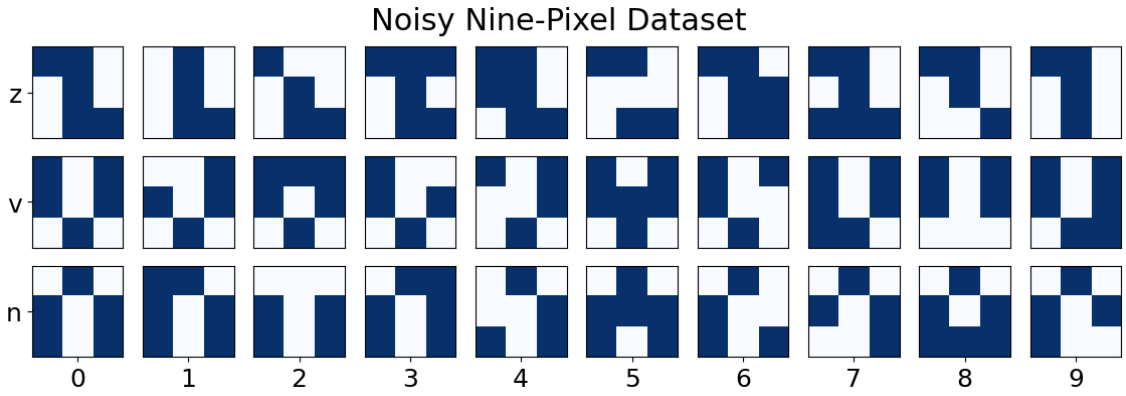


Figure 2.13: Nine-pixel data set with one-pixel noise

Algorithm 1 now makes updates to the dendrites of all three neurons. Because non-converging behavior was already observed in the non-noisy nine-pixel problem, we now investigate potential automated solutions for staying within dynamical range during learning and introduce an *elasticity* parameter. If elasticity is *on*, the update rule will also be a function of flux such that any time the absolute value of trial-averaged received flux $\overline{\phi}_{r_t}$ for a dendrite would exceed 0.5 given the updated offset (thereby entering the rollover regime) the sign of the update is switched. This 'bounces' the effective weight off of the rollover point and back into the dynamical range (here selected to be $\phi_r \in [-0.5, 0.5]$). The update from algorithm 1 becomes

$$\phi_{\text{offset}_{i_{t+1}}} \leftarrow \begin{cases} -\overline{s}_{i_t} \cdot \Delta y_t \cdot \eta & \phi_{\text{offset}_{i_{t+1}}} + \overline{\phi}_{r_{i_t}} \notin [-\phi_0/2, \phi_0/2] \\ \overline{s}_{i_t} \cdot \Delta y_t \cdot \eta & \phi_{\text{offset}_{i_{t+1}}} + \overline{\phi}_{r_{i_t}} \in [-\phi_0/2, \phi_0/2] \end{cases} \tag{2.4}$$

It is expected a bounce will briefly reduce accuracy because it moves the dendritic activity away from the desired output, but will result in an overall more stable convergence. An *inelastic* collision against the dynamical boundaries negates any updates that would push its trial-averaged received flux with proposed offset out of these bounds, expressed by the following update revision to algorithm 1

$$\phi_{\text{offset}_{i_{t+1}}} \leftarrow \begin{cases} 0 & \phi_{\text{offset}_{i_{t+1}}} + \overline{\phi}_{r_{i_t}} \notin [-\phi_0/2, \phi_0/2] \\ \overline{s}_{i_t} \cdot \Delta y_t \cdot \eta & \phi_{\text{offset}_{i_{t+1}}} + \overline{\phi}_{r_{i_t}} \in [-\phi_0/2, \phi_0/2] \end{cases} \tag{2.5}$$

The offsets are not 'stuck' there and can still move back into the dynamical range given subsequent updates. The inelastic regime will too be explored here and finally so will the

unbounded regime where updates are free to take whatever values emerge, as in algorithm 1.

Another experimental condition here examined is *intermittent validation*, where instead of evaluating convergence on performance measured *while* flux offset updates are still being made after every trial, a cycle over the whole 30 sample dataset with *frozen weights* (no updates) is made between every full update cycle. This is done so as to prevent within-cycle updates from vibrating performance around the convergence weights, causing for slower convergence. A correct classification is on a winner-take-all basis, while error-driven updates are derived from the difference between desired and actual spiking. Therefore, it is helpful to orient the convergence requisites based on successful classification rather than the more sensitive error-updates. Note, convergence in both the *update-driven* and the *intermittent validation* schemes are based on 30 correct WTA classifications, but updates are still being made while measuring classification performance after each single-sample trial in the update-driven case.

Thus we have two extra experimental conditions, *elastic weight collision* and *intermittent validation*, the results of which are produced in Table 2.2 for 100 learning epochs each.

	Unbounded	Inelastic	Elastic
Update-Driven	/	/	54
Intermittent Validation	38	6	2

Table 2.2: Total epochs to converge on noisy nine-pixel solution.

Firstly, we observe that elastic weight collision with dynamical boundaries is most favorable for update-driven convergence. In the case of inelastic collision, effective weights may become inert at the dynamical boundaries and in the case of unbounded updating, dendritic activity can be driven to the trough regions of the periodic flux responses from Fig. 2.5 where zero signal will be integrated and the updates thus become zero according to algorithm 1, which may slow opportunity to converge. Secondly, we see that convergence is reached for for all three regimes if the convergence condition is based on intermittent validation. For elastic collisions, the improvement is 27-fold. The two elasticity methods result in similar effective-weight distributions (see Figs. 2.14), where each neuron excites the pixels associated with its class, and depresses otherwise, thereby learning a robust solution. The inelastic case demonstrates how, if not for intermittent validation, the offsets may 'vibrate' around the solution and not necessarily align for a full correct classification cycle while updates are being made.

i_b	τ	β	s_{th}	η
1.8, 2	50, 150	$2\pi \cdot 10^{[2,3]}$	0.25, 0.5, 0.75	0.01, 0.015, 0.02

Table 2.3: Parameter sweep.

Further investigations into a range of parameters (432 combinations, see Table 2.3) reveal that all elasticity regimes (including unbounded) and all validation schemes do often lead to convergence. 187 configurations converged in *under* 25 training epochs. For a (hardware-preferred) bias current of 1.8, of those configurations that converge, elastic and inelastic weight collisions more often achieve 100% classification accuracy in a below-average number of training epochs than the unbounded regime. If the bias current is raised to a value of 2 (though this may create noisy hardware behavior), all elasticity regimes lead to a roughly equal distribution of convergence times. This is likely because the trough region beyond the rollover point is less severe for higher bias current values. In all cases, intermittent validation improves convergence time. Random weight initializations for ten trails each of the settings from Table 2.1 show that only configurations with elastic collisions converge and only when paired with intermittent validation.

2.6 Discussion

In conclusion, we have demonstrated, as a test case, the computational efficacy of a simple on-chip learning rule for loop neurons. To interpret these results, we may borrow lessons from the computational primitives of SOENs. That convergence was not guaranteed for initializations that were on the upper end of the neuron's dynamical range in the linear task or for unbounded updates in the non-linear task, is expected to be a consequence of rollover. Given the dynamics of our learning rules, if a neuron is under-firing in terms of desired output because its received flux has *rolled-over*, the update will erroneously give even more flux to the most active dendrites, potentially pushing the neuron even further into a flux-response trough (see Fig. 2.5) and making firing even *less* likely. This challenge can be mitigated in a number of ways. Firstly, we can initialize in a good dynamical range, which can be determined by setting the parameters to evoke a firing rate somewhere in between the minimum and maximum rate given an input commensurate to the task in question. Another opportunity to remedy this limitation is in circuit design. It is feasible to design circuit mechanisms that bounce back or truncate flux response at the point of rollover (elastic or inelastic weight collisions). We find both methods tend to converge more quickly than unbounded updates. All update regimes benefit from an intermittent validation scheme which better assesses convergence on the whole dataset for a given

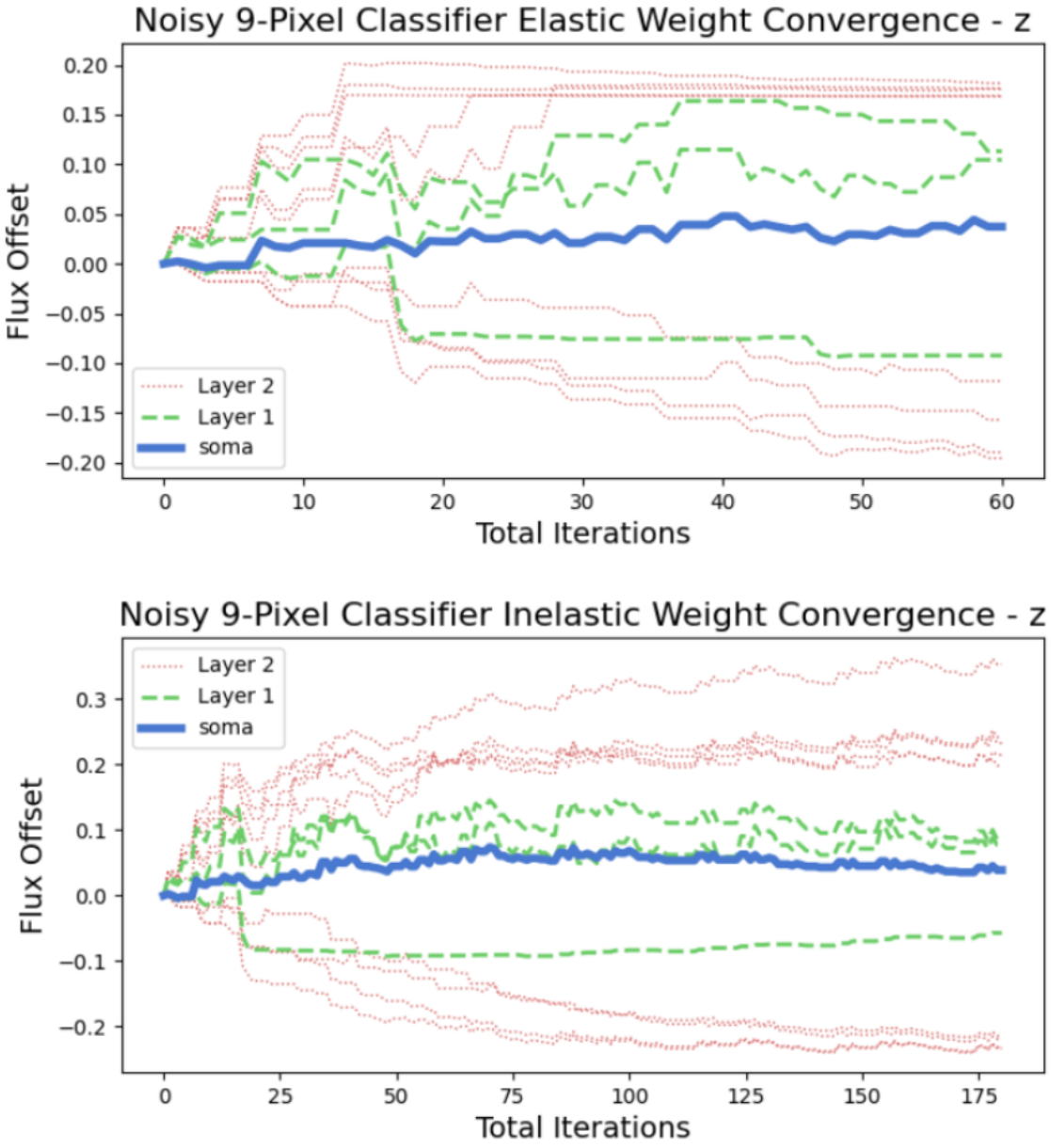


Figure 2.14: Offset flux trajectories for elastic and inelastic neurons.

update cycle. Given a more full parameter sweep over the computational space of loop neurons, however, *we find the arbor-update rule to be generally robust over a wide variety of conditions*. Equipped with these lessons, headway is made on automated on-chip learning for SOEN loop neurons with mechanisms that may very well scale to larger networks. These findings should also be tested on temporal tasks, for which spiking networks and dendritic processing are well-suited. Furthermore, there may be interesting analogues to back-propagation in dendritic learning on loop-neurons. Using precise somatic signal differences (in place of spike differences) may yield a better sensitivity to error. If paired with path-specific updates that trace information backward on a dendritic branch to inform activity-based updates, it may be possible implement a version of the arbor-update rule in a paradigm similar to deep learning. Finally, applying activity-based updates to larger networks of loop neurons should be investigated for a full appreciation of the speed and scalability of SOENs.

2.7 Acknowledgements

This is a contribution of NIST, an agency of the US government, not subject to copyright.

2. THE ARBOR UPDATE RULE FOR IMAGE CLASSIFICATION

3

The Arbor Update Rule for Spatiotemporal Processing

Citation and author list: [125]

Paper Abstract

Superconducting optoelectronic networks (SOENs) utilize the speed and scaling advantages of photonic communication along with superconducting electronics to host brain-like structures and information processing. It is therefore natural for SOENs to draw inspiration from the complexity of the brain to realize useful machine learning capabilities via neuro-inspired algorithms and architectures. The work here leans on the neuroscience of neurons with non-trivial morphologies to implement a mechanism for sequence detection in simulated SOEN neurons equipped with active dendritic arbors. This result is expanded to a more general online pattern-sequence detector and is tested on a nine-pixel binary video feed. A pattern-recognizing layer of neurons acts as a feature filter on an incoming stream of data and feeds into different branches of a temporally sensitive neuron, which will only spike when its branches are excited in the correct order. Temporal sensitivity in arbor branches is a function of connection and leak-rate (time constants), both of which are variable in SOEN hardware. This SOEN-sequence-detector is then successfully applied to an anomaly detection test case and, finally, onto an abridged ECG-5000 time series classification task. These results may stand as a precursor to more sophisticated computations and suggest the viability of SOENs for spatiotemporal data processing.

To design systems with brain-inspired intelligence, neuromorphic engineers must balance the complexity of the brain with what is practical to realize in physical hardware and

there are a number of varied efforts to do so [126–130]. Superconducting optoelectronic networks (SOENs) [131, 132] attempt this balance of complexity by striving to incorporate rich biological realism, while still remaining tractable and trainable.

A key neuro-inspired property of SOENs is its implementation of dendritic trees. Dendritic processing uses physical features of the cell structure to process information and has been shown to perform numerous useful computations [133–141]. It has further been shown to be suitable for neuromorphic learning and memory [142, 143]. The work presented here is a demonstration that specific dendritic arbor architectures can be used to perform pattern recognition and sequence detection. The connectivity graph of a neuron's dendritic arbor is shown to enable single neurons to respond preferentially to a certain input pattern. Leveraging different time constants across dendritic branches facilitates sequence detection. SOENs can host time constants on the order of nanoseconds to milliseconds, thus allowing for considerable temporal sensitivity at a range of scales. The strengths of feed-forward connections between synapses, dendrites, and the soma can be learned with the arbor update rule [108] by weighting a global error signal with local dendritic information.

Section I gives a brief introduction to SOENs and simulation methods. Section II gives an overview of dendritic processing and provides a demonstration of a bio-inspired non-point neuron in SOENs that is sensitive to pattern features and timing. In Section III, a multi-neuron online sequence detection system comprised of a pattern layer and specialized timing neuron is implemented and in section IV it is incorporated in an anomaly detector system. Section V implements the sequence detection system on ECG-5000 time-series data from [144, 145]. Finally, findings and future work are discussed in Section VI.

3.1 SOENs

The monolithic integration of superconductors, semi-conductors, and photonics for neuromorphic computing is motivated by the goals of speed and scale. Photonic communication allows for low-latency, long-distance sharing of information free from electronic parasitics. Photons transmit the neuronal spikes of SOENs. Employing superconducting electronics at low temperature (1K - 4K) allows for synapses sensitive to single photons based on superconducting single-photon detectors [132], as well as dendrites based on superconducting quantum interference devices (SQUIDs) [146] to be implemented. Together with a coupled inductive and resistive circuit, SQUIDs form a leaky-integrator dendrite with known activation thresholds and well-behaved dynamics whose bounds are defined in fabrication. When

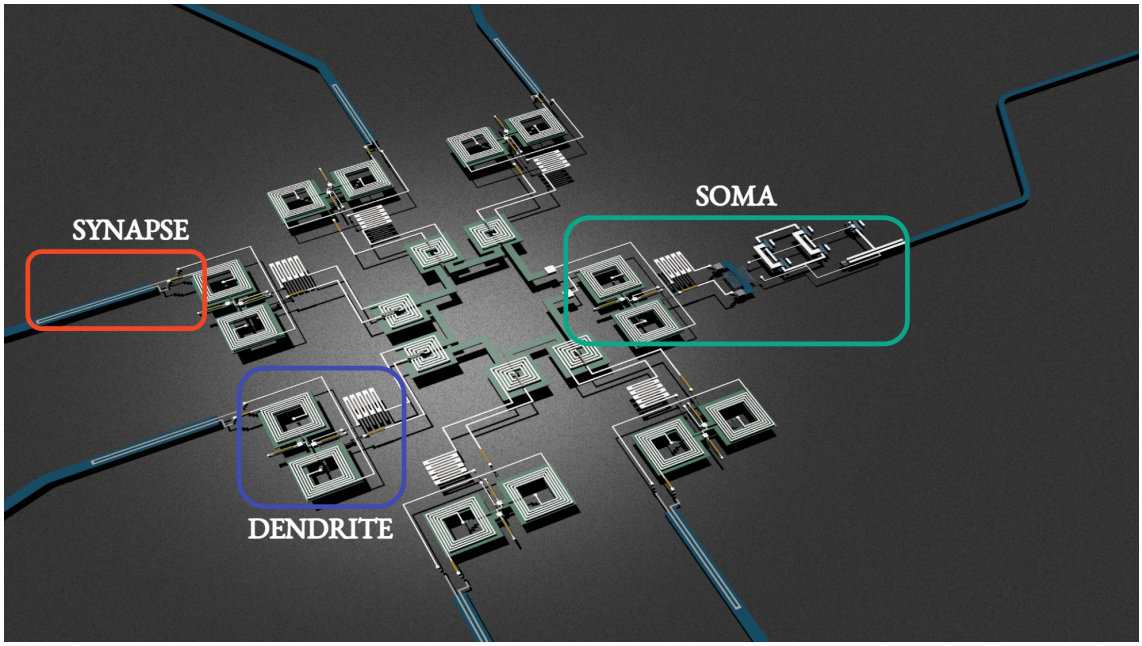


Figure 3.1: A SOENs neuron with six synapse/dendrite pairs (examples outlined in red and blue) and one soma (outlined in green). The figure-eight-shaped coils inside the dendrite region are superconducting quantum interference devices (SQUIDs). The meandering wires are integration loops. The central region of the graphic is a passive collection coil. The remaining hardware in the soma to the right of the SQUID and integration loops is the transmitter circuit.

outfitted with a thresholding circuit and a silicon-diode lightsource, a dendrite becomes a soma. Thus these building blocks (synapses, dendrites, and soma) realize a complete SOEN neuron (Fig. 3.1). All SOEN systems in this paper are simulated using *sim-soens* [147], a high-level instantiation of the phenomenological model of superconducting optoelectronic neurons [148], which captures the behavior of SOEN components with a χ^2 agreement of 10^{-4} to standard circuit equations.

For the purposes of this work, it is sufficient to understand that the activation of any given dendrite or soma is referred to as the signal, s, and is accrued in an element according to the dendritic transfer function approximated in simulation by the forward Euler solution

$$s_{t+1} = s_t \left(1 - \Delta t \frac{\alpha}{\beta} + \frac{\Delta t}{\beta} r(\phi, s)\right) \tag{3.1}$$

where Δt is the simulation time step. α and β are resistance and inductance constants respectively. The source function, $r(...)$, increases monotonically (for bounds determined in hardware and by bias current) with flux ϕ received by the dendrite from either synapse elements or other dendrites. As signal s accumulates in the dendritic integration loop, r saturates, resulting in some maximum possible signal value. Preset neuron parameters and online learning mechanisms are both constructed such that the function $r(...)$ and updates to s remain in a confined range. All quantities are normalized to be dimensionless units of time or critical current.

Qualitatively, SOEN synapses convert photons to flux, or incoming spike-events to integrated dendritic membrane potential. SOEN dendrites receive flux and integrate it into signal according to equation 3.1, and then either couple flux to downstream components, or trigger a transmitter circuit when a somatic dendrite reaches firing threshold. Coupling strengths between components have some predetermined value set by relative inductor sizes and can be described in simulation by

$$\phi_i = \sum_{j=1}^{n} J_{ij} s_j, \tag{3.2}$$

where the received flux ϕ of the i^{th} dendrite is determined by the integrated signal of the j^{th} dendrite according to the coupling strength J_{ij}. This value can also be dynamically offset in the positive or negative direction by coupling additional flux of a changeable value stored in a memory loop (a superconducting current in a loop) [149]. The value in a memory loop can be changed with either photons or flux. With this scheme, SOENs can learn using methods that should be entirely implementable on-chip [108].

3.2 Dendritic Processing

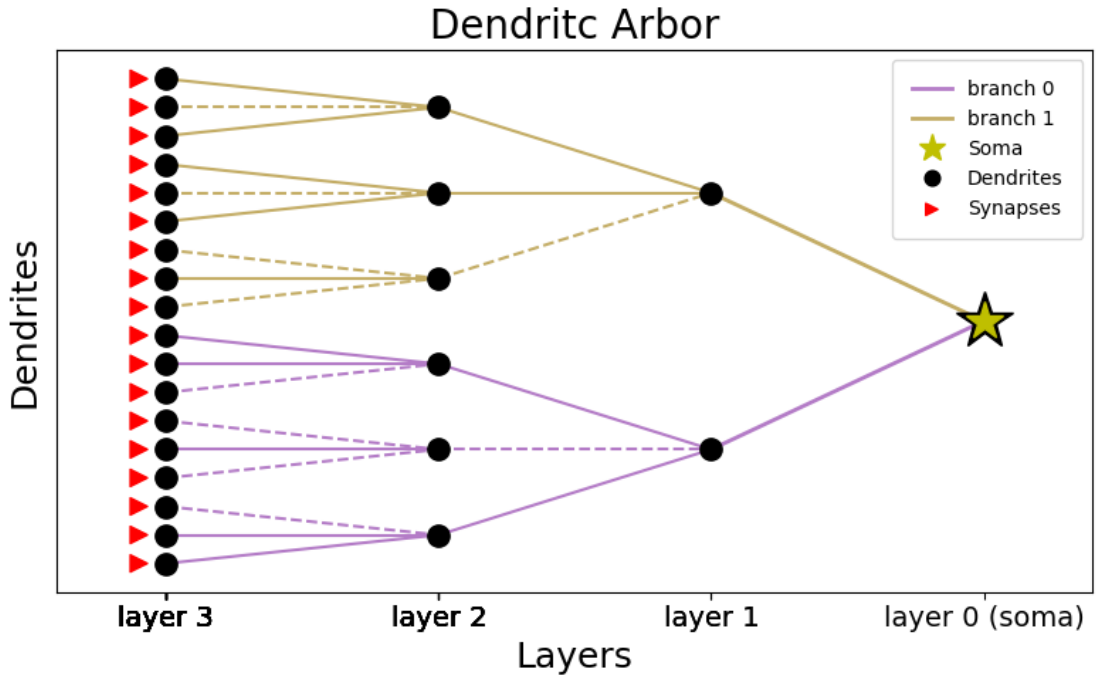


Figure 3.2: The morphology of a pattern-recognizing basal-proximal neuron with hand-selected weights to resonate with the $z-n$ nine-pixel pattern sequence. Dotted lines indicates inhibition. It can be seen that the purple branch (branch 0) is designed to be excited by the z pattern. An unraveled z image fed into the synapses would result in excitation at the pixel associated dendrites and inhibition otherwise. Likewise, the gold branch (branch 1) is designed to respond to the n pattern. The distance of the basal branch (purple, branch 0) to the soma is simulated with a weaker connection strength. The proximal branch is more strongly connected (gold, branch 1).

Biological neurons are comprised of many integrated parts coming together in complex, meaningful structures capable of computations within every neuron. While the specific role of certain neuronal sub-structures is not always clear, dendritic processing has been shown to facilitate context-learning and sequence recognition [134, 150, 151]. Relevant to the work presented here, there is evidence for *basal* dendritic branches responding to specific contextual keys, priming (but not alone causing) a neuron to fire. Once in this context-primed state, event-triggered excitation in a *proximal* dendritic branch may provide the remaining membrane potential to cause neuronal firing. In this model, neither context nor event are sufficient to cause firing alone. Moreover, these basal and proximal branches

must be excited in the correct order to result in firing. An example of a SOEN neuron with an analogous structure (henceforth referred to as a basal-proximal neuron) is depicted in Fig. 3.2. All nine possible two-pattern sequence scenarios for the nine-pixel images Z, V, and N are plotted in Fig. 3.3.

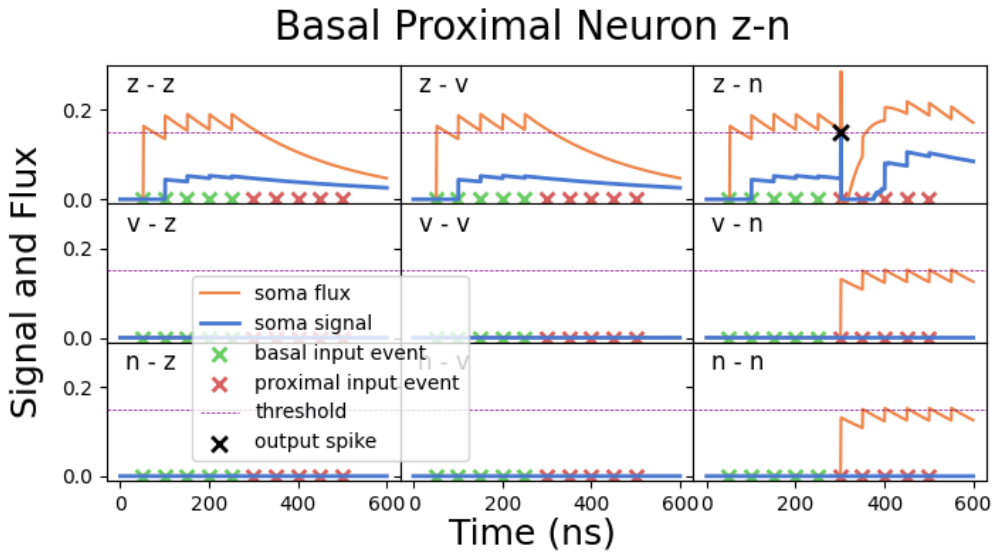


Figure 3.3: The nine possible basal-proximal orderings for a two pattern sequence drawing from three possible patterns. Only one order, $z \rightarrow n$, results in the somatic signal (blue) reaching firing threshold. The context-primed state can be seen in the received-flux (orange line) when the z pattern arrives first and the event-triggered state can be observed when the n pattern arrives second. Neither alone is sufficient, given the flux-to-signal dendritic transfer function, to drive the somatic signal to threshold.

Basal context and proximal events can be learned using the arbor update rule [108]. By propagating a spike difference error globally and weighting it with local dendritic information, the plasticity mechanism of loop dendrites [149] can be updated such that both context and event are learned, the results of which are plotted for the more difficult extension of five possible input patterns in Fig. 3.4. All possible sequences of the input patterns displayed in Fig. 3.5 are cycled over iteratively, with target spiking set to zero for all but the sequence to be recognized, which is assigned a target spiking value of ten. The error between the actual quantity of output spikes and the target quantity (Δy^{trial}) is propagated back to all dendrites in the neuron where it is multiplied by the local signal information (trial-averaged signal $\overline{s}_i^{\text{trial}}$), and weighted by some learning rate η, to determined the new

flux offset ϕ_{offset} value at each dendrite, according to

$$\phi_{\text{offset}_i}^{\text{trial}+1} \leftarrow \bar{s}_i^{\text{trial}} \cdot \Delta y^{\text{trial}} \cdot \eta \tag{3.3}$$

for some i^{th} dendrite after a trial iteration. This is the dynamic weighting scheme of loop neurons, where a value is stored in a memory loop (of superconducting current) and used to offset the flux of that dendrite in the positive or negative direction. In this way, arbor updates can be made online and stored in hardware.

Figure 3.4: Signal in the soma for an $n - x$ trained neuron given different input sequences. Each grid cell here corresponds to the average excitatory flux at the soma of a basal-proximal SOEN neuron. Basal input is received first, with a total of five pattern-specified input synaptic events, followed by five proximal input events. This is after learning the $n - x$ sequence with the arbor update rule in only 7 training epochs. The correct ordering of input events indeed results in the most somatic activity and also the most output spikes.

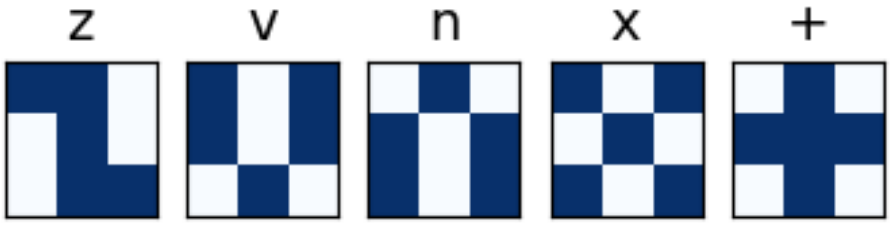


Figure 3.5: These patterns correspond to input of nine dimensions, where dark pixels equate to some amount of input spikes (depending on the task) and blank pixels do not fire. Usually, one pixel channel feeds into one synapse on a neuron.

3.3 Online Sequence Detection

The prime-then-trigger evocation of the above basal-proximal neuron inspires a more general sequence detection system in SOENs, whereby the partitioning of input into different branches is automated (rather than fixed) by a layer of *pattern neurons* that then feeds into a basal-proximal-like *timing neuron*. These pattern and timing neurons together form a SOEN pattern-sequence detection module, each component of which are here examined individually before being tested together. The main idea is that pattern neurons filter spatial features and timing neurons filter temporal features.

3.3.1 Pattern Neurons

As demonstrated in [108] and shown in Fig. 3.6, simple binary images can easily be learned in SOEN neurons using the arbor update rule, even in the presence of noise. A layer of neurons, each associated with one input pattern, and connected to each other with mutual inhibition, can serve as a classifer in a winner-take-all (WTA) scheme, whereby the pattern associated with the highest-spiking neuron is designated as the output prediction. In some sense, the mapping of a binary image to some spiking output is a dimensional reduction according to a pattern-based filter that selects for certain features much like a convolutional kernel would when scanning over an image. For a nine-pixel grid, filters associated with various spatial patterns can be hand-picked (as in Fig. 3.3) or learned (Fig. 3.4). The underlying mechanism is that dendrites associated with a specific pattern feature are excited and the rest are depressed. This results in high firing activity for a given input pattern and little-to-none otherwise. A layer of various possible sequence pattern-recognizing neurons can therefore be constructed such that firing activity can automatically

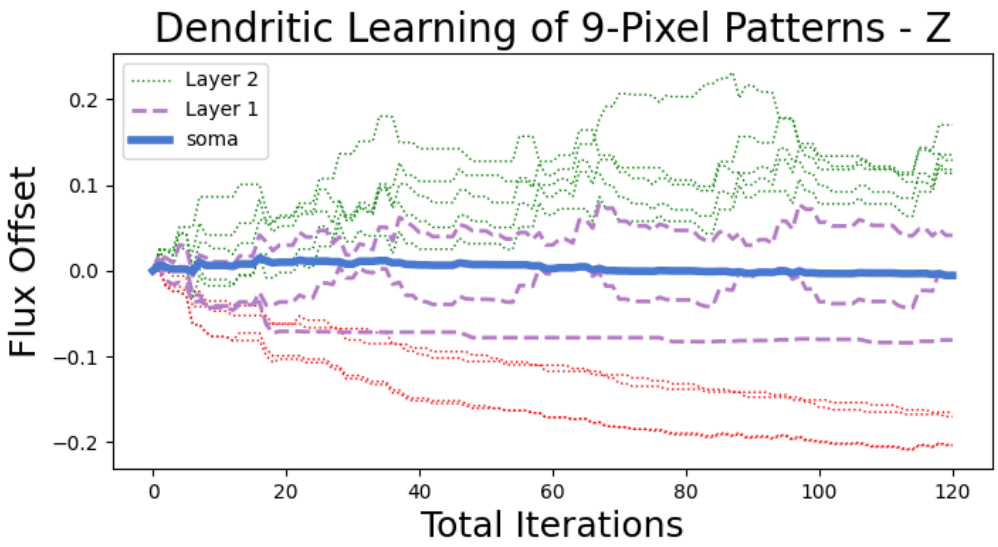


Figure 3.6: An example of a neuron with an arbor of 13 dendrites (fan-in of three), learning to recognize the z nine-pixel pattern, even with the presence of one-pixel noise, among noisy samples of v andg n patterns, as detailed in [108]. This neuron is in a mutually inhibitory winner-take-all context with neurons corresponding to v and n patterns. All neurons learn to spike in response to their associated pattern and remain relatively quiescent otherwise. At the synaptic layer, it can be seen that dendrites associated with the z input pixels (green) are strengthened and dendrites associated with empty pixels (red) are depressed.

be filtered down different network paths according to spatial features.

3.3.2 Timing Neurons

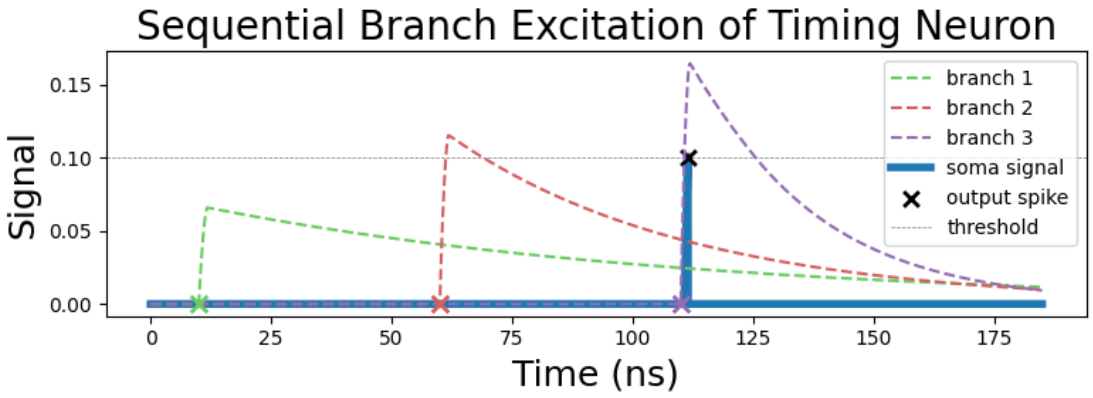


Figure 3.7: A timing neuron driven to firing by the correctly ordered excitation of its branches. The somatic signal (solid blue line) is fed by the signal of all three dendritic branches (dotted lines). Each dendritic signal is weighted by its connection strength to the soma, so that their relative contributions to firing can be observed. Input events for synapes are marked with the color corresponding to their branch.

The broad range of time constants (leak rates) definable in SOEN dendrites may be exploited to construct neurons whose branches must be excited in the correct order to reach firing threshold. This can be achieved by constructing a neuron with a number of single-dendrite branches equivalent to the length of the sequence to be recognized. Assuming branches are excited one at at time, manipulating the leak rates and coupling strengths of each branch can allow for an implicit excitation order required to fire. Choosing the leak rate τ to be approximately equivalent to the duration of a sequence pattern input $\Delta t_{\text{pattern}}$ multiplied by the number of branches b in the neuron minus the order placement i results in the soma only being able to receive input from all dendrites simultaneously if they are excited in the right order. The weight of that dendritic branch follows an inverse trend, increasing with the order it is to be excited, such that if the first branch is excited many times in a row it will not be strong enough to drive the soma to fire alone. Similarly, even though the last branch has a strong weighting, it cannot drive the soma to fire because its signal decays before the next input spike arrives. Thus, branch leak rates and strengths can be fixed in hardware fabrication with consideration for the following relationships:

$$\tau_i \approx \Delta t_{\text{pattern}}(b - i), \tag{3.4}$$

$$w_i \propto i. \tag{3.5}$$

The dendritic arbor activity for a correctly ordered excitation of timing neuron branches is pictured in Fig. 3.7. Note that due to the different time constants derived from equation 3.4, only that order of branch excitation can result in firing because only branch A has a long enough time constant τ_A to be excited first and still contribute to somatic flux (and therefore signal) during the remaining two branch excitations. Even if the branch order were to be ACB, the time constant associated with branch C is too short to allow it to contribute to the soma while branch B is being subsequently excited. Because of equation 3.5, two or three A patterns in a row is also not sufficient for firing. Their activity would overlap, but not be strong enough in total to reach threshold. The key to the timing neuron firing is for all three branches to contribute to the flux received by the soma *simultaneously* and therefore the only viable order of excitation is A, B, C.

Similar to pattern neurons, the timing neuron can act as a filter on input data, but now in the time domain. Together, pattern and timing neurons equip SOENs with a filtering system that can be responsive to both specific spatial and temporal signatures. By combining these computational assets, an online spatial-pattern sequence detector may be instantiated.

3.3.3 SOEN Pattern-Sequence Detector

The architecture of the pattern-sequence detector (Fig. 3.8) is such that all pattern neurons are adjacent and not mutually connected. The number of possible input patterns determines how many neurons there ought to be in the layer (one-per-pattern). A stream of consecutive patterns is fed indiscriminately to all pattern neurons, each with a synaptic layer of equivalent size to input dimensionality (one pixel feeds to one synapse in each pattern neuron, a dark pixel corresponds to spiking input). Each pattern neuron can either be trained with the arbor update rule or handcrafted. Only the neuron associated with the presented pattern should be excited enough to fire. The timing neuron has as many mono-dendritic branches as there are patterns in the desired sequence to be detected. This number need not be equivalent to the number of possible patterns or the total length of the input sequence. The branches have an implicit excitation order. Depending on the desired sequence, the appropriate pattern neurons are connected to the appropriate branches. Because only the correct associated pattern neuron should fire in the presence

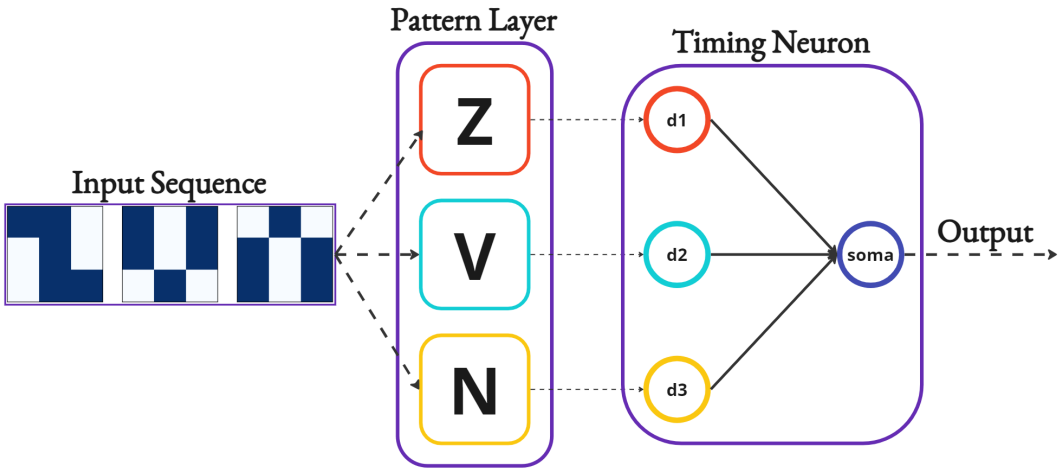


Figure 3.8: Pattern-Sequencer System: Dotted arrows are photonic firing, solid arrows are dendritic arbor connections.

of a given input pattern over some time, the corresponding branch (if there is one) of the timing neuron will be excited if that pattern is occurring in the input. Like the previous basal-proximal examples, if it is the first branch, the neuron enters a contextually primed state. This level of priming increases if the correct order of branches are excited, and leaks away otherwise. If all up to the penultimate branch have been excited in the correct order, the final triggering event of the pattern associated with the final branch will evoke the firing of the timing neuron and the sequence will have been detected.

Qualitatively, the task of the pattern-sequence detector is to fire when it has just received a specific sequence of input patterns. It should fire under no other conditions. The results of all possible three-image input pattern sequences (with three consecutive input events for each image), filtered by a pattern layer into a timing neuron are documented in Table 3.1. It can be seen that however many times a given pattern occurs in a sequence, the corresponding pattern neuron fires the same amount of times. This verifies that the pattern layer is filtering the input correctly. However, the timing neuron fires only when the correct sequence of input patterns is seen (in this case, z-v-n). Thus, after the temporal filter is applied to the already spatially filtered patterns, the correct sequence is found.

The timing neuron activity is shown in Fig. 3.9 for the case where the desired sequence is observed. The dendritic activity (dotted lines) is particularly demonstrative of the descending time-constant strategy, where a spike only occurs when activity from all branches overlap, due to the first branch integrating all the way from the start of the trial, the

Sequence	$[Z, V, N]$	Timing Neuron
Z - Z - Z	[3, 0, 0]	0
Z - Z - V	[2, 1, 0]	0
Z - Z - N	[2, 0, 1]	0
Z - V - Z	[2, 1, 0]	0
Z - V - V	[1, 2, 0]	0
Z - V - N	**[1, 1, 1]**	**1**
Z - N - Z	[2, 0, 1]	0
Z - N - V	[1, 1, 1]	0
Z - N - N	[1, 0, 2]	0
V - Z - Z	[2, 1, 0]	0
V - Z - V	[1, 2, 0]	0
V - Z - N	[1, 1, 1]	0
V - V - Z	[1, 2, 0]	0
V - V - V	[0, 3, 0]	0
V - V - N	[0, 2, 1]	0
V - N - Z	[1, 1, 1]	0
V - N - V	[0, 2, 1]	0
V - N - N	[0, 1, 2]	0
N - Z - Z	[2, 0, 1]	0
N - Z - V	[1, 1, 1]	0
N - Z - N	[1, 0, 2]	0
N - V - Z	[1, 1, 1]	0
N - V - V	[0, 2, 1]	0
N - V - N	[0, 1, 2]	0
N - N - Z	[1, 0, 2]	0
N - N - V	[0, 1, 2]	0
N - N - N	[0, 0, 3]	0

Table 3.1: Col 1. The sequence of input patterns
Col 2. The firing of each pattern neuron respectively.
Col3. The firing of the timing neuron.

second from the middle, and the last only in the final third, each with increasing strength. Note the similarity of this result to the contrived timing neuron example shown in Fig. 3.7. The improvement is that the detection system did not require the patterns to be hand-fed to specific branches in order, but were rather filtered and routed by the pattern layer neurons. A viable sequence detection method, with automatic spatial and temporal filtering, has thus been demonstrated in SOENs, using design and training methods simple enough to realize on chip.

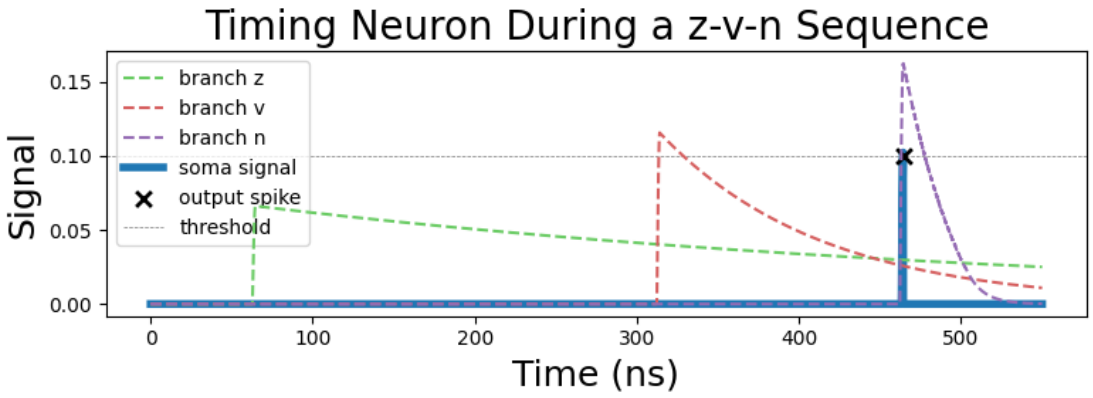


Figure 3.9: Internal activity of timing neuron when system encounters correct sequence, as filtered through a pattern-processsing layer.

3.4 Anomaly Detection

An application of the above sequence detector would be that of anomaly detection, whereby aberrations in a known sequence elicit an output from a system that is quiescent otherwise. Figure 3.10 outlines an architecture for this task. By having the output of the sequence detecting neuron depress one branch of an 'anomaly detector' neuron, and by having the other branch be fed directly by sequence input, excited by any pattern, the neuron can be constructed such that this balance rests at the edge of spiking. If the desired sequence is not repeatedly recognized, the timing neuron momentarily ceases to fire, and the excitatory branch of the anomaly neuron wins out, driving it to fire, as shown in Fig. 3.11. This mechanism of creating a tension between excitatory and inhibitory branches in a single neuron may be indicative of general perturbation-detection capabilities.

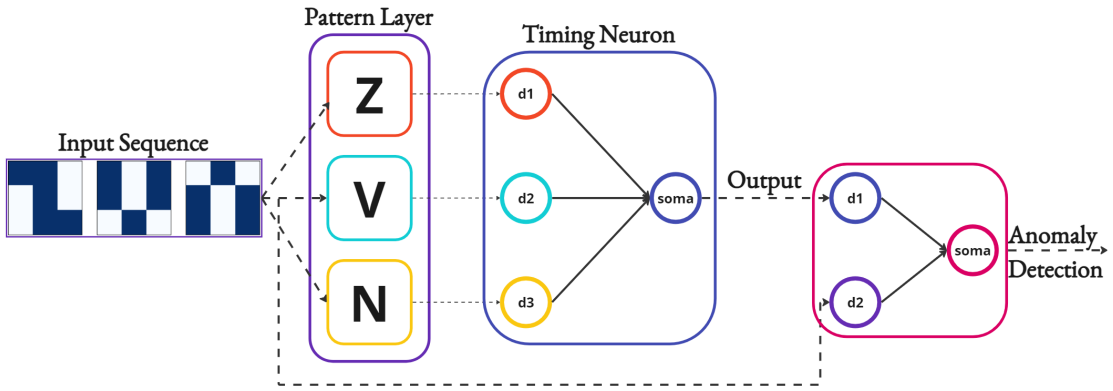


Figure 3.10: A pattern sequence detection module feeding into one branch of an anomaly detection neuron. The other branch is fed directly by the pattern sequence. The two branches are in excitatory/inhibitory tension.

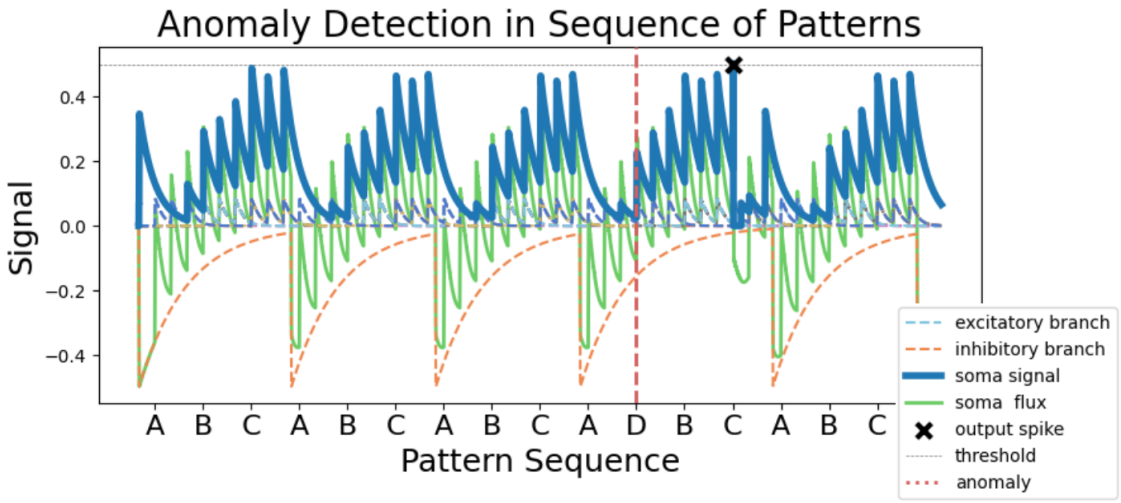


Figure 3.11: Somatic and arbor activity of an anomaly detection neuron when encountering one abberation amidst an otherwise correctly repeating sequence. The solid blue line is somatic signal and the solid green line is received flux (of the soma from the two dendritic branches). Dotted signals are of the dendrites, and the orange dotted signal is specifically from the sequence-detector-fed branch. Each label on the x-axis corresponds to three input events of that pattern, separated by 50 ns, fed into the sequence detector pattern layer and the pattern input branch of the anomaly neuron. The sequential lettering of $ABCD$ corresponds to to the nine-pixel patterns $zvnx$. One dummy spike is fed into the pattern input branch of the anomaly neuron at the start of the trial, as there were no sequences yet to detect. The anomaly neuron only fires after encountering the aberration D, marked by the vertical red dotted line.

3.5 ECG Time Series Classification

Nine-pixel datasets not only offer tractable dynamics for didactic purposes, they also can inform algorithm development for more scaled tasks. The above presented methods are now applied to a time-series classification task on an electrocardiogram (ECG) dataset [144,145], wherein anomalous heartbeats should be distinguishable from healthy heartbeats. While the time scale of a heartbeat should be within range of time-constants achievable in SOENs, the data is here sped to the order of nanoseconds to showcase SOENs extremely fast processing speeds (a scale for which appropriate datasets are less readily available). The ECG-5000 dataset is used in a binary classification scheme that calls for the separation of healthy and anomalous heartbeats. These two time-series classes (and their means) are visualized in Fig. 3.12. A natural partitioning of these signals in the time domain is into thirds and these temporal chunks can be used as distinct patterns themselves. This allows for an easy mapping onto the sequence detection system from Fig. 3.8.

To convert continuous time series data such that its information can be spread across a dendritic arbor, signals can be binned along the y-axis (Voltage in this case), as depicted (for the mean signals of each heartbeat class) in Fig. 3.13. It can be seen that the time-series structure is preserved (albeit in lower resolution) by this prepossessing method. Each step in time is represented by a one-hot-encoded sixteen-dimensional vector corresponding to the signal amplitude at that moment. The binned signals can be used to generate multi-channel spike-trains according to the SOENs synapse constraint of a 35 ns reset time (Fig. 3.14). For a given amplitude vector, if a channel has a value of one at its associated binned-signal index, a spike is added—so long as there is not already a spike present in that channel from within the last 35 ns.

To analyze this preprocessing method, the spiking input-channels can be reshaped into pixel images and the average amount of spikes for each pixel of every chunk over both classes can be plotted as a four-by-four pixel image, as shown in Fig. 3.15. It is immediately clear that there is considerable overlap across classes and across chunks. Chunk B occupies identical amplitude bins in both normal and anomalous heartbeats. Moreover chunk B images can be construed as a subset of either A or C chunks of either class. This overlap of classes and chunks may suggest that both the number of bins and the number of chunks could be better optimized for. There are likely chunking and binning numbers that resonate with natural features in any given dataset. However, this simplified arrangement is straightforward to imagine computationally and is therefore appropriate for the purposes

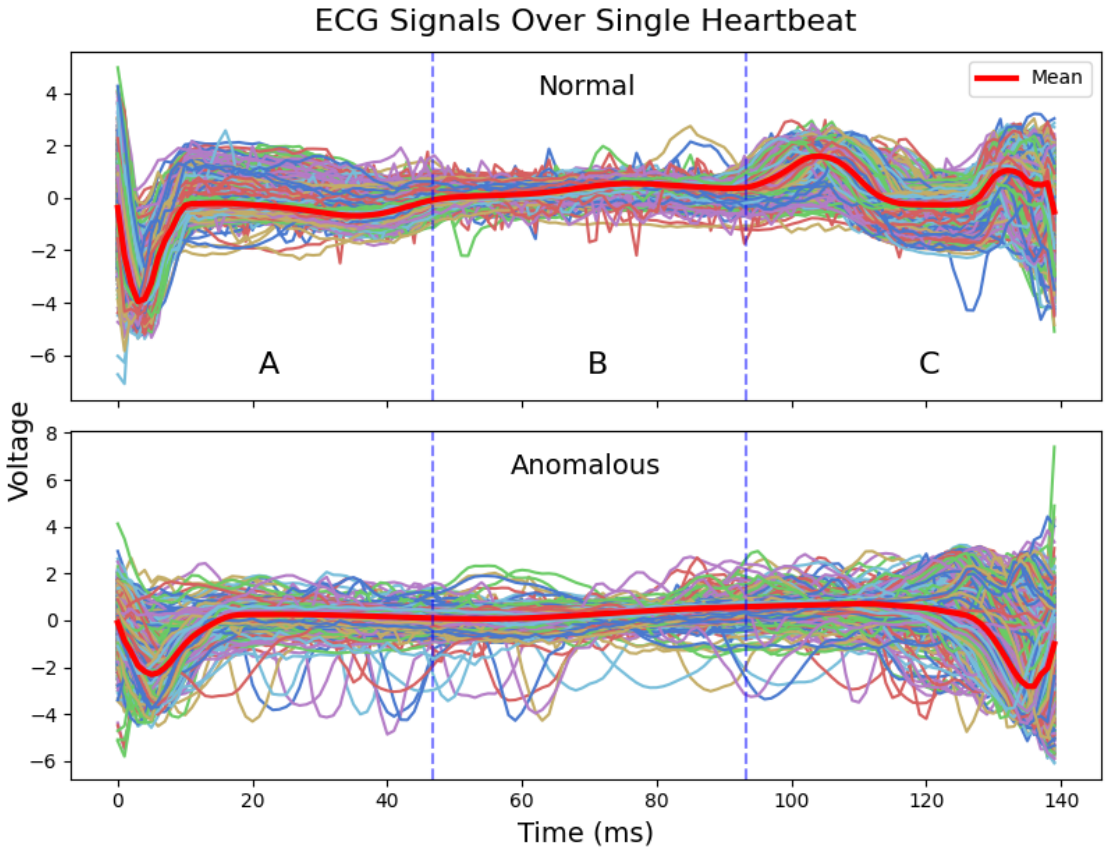


Figure 3.12: Normal and anomalous ECG-measured heartbeats. Each signal is a 140 ms splice from a stream of heartbeats belonging to its respective class. Regions **A**,**B**, and **C** are of equal length in the time dimension and can be used for natural temporal chunking of the data.

of this example. Furthermore, this preprocessing method may be achievable in hardware through an array of thresholding mechanisms.

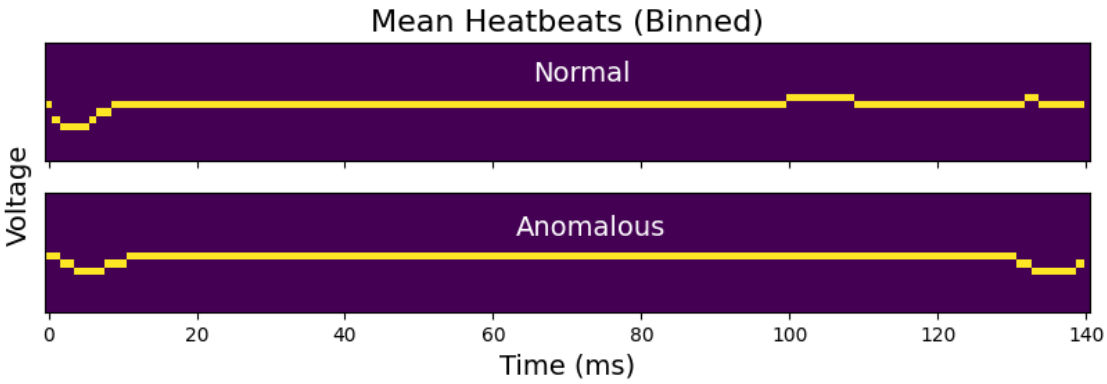


Figure 3.13: Binned ECG signal data for the mean signal of both normal and anomalous heartbeats. Each step in time corresponds to a one-hot-encoded vector of the signal amplitude at that moment in time.

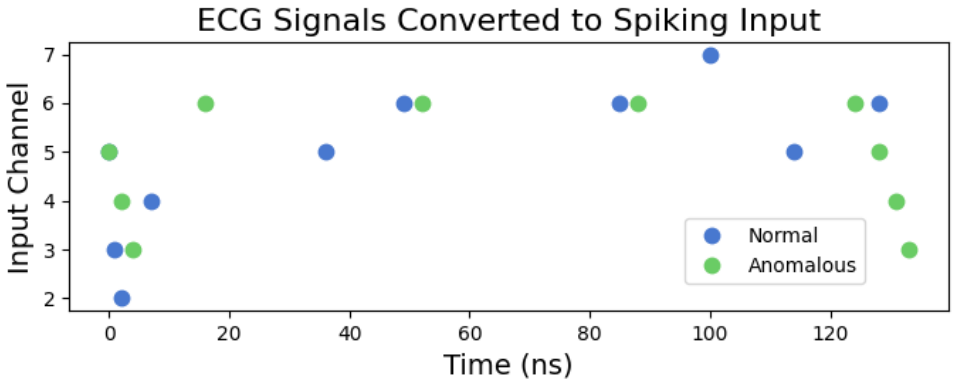


Figure 3.14: Raster plot of sixteen input spike-train channels converted from the binned amplitude signals of the mean signals for normal and anomalous heatbeat signals from Fig. 3.13. Interspike intervals is adapted to appropriate scales for SOENs computing (nanoseconds).

Despite a dataform with notable inter-class overlap, the SOENs sequence detection system manages to correctly distinguish head-to-head comparisons of normal and anomalous heartbeats with a 93% test accuracy on 4000 signals using spike-first comparison on the the timing neuron output after only one epoch of training. Pattern neurons are trained on chunks A, B, and C for the first 3200 samples of the dataset and then tested with a timing neuron output on 800 remaining samples. It is found that the constraints of equations

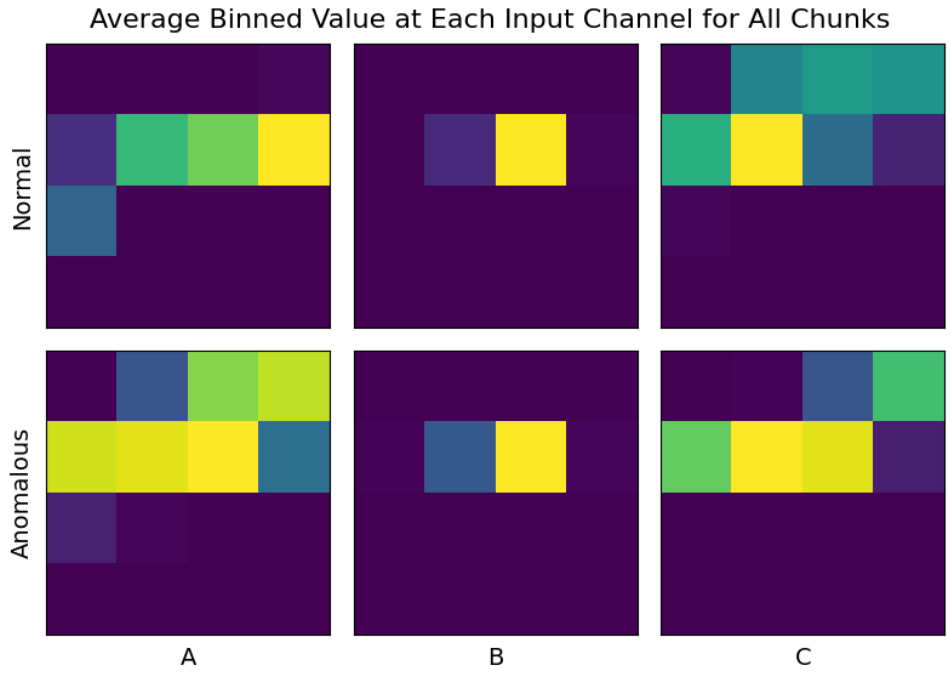


Figure 3.15: Each pixel corresponds to the amount of spikes encountered at one of the sixteen input-channels for the signal-to-spike converted ECG of normal and anomalous heartbeat signals across all pattern chunks.

3.4 and 3.5 should be relaxed to enable occasional firing for out-of-order branch excitation (in particular of B and C). Because of class overlap, it is less straightforward to train a neuron that responds to B-chunks more rapidly than to C-chunks without the inclusion of inhibitory dendrites, which have been excluded from this work for simplicity. Inhibition does typically allow for better class separation in SOEN neurons. Allowing an out-of-order firing of B and C pattern neurons to elicit a spike in the timing neuron accommodates more robust time-series classification. Fig. 3.16 demonstrates how this adjustment results in a sequence detection system that spikes sooner and more often when presented with a normal heartbeat than with an anomalous heartbeat. Weaker spiking activity for anomalous spans in a time-series can likely be leveraged for a full anomaly detection system in exactly the way as implemented in Section IV. It is also worth noting that the spatial-then-temporal processing of the SOEN sequence detection system does appear to be necessary, as a single SOEN neuron using the arbor update is unable to distinguish between the two signal types. Therefore, the pattern-layer-to-timing-neuron architecture must be capturing features that are necessary for classification.

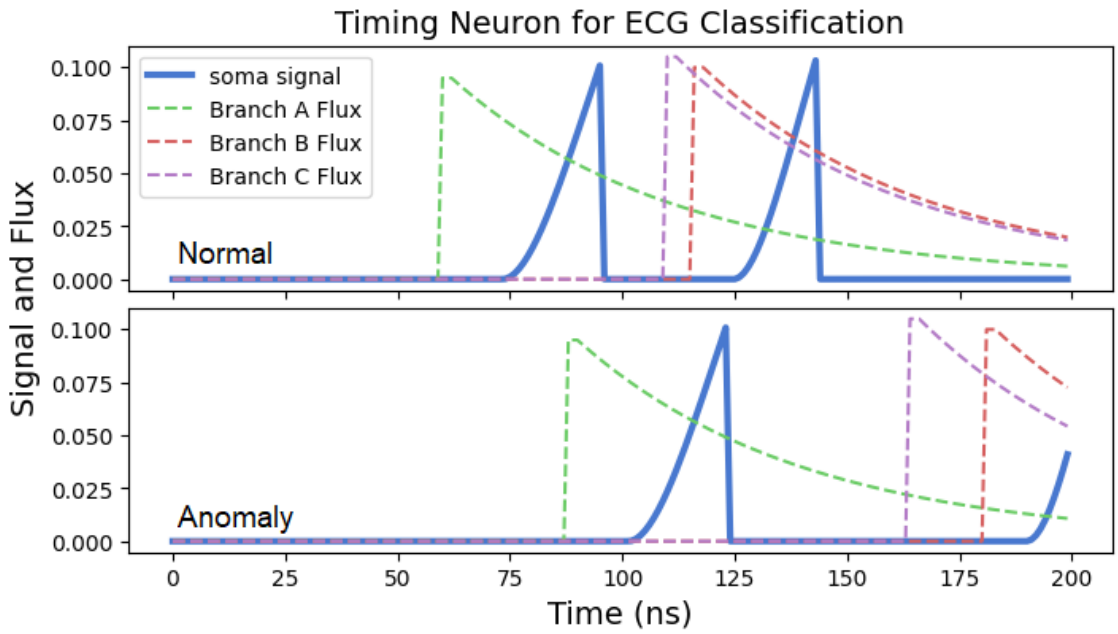


Figure 3.16: Timing neuron of a SOENs sequence detector system encountering pattern-layer-filtered input for both a normal and anomalous heartbeat signal. Note that varying leak rates of timing branches are not visible in the flux domain and the amplitude of flux is artificially weighted by its branch connection strength to the soma for better visualization.

3.6 Discussion

Sequence detection has a number of real-life applications that may be more appropriate for SOENs than ECG data, such as the classification of cellular signal signatures [152]. Moreover sequence detection of spatial patterns is a precursor to action recognition in video data – a task that SOENs may be uniquely well-matched to process en masse due to speed and scaling capabilities. Work presented here may also be extended to detect signal anomalies in an unsupervised fashion by using the arbor update rule to automatically balance timing neuron output with sequence input at the edge of spiking. After some learning period, any perturbation in the signal should cause anomalous spiking. In general, this work ought to be extended to more complex and larger datasets beyond nine-pixel patterns and ECG signal classification. Simple nine-pixel patterns may, however, be suited for early implementations on physical hardware. In the long term, due to the frequency at which SOENs operate, real-time pattern-sequence detection in these systems may be extremely powerful for high-speed and high-bandwidth tasks in the spatiotemporal domain, therefore motivating a demand to extend the preliminary investigations presented here.

Acknowledgment

This work was made possible by the institutional support from the National Institute of Standards and Technology (Award No. 70NANB18H006).

3. THE ARBOR UPDATE RULE FOR SPATIOTEMPORAL PROCESSING

4

Recurrent Multiplexed Gradient Descent for Time-Series Classification

Citation and author list: [153]

Paper Abstract

The brain implements recurrent neural networks (RNNs) efficiently. Modern computing hardware does not. Although specialized neuromorphic systems are well suited for recurrent implementations in the inference phase, training RNNs on-chip is a challenge and not amenable to backpropagation through time (BPTT). To mitigate this mismatch of RNNs and hardware, we propose the application of Multiplexed Gradient Descent (MGD) for model-free learning in recurrent networks with simple operations realizable by a number of neuromorphic systems. MGD does not require analytic differentiation, weight transport, diverse computations, or even a hardware model. Nevertheless, MGD can estimate the true gradient of a loss landscape for any differentiable objective function and is able to do so with multiplexed perturbations. Considerable time advantages are accessible by minimizing the resolution of gradient estimation required for gradient descent. The only operations needed for MGD are those canonically associated with an RNN along with additional mechanisms for global broadcast, local memory, and pseudorandom value generation. In this paper, we demonstrate that MGD converges to the true gradient in agreement with BPTT for a recurrent state space model (SSM), independent of learning task and network initialization. MGD is found to perform gradient descent with efficient gradient approximations on real-world time series data, obtaining a 99% test accuracy on

the ECG anomaly dataset and 93% on the full imbalanced multiclass ECG-5000, with similar results over a wide variety of network sizes, initialization, and hyperparameter settings. We show that learning in the recurrent layer is self-regulating for stability by analyzing its ongoing spectral radius. The separability of learned representations by this method are visually and quantitatively analyzed with k-means clustering, principal component analysis, and distance measures. Finally, prospects for neuromorphic hardware implementations are proposed given the modest operation and topological requirements to learn in RNNs with MGD *in-situ*, thereby offering a viable path toward the full neuromorphic advantage of learning in RNNs on-chip.

4.1 Introduction

Table 4.1: A case for on-chip RNNs trained with MGD.

System	Efficient RNNs	BPTT	MGD
Brain	✓	✗	?
CPU/GPU	✗	✓	✓
Simulated Neuromorphics	✗	✓	✓
Physical Neuromorphics	✓	✗	✓

Recurrent neural networks (RNNs) are the natural choice for temporally dependent computing. Certainly, RNNs are nature's choice, as the vast majority of all network topologies in the brain are recurrent [154–156]. From a theoretical standpoint, cycles in a network create temporal offsets that are capable of producing higher-dimensional reconstructions according to Taken's theorem [157]. Intuitively, allowing information from previous inputs to persist in a network through recurrence, rather than being immediately extruded through an output layer, readily paints the picture of a memory window for input-history-informed network activity. Indeed, RNNs are the established choice for brain-inspired computing in the time domain [158–162] , and have moreover seen a resurgence with state-of-the-art results in sequence processing by way of Legendre Memory Units [163, 164] and later State Space Models (SSMs, Fig. 4.1) [67, 165, 166] with strong performance even in the large language model (LLM) domain. While modern SSM results leverage GPU-trainable *approximations* of recurrence, efficient and scalable in-hardware training of true RNNs remains to be seen.

Generally, the motivation to realize temporal computing with brain-inspired efficiency using RNNs is at odds with modern computing hardware. Although iterative matrix

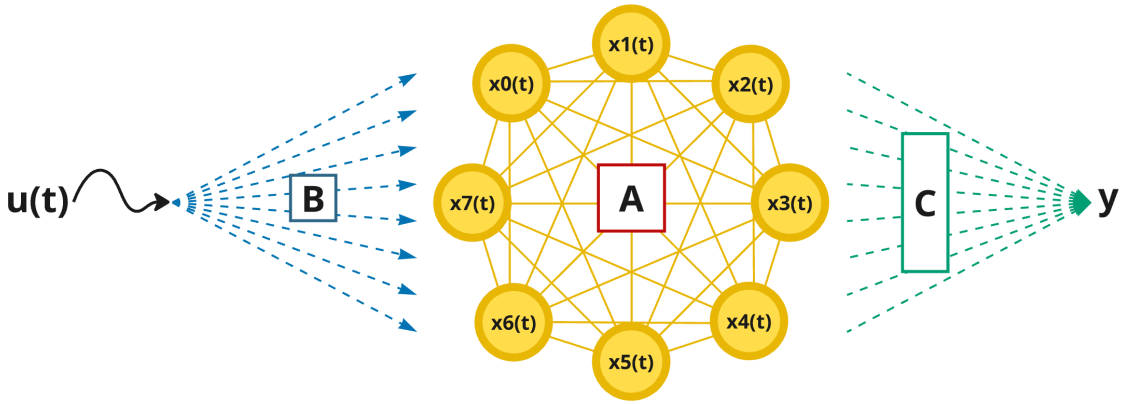


Figure 4.1: Input signal $u(t)$ is projected to state $x(t) \in \mathbb{R}^N$ according to projection matrix $B \in \mathbb{R}^N$ where N is the size of the state. $x(t)$ is moreover subject to the recurrent transition matrix $A \in \mathbb{R}^{N \times N}$ at ever step in time. Here, only the final state $x(t = T)$ is mapped by readout C to some output y, which is associated with some objective function to guide learning.

multiplication can instantiate a recurrent system in a way well suited to both GPU parallelism [167] and the backpropagation-through-time (BPTT) learning algorithm, it fails to take advantage of in-memory computation with dedicated neural circuit elements as is observed in the brain. Even basic field programmable array (FPGA) implementations of RNNs have been shown to outperform CPUs and GPUs [168], despite that FPGAs are not uniquely designed for RNNs. Though GPUs are the most powerful computing hardware for multilayer perceptrons (MLPs) and convolutional architectures, they still struggle to implement RNNs directly with commensurate efficiency, and this is all the more reason to consider brain-inspired methods on brain-inspired hardware.

Neuromorphic hardware can implement neuronal activations and recurrence through native properties of physical circuit elements, resulting in massive efficiency gains that enable small-footprint, real-time devices that enjoy inherent low power through asynchronous parallelism, in-memory compute, and physical activation functions [169]. However, these extremely efficient implementations of neural networks do not follow the von Neumann architectural scheme and are therefore less poised to implement the diverse and sophisticated computational requirements of backpropagation, and moreover BPTT, *on-chip*. This is because BPTT requires a variety of computation types (including differentiation on well-defined activation functions), weight transport [87] (the topological constraint of backward passes requiring every new downstream weight to be routed to every learnable parameter),

and unrolling the recurrent graph in time – none of which are suitable for specialized hardware that involve noisy, unconventional, and even non-differentiable activation functions, dedicated topologies, and in-time in-memory compute.

An alternative solution is to train entirely off-chip with simulated hardware models before transferring weights to the neuromorphic device, or even cycling through on-chip forward passes, off-chip backward calculations, and on-chip updates. These approaches suffer from susceptibility to inaccurate hardware models, imperfect weight transfer techniques, device defects, and noise. The motivation for on-chip, *in-situ* training methods is therefore great because learning on the hardware itself, and around its associated noise and defects, circumvents much of the aforementioned susceptibilities by including these properties in the learning procedure. The limiting factor thus becomes what learning algorithms can be instantiated with the limited (but efficient) operation scope of neuromorphic hardware? A variety of choices have been explored [170–172], but a definitive solution to *in-situ* training in neuromorphic systems, and especially those implementing RNNs, is not yet salient. This claim is evidenced by the sparsity of experimental papers that demonstrate recurrent on-chip neuromorphic learning, though promising efforts have been made [173–176].

We here propose an existing learning algorithm, multiplexed gradient descent (MGD) [103, 177], as a viable solution to the mismatch between the potential neuromorphic advantage of implementing RNNs and the neuromorphic disadvantage of on-chip learning in RNNs, thus advancing toward a caveat-free neuromorphic advantage in modern computing (see Table 4.1). MGD requires only simple operations and topologies in the form of random number generation, global broadcast, and local memory, all of which are commonly found in neuromorphic systems (as we will examine in Sec. 5.9), and are present in the brain in the form of stochasticity, neuro-modulation, and synaptic storage respectively.

Perturbative methods are well known to reliably estimate the true gradient with respect to an objective function for sufficiently small perturbation sizes and have strong theoretical grounds for this guarantee [178]. The mechanics of perturbative learning are most obviously understood by the finite difference method [100], whereby a single parameter of a system is randomly perturbed by some value ϵ, whose contribution to the change in global cost is perfectly isolated. By iterating over all parameters, the method converges to the true gradient in the limit of $\epsilon \to 0$ and is guaranteed to work for *any* differentiable function with a tractable gradient. Importantly, perturbative methods of this sort are model-free (i.e., the function-to-differentiate need not be known). MGD introduces an approach to multiplex (parallelize) these perturbations in a time-efficient manner by employing random

$\pm\epsilon$ perturbations at every learnable parameter simultaneously, and by moderating three key hyperparameters:

1. τ_θ: Number of iterations to integrate into a gradient estimation before making an update.

2. τ_p: Number of iterations per set of perturbation values.

3. τ_x: Number of iterations per training example.

To execute gradient descent in a neural network using MGD, it is often sufficient to batch a set of training examples (τ_θ = batch size), make a random $\pm\epsilon$ perturbation to all parameters (τ_p = batch size) only once before making an update ($\tau_x = 1$) [103], and this happens to be on the upper end of operation efficiency for perturbative gradient descent, roughly $\mathcal{O}(N_\Theta \cdot iter)$, where N_Θ is the total number of parameters and $iter$ is the total number of training iterations. The method thus benefits from efficient multiplexing of perturbations and updates.

The key insight as to why MGD works is that over time the history of random perturbation signs for a given parameter becomes increasingly unique to that parameter, and likewise updates to that parameter become increasingly uniquely correlated to its true contribution to the cost. According to this insight, and by [178], this method is guaranteed to approximate the true gradient for any differentiable system, regardless of system topology, activation functions, hardware defects, and/or noise (given an appropriate noise-to-ϵ ratio). This then should apply (as will be demonstrated) to recurrent topologies with operations confined to those afforded by specialized neuromorphic hardware implementations.

The MGD learning algorithm starts by accumulating an estimate of the gradient,

$$\nabla_i \leftarrow \Delta C \theta_i \tag{4.1}$$

where ∇_i is the i^{th} component of the gradient, ΔC is the change in cost before and after multiplexed perturbations are applied, and θ_i is the sign and magnitude of the perturbation to the learnable parameter associated with the i^{th} component of the gradient. The estimation is accumulated over a number of iterations designated by the hyperparameter τ_θ, and stored in a distributed component-wise fashion before the simple update

$$\Theta \leftarrow -\eta \nabla / \tau_\theta \tag{4.2}$$

is applied to the set of all learnable parameters Θ according to learning rate η.

4. RECURRENT MULTIPLEXED GRADIENT DESCENT FOR TIME-SERIES CLASSIFICATION

The work presented here is the first application of MGD toward learning in RNNs, and we therefore make contact with where RNNs are currently seeing the greatest advantages in modern computing. Legendre Memory Units (LMUs), and later State Space Models (SSMs), utilize principled structuring of an input-to-hidden projection matrix, and linearly recurrent hidden layer, to recoup coefficients to the Legendre polynomials in the form of neuronal activations for a specified sliding window of time over a signal or sequence. These dynamics accommodate improved history-informed representations at any given moment in the recurrent state, and therefore promote more effective temporal learning and smoother gradient initializations. Adopting the common notation of SSM literature, we describe (and depict, in Fig. 4.1) the associated system of equations accordingly,

$$x'(t) = Ax(t) + Bu(t) \tag{4.3}$$

$$y(t) = Cx(t) + Dx(t), \tag{4.4}$$

where B projects temporal input $u(t)$ to state space $x(t)$, which is itself subject to the recurrent transition matrix A. The readout map C projects $x(t)$ to some output $y(t)$. Note, that A and B are discretized as a function of input length and simulation time step (according to [67]). As is typical in SSMs, $D := 0$ and C can represent any sort of readout mapping (linear, nonlinear, multilayer, etc.), which often uses only the final state for classification, and in which case we refer to $y(t)$ instead as y. Dropping D, Equation 4.4 becomes $y = Cx(t = T)$, where T is the length of input. Learning can take place on any of these active parameters (A, B, and C) with the objective function being associated only with the final output y. We indeed show that MGD can train the entire system, (projection B, recurrence A, and readout C). Moreover, MGD is found to increase machine learning performance and quantitatively improve separability in state representations of input.

When considering the relationship of contemporary SSM literature to an operation-limited learning algorithm for training an SSM in its *recurrent* form, it is important to note that although SSM architectures like S4 [165] employ efficient approximations of recurrent dynamics according to a convolutional kernel on unrolled time, they do not instantiate learning in physical in-memory recurrence. Specialized, naturally recurrent hardware (like the brain), stands to gain from efficient implementations of RNNs rather than kernel approximations, which are non-physical and require sophisticated training techniques that would not readily transfer to specialized hardware. This work offers a proof of concept for on-chip training of physical RNNs that should in theory have access to some of the many impressive results of modern SSMs on GPUs. We show that MGD

converges to the true recurrent gradient, independent of parameter count and network topology, therefore making a strong suggestion that these findings should scale to the greater scopes implementable in specialized neuromorphic hardware.

We thus propose a method for performing gradient descent in RNNs using operations that are conducive to on-chip implementation in neuromorphic systems. In Section 4.2, we demonstrate empirically that MGD converges to the true gradient for the entire state space architecture, including the recurrent layer. In Section 5.7, we apply this finding toward machine learning performance on real data and investigate the learning dynamics of these results according to the ongoing spectral radius of the recurrent matrix. The class separability of state representations is scored with k-means clustering and visualized with principle component analysis (PCA). Finally, appropriate candidates for hardware implementations and suggestions for future work are proposed in Section 5.9.

4.2 Gradient Convergence

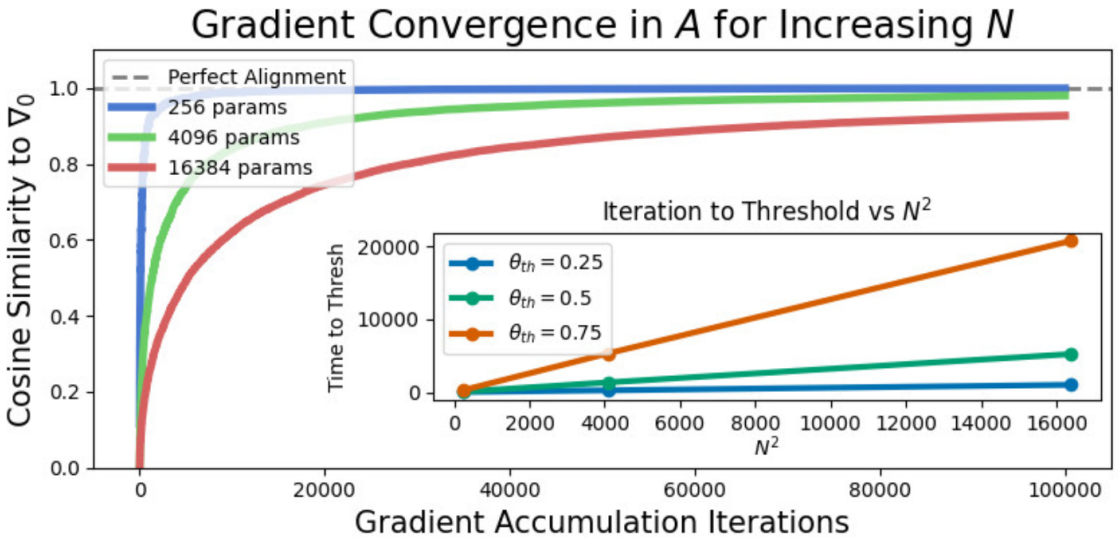


Figure 4.2: Alignment of estimated gradient with true gradient over time for increasing state size N. A grows with N^2. The inset shows time-to-threshold over increasing parameter count (N^2), for different selected thresholds.

The first step in demonstrating the efficacy of MGD gradient estimation for recurrent systems is to examine the alignment of estimated gradients with the true gradient ∇_0, which is here computed with BPTT on random samples from the ECG-5000 dataset. In

every case ∇_0 is corroborated in the perturbative context with the finite difference (FD) method in the limit of $\epsilon \to 0$. It is found that BPTT and FD are always in alignment. Although the FD is an excellent gradient estimator, it requires an entire forward pass *per parameter*, so at best it may here serve as a watermark for perturbation-derived gradient estimation, and not as a viable hardware-efficient learning algorithm itself.

Fig.4.2 shows the MGD gradient approximation method converging to the true backpropagation-derived gradient for all parameters associated with the recurrent matrix A, which scale with the state size N according to N^2. Note, the full A-B-C architecture is used in the forward pass and the gradient is estimated specifically for A through multiplexed perturbations on its respective parameters only. Again, referring the reader to [103], it is important to realize that gradient descent *does not require the true gradient*, but rather only an approximation that is in positive alignment (the same direction) as the true gradient. This insight is also the nature of stochastic gradient descent (SGD) [84], one of the most common variants on pure backpropagation in machine learning. We see here that MGD indeed is immediately in positive alignment, even after just one iteration. Therefore it is sufficient to set a low threshold for alignment convergence to benefit from improved scaling (see Fig. 4.2 inset). The MGD estimated gradient should always be in alignment with the true gradient, so long as a perturbation is not made exactly orthogonal to the gradient. This is because a perturbation that either increases or decreases the cost equivalently gives the same amount of information. Even for a single parameter, a perturbation that is orthogonal to the cost is extremely unlikely. A smooth, descending loss landscape, for example, would have only exactly two directions that could be orthogonal to the gradient. *En mass* it would be virtually impossible that perturbation information in the cost would be outweighed by orthogonal noise.

It is also found that the B and C parameter groups converge to the true gradient according to MGD, including in the presence of up-or-downstream recurrence. This finding prompts an investigation into a layerwise training scheme (collecting the gradient for each layer at every iteration separately), and it is found that all parameters converge together. Simultaneous gradient estimation across all layers does result in a reduced final alignment with respect to the total gradient, which is likely due to weight diffusion, but the alignment is still always positive and therefore sufficient for effective gradient descent, which we will see in Sec. 5.7. All of these findings extend to batching in an SGD fashion (i.e., $\tau_\theta = \tau_p =$ batch size, and $\tau_x = 1$). Like SGD with backpropagation, batch size does negatively correlate with gradient accuracy (linearly), and like with backpropagation, this is found to be

an asset for more regularized learning. In the case of RNNs trained with MGD, this may even result in better recurrent stability, as we will also explore in Sec. 5.7.

Having seen that MGD accurately converges on the true gradient, and is already in positive alignment with the true gradient in early iterations, we may now apply these findings to recurrent learning in RNNs on real data.

4.3 Results and Analysis

To give insight on training RNNs with MGD, we here consider a number of architectures, analytics, and visualizations. Precise details on datasets and implementation techniques are available in section 5.8 and moreover through the public git repository associated with this work.

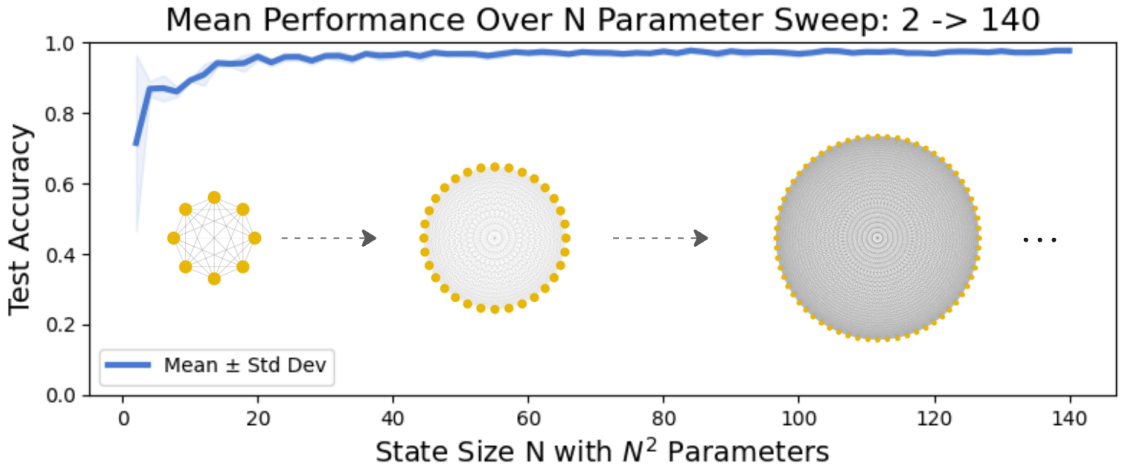


Figure 4.3: ECG-anomaly performance for increasing N over multiple initializations (five seeds each with randomness in the readout and perturbation draw). It can be seen that the variance is fairly low, and the performance is fairly high, even for small state sizes. This suggests a robustness to random variations in the model.

Please refer to Fig. 4.1 for a diagrammatic overview of the general architecture used in the results below, adopted from state space model literature. For the entire sequence length input is projected to the state $x \in \mathbb{R}^N$ according to $B \in \mathbb{R}^N$ and is subject to the recurrent transition matrix $A \in \mathbb{R}^{N,N}$, both of which are pre-derived and discretized according to the HiPPO initialization [67]. Once the input phase is complete, the final state $x(t = T)$ is mapped to some output y according to readout C and subsequently to an objective function (e.g. cross-entropy loss). C can stand for any number of readouts,

such as a multilayer perceptron, or a linear readout, all of which are suitable for training with MGD. We refer to the trainable parameters in a learning regime accordingly (e.g. an ABC-training model, or a C-only model).

4.3.1 Learning

As a first look into training RNNs with MGD, we consider the 5000-sample ECG-Anomaly detection task, which involves natural data in the time domain. We see this dataset is handily solved for a range of state sizes in Fig. 4.3, implicating trainability of both small and large recurrent networks with MGD. Fig. 4.4 visualizes the system as it learns to separate heartbeat classes with a test accuracy of 99%. In all cases, A, B, and C are trained together, and it is found that the model could utilize either a linear single-layer readout map or a nonlinear MLP with to a roughly equivalent effect.

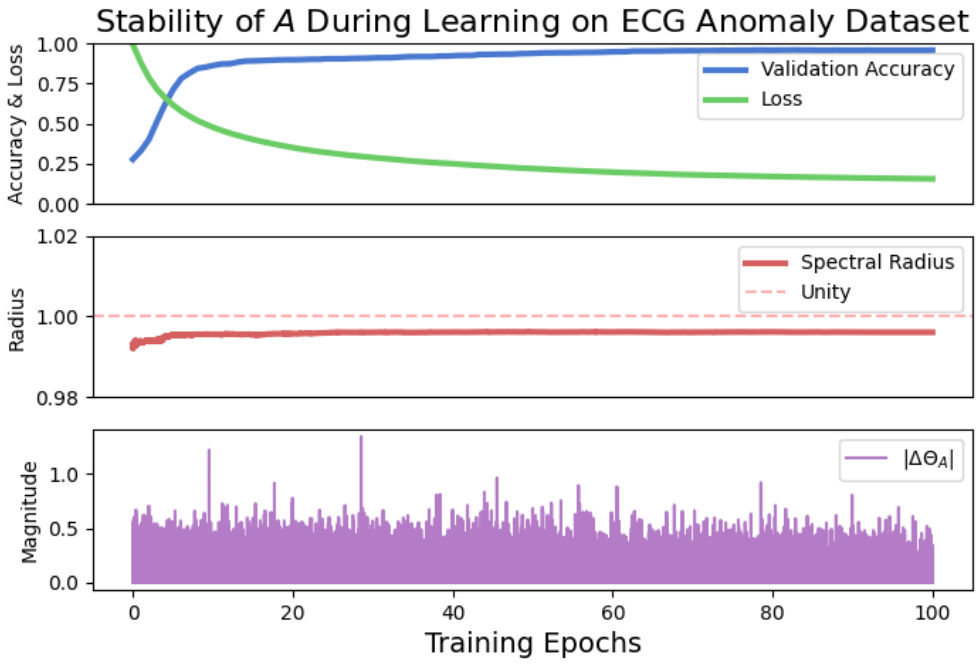


Figure 4.4: (Top) Validation accuracy and loss during training. (Middle) Ongoing spectral radius of A during training, which proves to be stable. (Bottom) The magnitude of updates being made to A, implying stability is not due to a placid update regime.

Importantly, the temperament of the recurrent layer proves to be fairly self-regulating when perturbed with MGD. Fig. 4.4 demonstrates the stable behavior A over the course of learning according to its spectral radius (its absolute maximum eigenvalue). When training

RNNs, this is not always guaranteed, as subtle changes in the recurrent layer can result in runaway dynamics that disable learning. Even BPTT often requires optimizers to maintain such balance. Generally, it is advisable to enforce the spectral radius of a fixed recurrent system to unity [159], as anything else should compound recurrent updates on themselves, resulting in runoff dynamics. Conversely, a small spectral radius can result in system dissipation that is too rapid to use for a learning scheme. However, it is observed with the MGD learning algorithm that the spectral radius tends to hover near unity, either above or below, without running off. This is an unexpected and convenient result which may follow from MGD's noisy estimations of the true gradient providing natural regularization. If the parameters make contact with a runoff region in the gradient, there would likely be several update iterations available to recognize this threat to cost minimization and move toward more stable regions. This reasoning is supported by how the gradient alignment becomes increasingly noisy as performance rises and the perturbation-to-loss ratio $\epsilon/cost$ increases. The resulting stability is observed over a wide range of parameter settings, loss functions, and readout maps (both linear and nonlinear), suggesting a rich parameter space for which MGD training responds smoothly to input.

In support of observed stability in A, we can refer to Fig. 4.5 and see that under a variety of conditions and initializations, training that involves the recurrent layer consistently performs well. A proves to be the most powerful single-trainable layer configuration, and A is involved in the top three highest performance means and the highest absolute performance. Training A, B, and C together results in the most stable performance with the least variance. Training C alone is the weakest classifier (though this still performs well), even given that it takes the pure HiPPO initialized SSM outputs as input. Notably, training B and C together results in strong performance, though B is associated to the objective function only through A, so the recurrent component is still integral. A is also stable for increasing state size N, and does improve performance with size, as seen in Table 4.2. Given these results, and robustness seen to state size and random seeds in Figs. 4.3 and 4.5, we can expect on an empirical basis that training A with MGD is prone to stability. Importantly, this same stability *is not* found using BPTT, which only was able to produce similar performance when using the Adam optimizer [90], a method unlikely to match hardware using dedicated circuits for computation.

4.3.2 Separability

From a theoretical standpoint, for a given input signal over a specified sliding window in time, a state space is recouping the Legendre polynomial coefficients associated with that

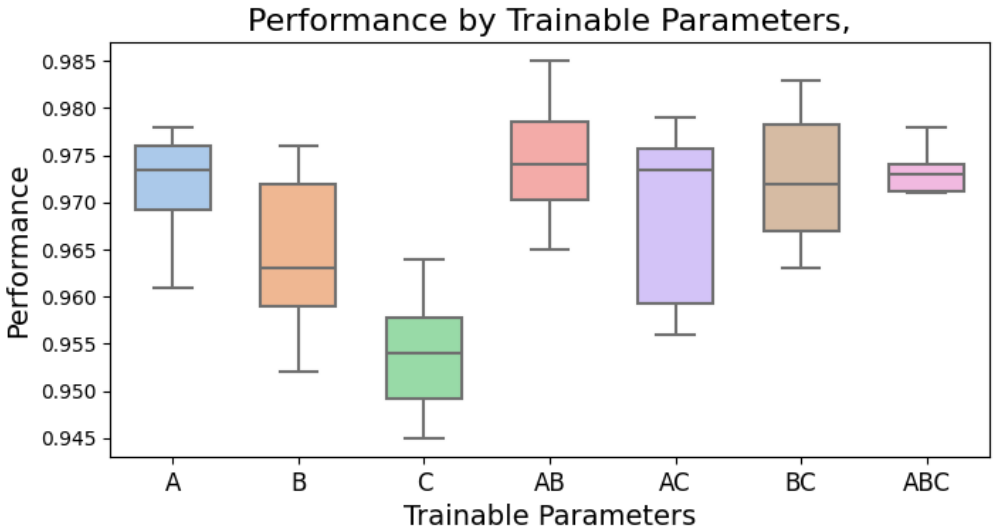


Figure 4.5: Hyperparameter sweep over all possible sets of trainable parameters with state size limited to $N = 32$ (to better tease out performance and for faster run time), with ten random seeds rerun for each configuration. Models all use a two-layer [N,N,1] MLP as their readout map C and learning rate is always normalized according to the parameter count and batch size (see 5.8). Generally, all configurations perform well and always tested over 93%. This is a typical box plot where the box center line corresponds to median, the upper and lower box bounds to the upper and lower quartiles, the whiskers to a 1.5x interquartile range, and circle-markers to outliers.

Table 4.2: A Few Highlighted performances.

dataset	N	C	loss	Accuracy
ECG-Anomaly	128	Linear	MSE	99.0%
ECG-Anomaly	128	MLP	CE	99.0 %
ECG-Anomaly	32	MLP	CE	98.5 %
ECG-5000	256	MLP	CE	93%

signal in its state activations to an order corresponding to the number of neurons in the state. This encodes history-related temporal information in the state vector at any given moment in time. While this is precisely the case at initialization, once training begins, state space dynamics will have no such guarantees. It becomes a question of interest then what sort of states are produced and what is their quality with respect to the learning task? The latter requires a metric for success in terms of state quality, which is readily available for estimation by a number of proxy metrics, all of which may be cast as statements of *separability*, by which we mean how easy is it to separate different states produced by the RNN with respect to their associated input class?

A first metric, as we have already seen, is of course raw classification performance. States more amenable to the classification task will be more classifiable because they are more separable. To isolate this metric from the readout map C, which is itself trained by MGD, we may also employ a simple logistic regression classifier over all states generated in response to a dataset to estimate the linear separability of the states produced before and after the MGD training procedure. We find that having trained A during a full ABC-training round essentially always improves logistic regression performance, and by as much as 5%, which we here take as a metric for improved linear separability.

Another method to assess state quality in terms of the separability of class representations is by way of K-means clustering, an unsupervised learning algorithm that clusters inputs optimally over a given number of centroids by ascertaining the centroid locations that produce the minimum distances to every data point. A classification accuracy can furthermore be produced by determining which centroid-to-label pairings result in the best classification. So as not to conflate this with our MGD model performances (which are better), we will refer to this as a separability score, because this is a good metric for determining how well separated class-associated data points are in any high dimensional space. Fig. 4.6 well illustrates the improved separability of state space representations for the *multiclass* ECG-5000 dataset (details in Sec. 5.8). We see a considerable boost in separability score (over 10% increase) as a function of MGD training, which itself produces a training accuracy of 93% with 256 states.

4.3.3 Dynamics

Beyond separability of the final state representation, a question also arises as to what sorts of dynamics are at play in higher-dimensions over time. PCA enables us to project these higher dimensions onto their components of maximum variance, and if this variance is taken across all states over all time and all inputs, we can appropriately compare the

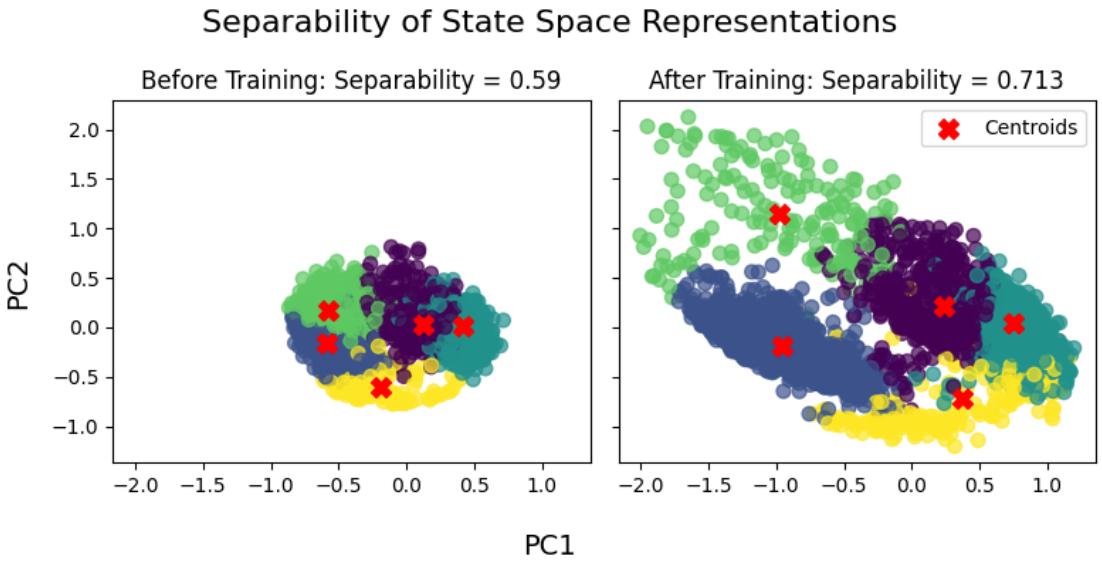


Figure 4.6: Final state vectors for every input in the multiclass ECG-5000 test set projected onto their first two principle components, and grouped according to the k-means clustering algorithm. The left plot is before the recurrent matrix A was subject to MGD training and the plot on the right is after. It can be qualitatively seen that training improves and quantitatively measured by the improved separability score (k-means accuracy).

Table 4.3: MGD-Related Operation Scope of Selected Neuromorphic Hardware

Hardware	On-Chip	Local Memory	Analog	Recurrent	RNG	Broadcast	MGD
Intel Loihi 2	✓	✓	✗	✓	✓	On/Off Chip	
BrainScaleS-2	✓	✓	✓	✓	✓	On/Off Chip	
SpiNNaker (1 & 2)	✗	✓	✗	✓	✓	Off Chip	

trajectories of high-dimensional input-driven state representations over classes. Referring to Fig. 4.7, we see that indeed states take on class-specific dynamics after being trained with MGD, and moreover that class separability across time is improved by training with MGD. Before training, class representations encircle each other throughout input duration and are often in close proximity (including in the final state). After training, it can be seen that class trajectories immediately separate from their common starting point and continue to separate throughout the input duration.

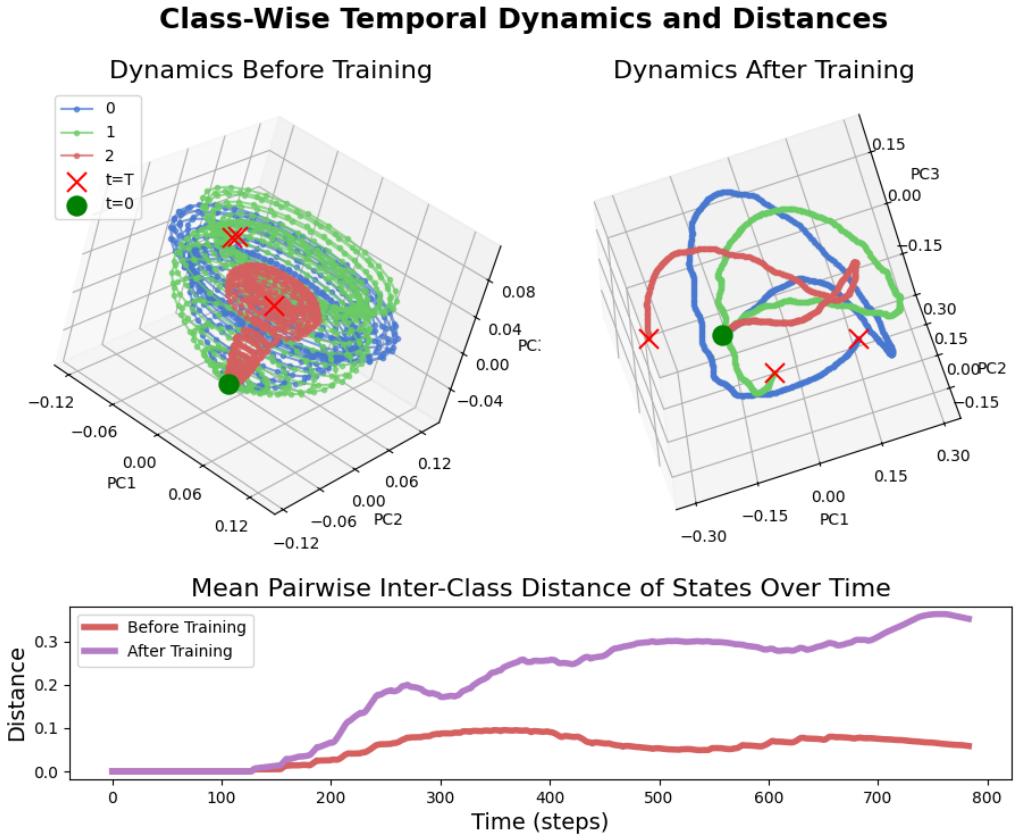


Figure 4.7: (Top) Mean state trajectories by class before and after training with MGD on sequential MNIST digits. Qualitatively, it is clear that paths are better separated post training. (Bottom) Quantifying temporal separability with mean pairwise inter-class distances of state representations over time. This is a proxy measure for separability and we see it is improved by training with MGD.

4.4 Methods

4.4.1 Simulations

Because training state space models in their recurrent (non-convolutional) form is not common practice, a codebase has been developed for this express purpose, and is publicly available at `https://github.com/ryangitsit/recurrent-mgd`, with directions to replicate primary results of this paper.

Simulations of recurrent neural networks can be executed in a number of ways, with the forward Euler method being the most straightforward choice and requiring the fewest steps of approximation. This, and the requirement to broadcast functions over non-trivial structures, make the JAX programming language a natural choice. A single simulation step directly follows equations 4.3 and 4.4, which can be batched in parallel over many examples simply by adding another dimension to input u and output y. Time is iterated over with the *jax.lax.scan* function and parameter updates are broadcast with *jax.tree.map* to all parameters at once.

4.4.2 Training and Testing

The objective function is applied to the final output of $y(t = T)$ as mapped according to readout C from the final internal RNN state $x(t = T)$. Therefore, the temporal components of the system are the input signal, the state to which the input signal is projected, and the state transitions. This is a typical scheme in SSMs and a reasonable approach for hardware implementations.

4.4.3 PCA and K-Means Clustering

K-Means clustering is used in a typical unsupervised fashion and accuracy on the test set is derived by optimally assigning classes to each cluster post-hoc. This is all done as a proxy for estimating how separable states are in high-dimensional space in order to quantitatively complement the visual separability of plotting PCA trajectories, which are themselves averaged over all state-space-paths for each sample with respect to class for the entire sequential MNIST dataset.

4.4.4 Datasets

The ECG-Anomaly dataset has 5000 samples, each of length 140 ms (and 140 data points), with a roughly even split of normal and anomalous heartbeats. The full multiclass ECG-

5000 includes more resolved classification over the anomalies for a total of five classes, which are steeply unevenly distributed. The training set has only 500 samples, and the test set, 4500. Few shot learning is required on the least-common classes.

4.4.5 Learning Rate

Explicit learning rates, and potentially decay, are the most straightforward to implement in specialized hardware, and we therefore constrain ourselves to only these techniques for deriving η. While it is expected that performance should improve with optimizers borrowed from machine learning, it is unreasonable to expect these multifaceted computations to sit well into dedicated circuits. We instead use the normalization equation from [103], except for that we count the number of parameters K according to how many times each weight is touched for one sample through time. Thus, we set

$$\Gamma = c \cdot \delta^2 \sqrt{K} \sqrt{1 + (K-1)/\tau_\theta} \tag{4.5}$$

$$\eta := 1/\Gamma \tag{4.6}$$

where δ is the magnitude of perturbation size being used globally, τ_θ is essentially batch size, and c is a variable coefficient, but is almost always set to the number of classes in the dataset.

4.5 Discussion and Hardware

Having seen that SSM-RNNs can be trained with MGD using only a simple set of operations, minimal iterations, varying state sizes, and for a large range of hyperparameters, the question then becomes *what tasks can this method scale to?* Given that MGD has been shown to estimate the true end-to-end gradient for an SSM, it is reasonable to aim for many of the impressive results and scales produced in SSM literature, culminating in competitive LLM performance. However, such results involve kernalized approximations of RNNs (reducing $N^2 \to N$) and are heavily optimized for GPUs, which is another case of adapting AI to modern hardware. However, there are an increasing number of efforts to develop hardware for the explicit purpose of intelligence, and especially brain-inspired intelligence, in the form of neuromorphic computing.

MGD generally offers a viable path for training unconventional systems, and the work presented here extends this viability to RNNs, where neuromorphic computing stands to shine most over conventional hardware by virtue of natural recurrence, in-memory compute,

and physically instantiated native activation functions. Table 4.3 includes just some of the forerunners in the neuromorphic hardware field [15,19,83], all of which host the operations necessary to implement MGD toward the training of recurrent networks. This list is not necessarily comprehensive, but rather an example of how to map hardware features to MGD requirements. Candidate in-development hardware also exists, such as superconducting optoelectronic networks [32, 120]. All of the above offer spiking communication (but not exclusively) and it is therefore of interest that spiking-SSMs have been shown to outperform even transformers on long-range dependency sequence tasks [179], and given that for the selected spiking dynamics surrogate functions were sufficient to train the system with BPTT, it should follow that this spiking architecture has sufficiently smooth gradients to be trained with MGD on-chip.

A natural extension for future work is to make these suggested hardware implementations a reality. Even prior to hardware, further extensions in the simulation domain can be accessed. Simulated spiking RNN implementations will advance the coupling of MGD to existing hardware efforts. Deep SSMs in the large language model domain would also strengthen the case for scalability. However, as neither RNNs nor MGD are best suited for simulations, it is of paramount interest to implement these methods on neuromorphic systems first and foremost. This may grant access to speeds and scales previously unattainable, and perhaps not foreseeable. The combination of *in-situ* training, RNNs, and specialized hardware, has not yet been realized and cannot be well mimicked on conventional computers. Thus, we outline an opportunity to train RNNs with MGD on-chip, and to potentially fulfill the promise of neuromorphic advantage with novel forms of machine intelligence.

Acknowledgment

This work was made possible by the institutional support from the National Institute of Standards and Technology and the University of Colorado Boulder.

Disclaimer

Commercial hardware are mentioned to contextualize algorithm applicability, but this does not imply endorsement of any product or service by NIST.

5

Delta MGD

Citation and author list: [8]

Paper Abstract

We present $\vec{\delta}$-Multiplexed Gradient Descent ($\vec{\delta}$-MGD), a fully neuromorphic algorithm that consistently performs alongside backpropagation with orders of magnitude fewer gradient component calculations across a number of datasets and network architectures. This performance is enabled by the finding that perturbative methods perform near-optimally when parameters are perturbed in the same direction as the true ternary gradient, and that even a noisy estimate of the gradient yields marked improvements in learning performance and convergence time. We show that local synaptic information can be used to make strong estimates of the ternary gradient using only presynaptic activation and postsynaptic bias gradients. Moreover, we find the latter can be cast into learnable class-wise representations, which we call *DeltaVecs*. According to these findings we realize a reduction of $N_{samples} \cdot N_{neurons}^2 \rightarrow N_{classes} \cdot N_{neurons}$ in terms of gradient components that must be explicitly computed per epoch. We further find that astrocyte-inspired subnetworks are able to learn DeltaVec representations in order to drive gradient-informed perturbations. Finally, we stack these methods together and show that using only neuro-inspired hardware-friendly operations we can perform alongside backpropagation in networks on the scale of millions of parameters, in DNNs, CNNs, and even transformers. Altogether, we offer an algorithm implementing novel machine learning methods and neuro-inspired operations to realize compelling neuromorphic intelligence.

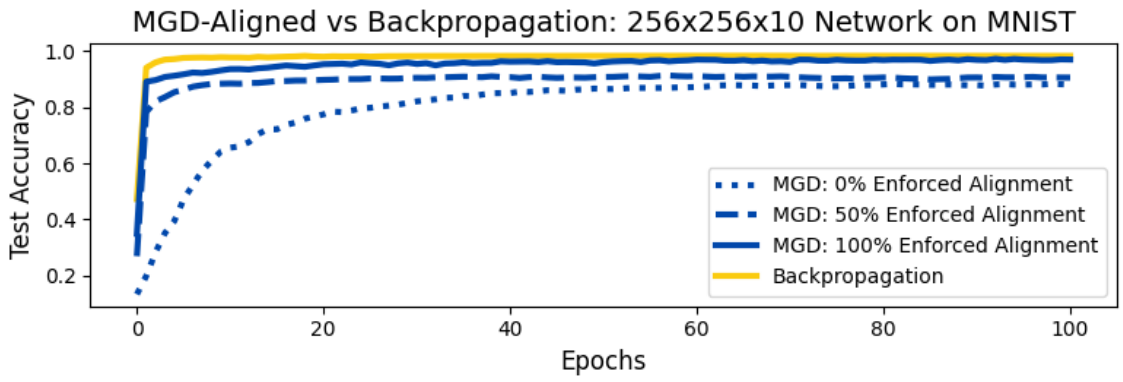


Figure 5.1: *MGD with gradient-aligned perturbations:* As the percentage of perturbations that are directed into signed-agreement with the ternary gradient increases, performance and convergence time are both improved.

5.1 Introduction

Neuromorphic computing offers a viable path toward specialized hardware for artificial intelligence by leveraging neuro-inspired physical-computing advantages for improved design of intelligent systems. A key challenge in neuromorphic computing is the development of hardware-suitable learning algorithms that are competitive with the backpropagation performance that drives nearly all learning in modern AI on traditional hardware. This work proposes a novel connection between astrocytes and perturbative methods, finding that astrocyte-like subnetworks can use locally-estimated compressed gradient representations to drive perturbations in order to realize massive performance gains.

We present $\vec{\delta}$-*MGD*, a hardware-friendly neuromorphic learning algorithm that consistently performs alongside backpropagation across a number of datasets and network architectures, including DNNs, CNNs, and transformers (preliminary), with orders of magnitude computational savings that grow with network size. $\vec{\delta}$-MGD is enabled by a sequence of novel observations and techniques that culminate in a fully neuromorphic learning scheme. These findings are enumerated below, with labels corresponding to their respective section numbers:

[I]**Gradient-aligned perturbations (MGD-aligned):** Perturbative methods can achieve backpropagation-like performance by aligning perturbations with a low-precision, noisy gradient estimation, as seen in Fig. 5.1. **Local gradient estimation (Hebbian MGD):** Using a Hebbian-like correlation of locally available information, a gradient estimation can be made to inform perturbation directions and implement

MGD-aligned. **Class-wise bias-gradient representations (DeltaVecs):** Bias gradients are found to have significant per-class structure that enables a massive reduction in the gradient search space, and offers a path to compact gradient representations with persistent viability over time. **Intelligent subnetworks (astrocytes):** Astrocyte-like subnetworks can serve to intelligently drive perturbations in more useful directions. In particular, they are able to efficiently learn, or sample, bias gradient representations and apply them according to local information in order to drive gradient-informed perturbations. **Fully neuromorphic implementation ($\tilde{\delta}$-*MGD*):** $\vec{\delta}$-MGD achieves backpropagation-like performance at a fraction of the operational cost by using astrocytes to estimate local class-wise gradient representations and drive aligned perturbations.

Sections I-V present standalone findings that together support the final algorithm $\vec{\delta}$-MGD of Sec. 5.6 and lead to the final results in 5.7. These findings mark a successful case of neuro-inspired improvements to hardware-compatible machine learning performance and are believed to be state-of-the-art perturbative performance in terms of accuracy over efficiency quotient [103, 177, 178, 180, 181].

5.2 Gradient-informed perturbations

We find that aligning perturbation vectors with a low-precision gradient estimate yields massive performance gains by maximizing perturbative signals measurable in the global cost. This finding is applied in the context of the multiplexed gradient descent (MGD) framework, which when directed with gradient-informed perturbations is referred to as *MGD-aligned*.

5.2.1 Perturbative Methods

Generally, perturbative methods correlate local parameter changes with a global change in cost in order to estimate the network gradient. Multiplexed gradient descent (MGD) [177] is a model-free gradient-descent framework through which perturbative methods, such as SPSA [178, 180] and finite-difference [100], can be implemented with hardware conscientiousness by introducing a trio of time constants τ_x, τ_θ, and τ_p governing the perturbation process. For example, SPSA is realized within the MGD framework when τ_θ=1 (simultaneous perturbation of all parameters) and $\tau_p = 1$ (updating parameters after every round of perturbations). Perturbations are drawn from a Bernoulli distribution where

each value has an equal and independent chance of being assigned $\pm\epsilon$. In SPSA, gradients are estimated for the i^{th} parameter according to

$$\hat{\nabla}_i \leftarrow \frac{\Delta C \cdot \tilde{\theta}_i}{\epsilon^2} \approx \frac{\Delta C}{\Delta \theta_i} \approx \frac{\partial C}{\partial \theta_i} \tag{5.1}$$

where $\hat{\nabla}_i$ is the i^{th} component of the estimated gradient and ΔC is the measured change in global cost with respect to an unperturbed forward pass. Referring to Fig. 5.2 (top), we see that the true sign of parameter-wise ΔC_i is not always correctly represented by the aggregate measure ΔC. This is because SPSA cannot directly measure each contribution ΔC_i, but rather only sees their aggregate through total change in network cost ΔC. The finite difference method [100] perturbs parameters one-at-a-time to measure perfectly their individual contributions to cost ΔC_i, but pays for this estimation accuracy with required forward passes per gradient-estimate scaling with the total number of network parameters N_Θ. MGD benefits from a more practical approach whereby repeated applications of Eqn. 5.1 with freshly drawn perturbation vectors will strengthen the statistical correlation of each estimated component with its true gradient by way of temporal integration. Increasing integration time τ_θ up to $\tau_\theta = N_\Theta$ will recoup the same estimation accuracy as finite difference [178], but again at the cost of N_Θ scaling. A key finding of MGD, however, is that perfect gradient estimations are not required to perform gradient descent, and that hardware efficiency and machine learning performance can be balanced by moderating τ_θ [103].

Table 5.1: Gradient calculation methods, estimation strength, scaling, and hardware viability.

∇ Method	$\frac{\partial C}{\partial \theta_i} \propto \frac{\Delta C}{\Delta \theta_i}$	Scaling	HW-friendly
Backprop	Perfect	$\mathcal{O}(bp)$	✗
Finite Difference	Very Strong	$\mathcal{O}(N_\Theta \cdot fp)$	✓
MGD($\tau = 1$) (SPSA)	Weak	$\mathcal{O}(fp)$	✓
MGD($\tau = T$)	Dependent	$\mathcal{O}(T \cdot fp)$	✓
MGD($\tau = N_\Theta$)	Very Strong	$\mathcal{O}(N_\Theta \cdot fp)$	✓
MGD-Aligned	Very Strong	$\mathcal{O}(fp)$	✓

bp = one backward pass, fp = one forward pass

The performance/efficiency tension is thus as follows (as listed in Table 5.1): If $\tau_\theta = N_\Theta$, gradient estimations are very strong, but scaling the network will punish convergence time. On the other hand, if $\tau\theta = 1$, gradient estimates are poor, but there is no punishment for

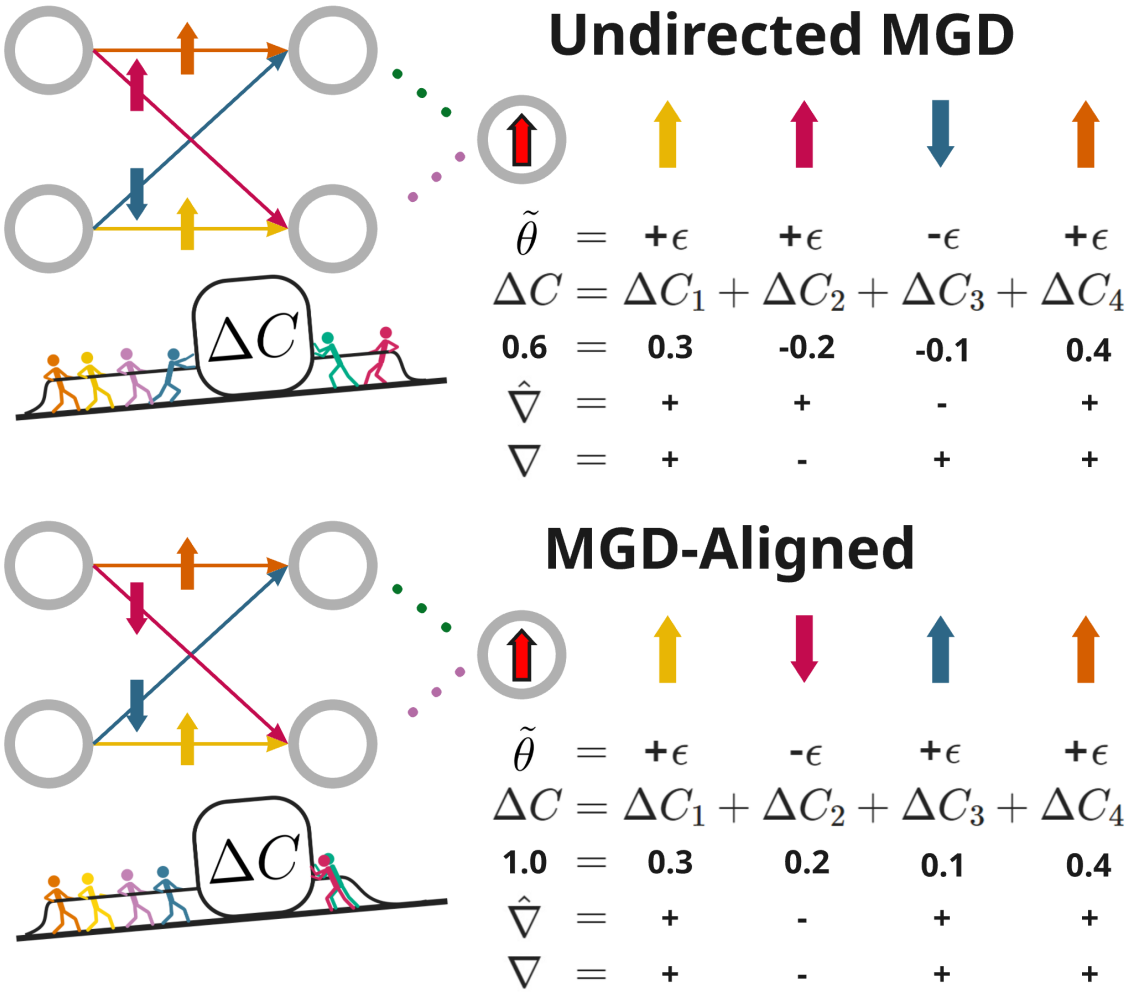


Figure 5.2: (Top) **Undirected MGD:** Weights are subject to random perturbations $\tilde{\theta}$ sampled from $\pm\epsilon$. Each perturbation $\tilde{\theta}_i$ affects the cost of the network, changing it by ΔC_i. $\hat{\nabla}$ is estimated according to $\Delta C \cdot \tilde{\theta}_i/\epsilon^2 \approx \Delta C/\Delta\tilde{\theta}_i$ for each component, and we can see here that the sign of this estimate is not always in the same direction as what would have been produced if ΔC_i were known creating a mismatch between estimated gradient $\hat{\nabla}$ and true gradient ∇. We can also observe, that the estimated gradient components that have the correct sign correspond to the majority of true total gradient magnitude, which is proportional to the unknown values ΔC_i. This is why undirected MGD still performs gradient descent, even if inefficiently. Conflicting contributions to cost are analogized to the disorderly stick-figure efforts depicted alongside the worked example. (Bottom) **MGD-Aligned:** Here we observe a straightforward remedy to the self-cancellation of undirected-MGD. When perturbations are directed such that a common sign will be produced by all ΔC_i components, *all* of their respective impacts on the network are accounted for in the ΔC measure, whose magnitude is thus increased. Each parameter is still updated with correct signage according to Equation 5.1 and we see all estimated components align with the true gradient.

scaling the number of parameters in a network. In general, we would like to maximize gradient estimate accuracy and minimize integration time. A long standing open question of perturbative methods is whether the performance/efficiency ratio could be improved by more selective perturbation vectors. To answer this, we first investigate where misalignment is accumulating according to random perturbations.

5.2.2 Washout

If a parameter is perturbed alone, the change in cost might be measured as being in a certain direction, but when that same perturbation is obfuscated by the result of all parameters being perturbed simultaneously, the change in cost may be measured as being in the *opposite* direction. Eqn. 5.1 approximates a measurement of the true gradient component $\frac{\Delta C_i}{\Delta \theta_i}$ with the efficiently measurable value $\frac{\Delta C}{\Delta \theta_i}$. This approximation simplifies the task of measuring perturbation-to-cost correlations one-by-one to a single measurement of how all perturbations associate collectively with a change in global cost. Each i^{th} parameter is affected by its own perturbation $\tilde{\theta}_i$, which can be then be related directly to the total ΔC. Again referring to Fig. 5.2 (top), we see that the true parameter-wise contribution to cost is not always represented in the measured change in global cost. For example, the dark red weight is subject to a positive perturbation, and a positive change in global cost is measured. This drives our Eqn. 5.1 estimate to assume that making this parameter bigger is increasing the cost of the network, suggesting a *positive gradient* and *negative update*. However, we see that in reality this parameter is associated with a negative contribution to the cost, meaning that increasing this parameter value would further minimize loss. This should ideally result in a *negative gradient* estimate and *positive update*. We therefore find that undirected MGD, which employs random Bernoulli perturbations, necessarily suffers from some percentage of parameters receiving erroneous updates. This is caused by their contributions to the global cost being canceled (*washed out*) by conflicting contributions from other parameters.

This characterization of the perturbation-cost relationship lends itself to a number of useful insights. First, in undirected MGD each parameter might receive a good, bad, or neutral update according to the sign agreement of ΔC and ΔC_i. The neutral case is that of weight diffusion [181], whereby updates are made to a parameter despite it having no correlation with the cost. Second, because local correlation is estimated according to Eqn. 5.1, and the measured change in cost is determined by $\Delta C = \sum_i^{N_\Theta} \Delta C_i$, the sign of ΔC will always represent the group of parameters that correlate with the majority impact on cost. Realize that each ΔC_i is determined by its corresponding $\tilde{\theta}_i$. When

the $sign(\Delta C_i) = sign(\Delta C)$ then the estimate is in the correct direction. Because per-parameter perturbations are what drive ΔC, the group of parameters with the strongest collective correlation to the cost will always determine the sign of ΔC and receive good updates. Equivalently, this implies that the angle between the estimated gradient and the true gradient should always be acute $\cos(\hat{\nabla}, \nabla) < 90°$ by virtue of the majority of gradient magnitude being represented by correctly-aligned estimates. This new insight is empirically supported in [103] and preempts the basis for deriving perturbation vectors that will maximize performance by minimizing washout and improving gradient estimations.

5.2.3 MGD Aligned

By inspection of Fig. 5.2, we can see that whatever the true gradient for a given component is, a set of perturbations exists such that a common sign is produced among all components ΔC_i, thus maximizing perturbative visibility in the cost, and guiding all component updates in the right direction. Enforcing this perturbative signal maximization is exactly equivalent to directing all perturbations into signed-agreement with the true ternary gradient, such that the correct relationships to their true gradient directions will be realized across a common direction for the measured change in cost. The same result is seen when perturbations are all directed together in the anti-direction of the ternary gradient, thus flipping ΔC. The key detail is that all perturbations should have the *same* correlation with the true gradient. To confirm this intuition, we turn to empirical measurements. We define an *agreement measure* Λ that computes the ratio of components in a vector A that are in ternary-signed agreement with those in another vector B

$$\Lambda = \frac{\sum_{i=0}^{N}[sign(A) = sign(B)]}{N}, \tag{5.2}$$

where N is the total number of components. Fig. 5.1 compares three different perturbation vector conditions against backpropagation, where 0%, 50%, and 100% of perturbation-vector components are enforced to agree with the true ternary gradient. When 100% of components are subject to this condition, $\Lambda = 1$. The effect of Λ on performance in terms of accuracy and convergence speed is unambiguous: the higher the Λ value, the better the accuracy and the faster the convergence. For $\Lambda = 1$ MGD performance adheres closely to that of backpropagation. We define *MGD-aligned* to be the case where $\Lambda(\tilde{\theta}, \nabla) \approx 1$, in contrast to *Undirected MGD* whose Bernoulli perturbations only result in $\cos(\hat{\nabla}, \nabla) < 90°$.

We find that MGD still performs gradient descent anywhere on the agreement spectrum from undirected to aligned. This shows that noisy estimations of the ternary gradient suffice

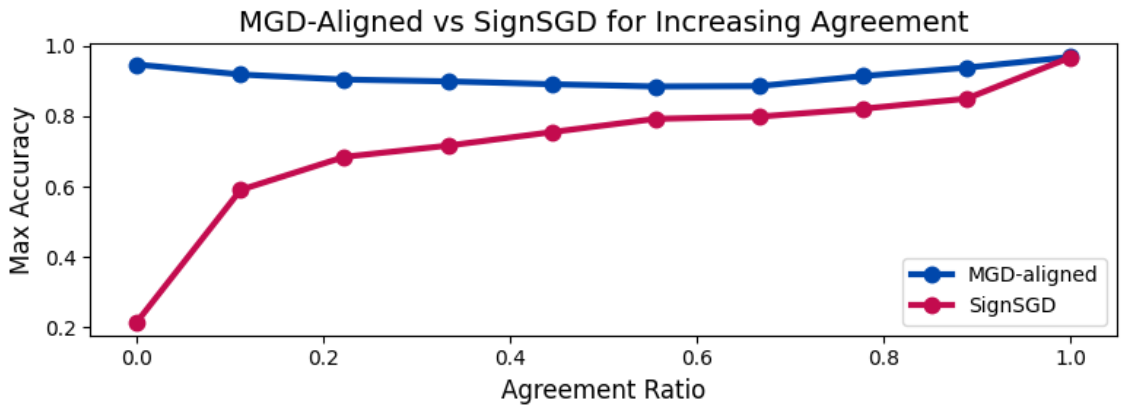


Figure 5.3: MGD vs SignSGD for different agreement ratios. SignSGD applies $\tilde{\theta}$ directly as a parameter update, whereas MGD utilizes $\tilde{\theta}$ as a perturbation vector. On the far left algorithms receive a random Bernoilli vector of $\pm\epsilon$ values for $\tilde{\theta}$. On the far right, $\tilde{\theta} = \epsilon \cdot sign(\nabla)$ such that $\Lambda = 1$. Each data point is the best performance of a sweep over ten learning rates, as the effective learning rate may be shifting for different $\tilde{\theta}$ distributions, which is likely the cause of the imperfect bowing in MGD performance.

for performance improvements. While it may be tempting to apply the ternary gradient directly toward learning, as is the case with SignSGD [182], we see in Fig. 5.3 that this approach is considerably less robust to noise. It should be stated that the motivation for SignSGD is to efficiently communicate gradient information across GPUs, rather than to realize neuromorphic learning on specialized hardware. The reason MGD is so much more noise tolerant is that updates are made proportional to ΔC, which is itself an empirical measure of how strong the ternary gradient estimate is correlated with the true gradient of the network. However, as already seen in Fig. 5.1, noisy perturbation vectors still cost in terms of performance and convergence speed. We are therefore motivated to investigate neuromorphic means for making strong ternary gradient estimates online and enable the scaling advantages outlined in Table 5.1.

5.3 Hebbian MGD: Deriving Ternary Gradient from Local Information

To access the performance gains of MGD-aligned, an estimate of the ternary gradient is required. Computing the gradient analytically, ternary or otherwise, is infeasible in dedicated hardware and we therefore propose a gradient-estimation method that is suitable for *in-situ* hardware implementation.

5.3.1 Estimation by Local Information

Perturbative methods can be implemented according to neuromorphic operations, but suffer with parameter scaling. This is unfavorable given that the number of learnable parameters (weights, biases, etc) N_Θ in a network scales roughly squared with the number of of neurons in a network $N_\Theta \propto N_n^2$. However, upon closer inspection of the problem space, as depicted in Fig. 5.4, we see that for any given weight parameter W_{ij} connecting neuron n_i with n_j, the partial derivative with respect to the cost function is given by the presynaptic activation a_i and the postsynaptic error term δ_j. Presynaptic activation is simply the value computed during a forward pass by a weight's upstream neuron and is available locally at the site of derivation. The postsynaptic delta is the bias gradient at the downstream neuron $\delta_j = \frac{\partial C}{\partial b_j}$, which can equivalently be thought of as the gradient at the postsynaptic neuron.

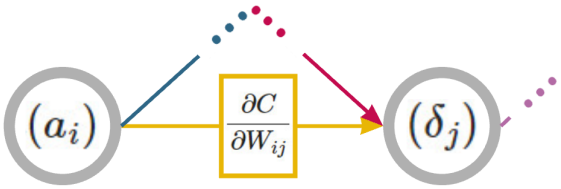


Figure 5.4: Any given network weight connects exactly two neurons $n_i \rightarrow n_j$. Its local gradient $\frac{\partial C}{\partial W_{ij}}$ is given by the activation value a_i of the *upstream* neuron n_i, and the delta δ_j of the *downstream* neuron n_j.

The consequence of this relation is that the gradient need only be computed for every *neuron* N_n in a network rather than for every *element* N_Θ, where $N_\Theta >> N_n$. The reduction of $\mathbb{R}^{N_\Theta} \rightarrow \mathbb{R}^{N_n}$ immediately enables improved scaling for perturbative gradient-estimation methods. This same insight underpins the *node perturbation* learning algorithm [180, 181, 183, 184], where finite difference is used to compute only bias gradients, which are applied along with presynaptic activations to perform single-layer backpropagation. Similar to SignSGD, however, node-perturbation is less noise robust than MGD as it expands on any error in gradient estimation by propagating it to other elements. MGD-aligned instead weights updates according to a true measurement of the gradient estimate's strength, making it robust to noise as already demonstrated in Fig. 5.3, though we still may opt to employ single-layer backpropagation for the *final* layer only, where it is an exact, local computation. For the remaining layers, only the signs of a_i and δ_j, reducing the resolution of the task from full-precision to ternary values according to

$$sign\left(\frac{\partial C}{\partial W_{ij}}\right) = sign(a_i) \cdot sign(\delta_j). \tag{5.3}$$

This approach is attractive for specialized hardware implementation because activation values are available locally, only N_n *ternary* gradient elements need be estimated, and only a local ternary multiplication is required for each weight in the network, all of which are desirable qualities for dedicated circuit operations.

5.3.2 Thresholding

The measurement of presynaptic activation a_i is accompanied by a caveat not previously addressed in our above implementation of MGD-Aligned. Presynaptic activations are a *per-sample* quantity. Moreover, dedicated in-memory hardware generally favors sample serialization over batching. Therefore, we must estimate the ternary gradient on a per-sample basis. A nontrivial issue arises in this regime whereby batch-averaging would have to occur over ternarized gradients rather than full-precision gradients. However, $sign(\bar{\nabla}) \neq \overline{sign(\nabla)}$. This is because a ternary summation of gradients across components neglects inter-sample per-component magnitude variance, as visualized in Fig. 5.5 (a). Ostensibly, using batch sizes of 1 should circumvent this issue, but in fact the effect pervades to per-sample updates because MGD and MGD-aligned apply uniform magnitude updates across all components as determined by the ΔC measured for that gradient estimation. Thus, a component might be very large in the positive direction for one sample, and very small in the opposite direction for another sample, but still be subject to equivalent size updates for similar values of ΔC, whose total value may be driven by other components entirely. The consequence to performance is shown in Fig. 5.5 (c).

The solution to this issue proves to be straightforward and involves only one hardware-friendly operation. By thresholding every presynaptic activation, we can approximately capture component-wise variance in our gradient estimation, preserving only the most significant components in each per-sample estimate. By way of Equation 5.3, replacing a presynaptic activation with zero results in a zero-value estimate of the ternary gradient. In the MGD-aligned context, this means no perturbation is made at that component and thus no update is made. The result is that MGD-aligned is now robust to multiplicative variance caused by conflicting gradient signs across samples in addition to its intrinsic robustness to additive noise . See Fig. 5.5 (b) for a visualization. In addition to better tailoring sample-wise gradient estimations to the larger learning objective, this method has the added benefit of increasing the signal of the remaining parameter perturbations. Selecting an appropriate threshold is simply a question of hyperparameter tuning. For a hyperbolic tangent function, we empirically find that some value marginally less than 0.5 tends to be effective, as shown in Fig. 5.5 (c).

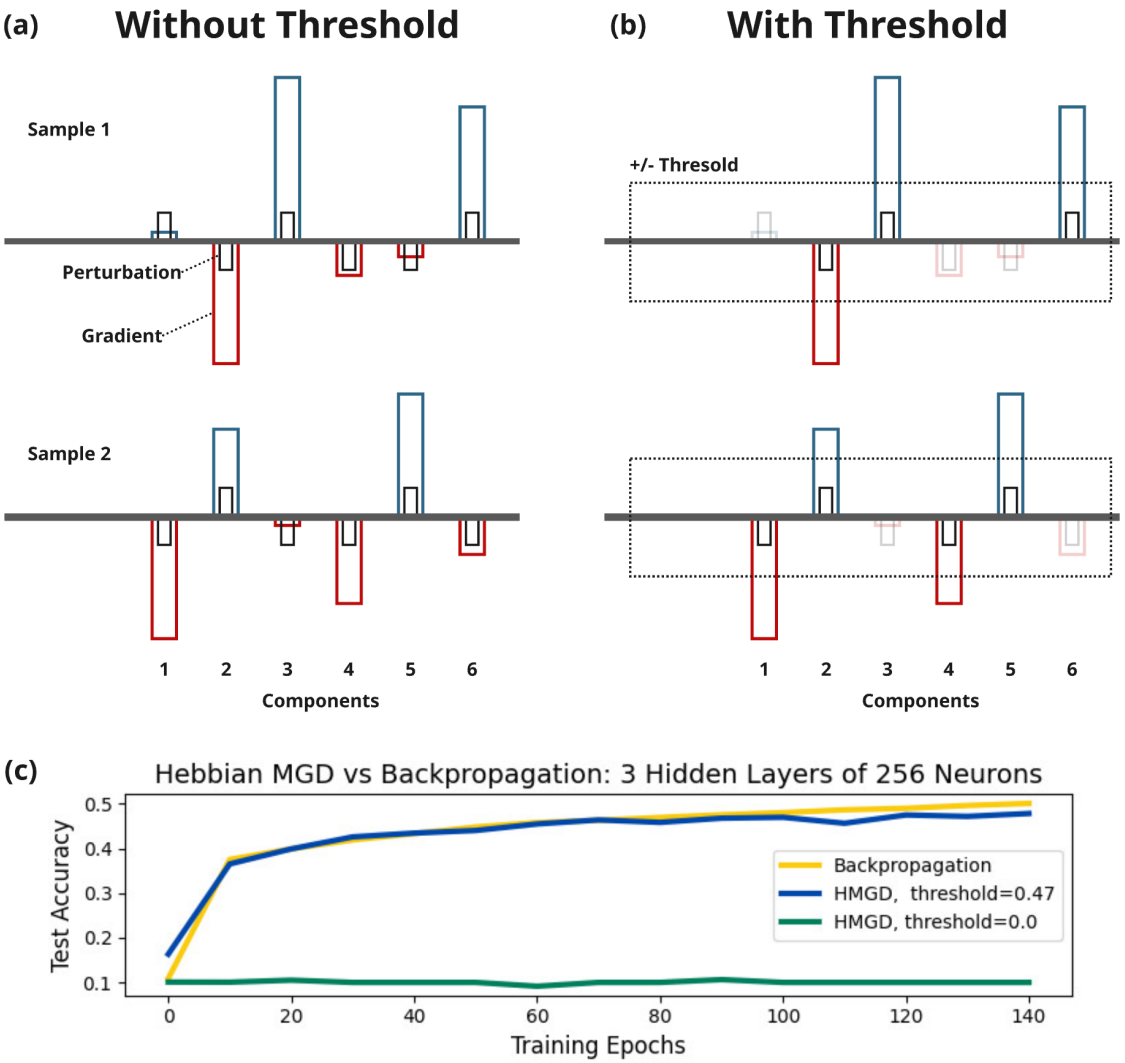


Figure 5.5: (a) Without Threshold: Perturbations are made for every component. We see that while the signed direction may be in agreement with the gradient for each component, the difference in magnitude can widely vary, as seen in component 1. Thus, the inter-sample per-component variance is high, which is problematic because perturbation magnitude remains constant. For roughly equivalent ΔC across samples, these parameters will receive the same size updates in *opposite directions*. This is possible because *intra*-sample component-wise variance can also be high, resulting in magnitudes of ΔC that are driven by the strongest components, misrepresenting the influence weaker ones. **(b) With Threshold:** We see that by setting some $\pm$ threshold value, small components will neither interfere with more significant gradients across samples, nor be subject to large updates within a sample according to other components. **(c) Hebbian MGD:** We see thresholding is necessary for the network to learn on a per-sample basis *and* that it is sufficient to keep pace with full-precision backpropagation on the CIFAR-10 dataset.

5.3.3 Hebbian MGD

In summary, we find that pre-and-postsynaptic information can be used to estimate the ternary gradient in a way that relates local activity with global reward signals and circumvents the weight transport problem of credit assignment [185] in network learning. We therefore make the loose analogy to *three-factor Hebbian learning* [61, 62], and refer to this method as *Hebbian-MGD*, implemented as follows: (1) Threshold the presynaptic activation according to its absolute value, and then ternarize it. (2) Attain a ternarized bias gradient estimation by finite difference or by the novel means presented later in this work. (3) Compute the estimated sign of network weights accordingly. (4) Use this signed gradient to define the perturbation vector on a per-sample basis. (5) Proceed with MGD-aligned learning. Fig. 5.5 (c) demonstrates the efficacy of this method in terms of machine learning performance with respect to the backpropagation gold standard, but unlike backpropagation, Hebbian MGD is a viable choice for neuromorphic hardware.

Having shown that MGD-Aligned performance is applicable to the entirely neuromorphic means of Hebbian MGD, we now turn our attention to further reductions in the problem space of bias gradient estimations.

5.4 DeltaVecs: Spatio-Temporal Stability in Class-Wise Gradients

We now present DeltaVecs, a new approach for representing bias gradients according to intrinsic gradient features, allowing for a reduction of the gradient estimation problem space by 10^6 elements for a network of about a million parameters, and by larger factors for growing network size. These findings directly support the neuromorphic algorithm stack being presented in this paper, but may also apply to advantages in gradient computation for machine learning at large.

The total number of gradient component computations involved with training a neural network is given by

$$\mathcal{O}(N_\Theta \times N_{samples} \times N_{epochs}),$$

where N_Θ is the total number of learnable network parameters, N_{epochs} is the number of epochs required to train the network, $N_{samples}$ is the number of training samples in the target dataset, and $\mathcal{O}(\cdot)$ gives the total number of gradient computations required to learn the data. We quickly see these numbers are impractically large. Even the minimal MNIST dataset has 60,000 samples and might train over ten to a hundred epochs depending on

learning rate. For a small network with only two hidden layers of 128 neurons each, trained on MNIST for only ten epochs, there are *70,969,200,000* gradient component calculations to perform. Hebbian MGD already reduces this space by a factor of N_n/N_Θ because weight derivatives can be attained implicitly from bias gradients. However, the $N_{samples}$ and N_{epochs} factors remain a nontrivial expense.

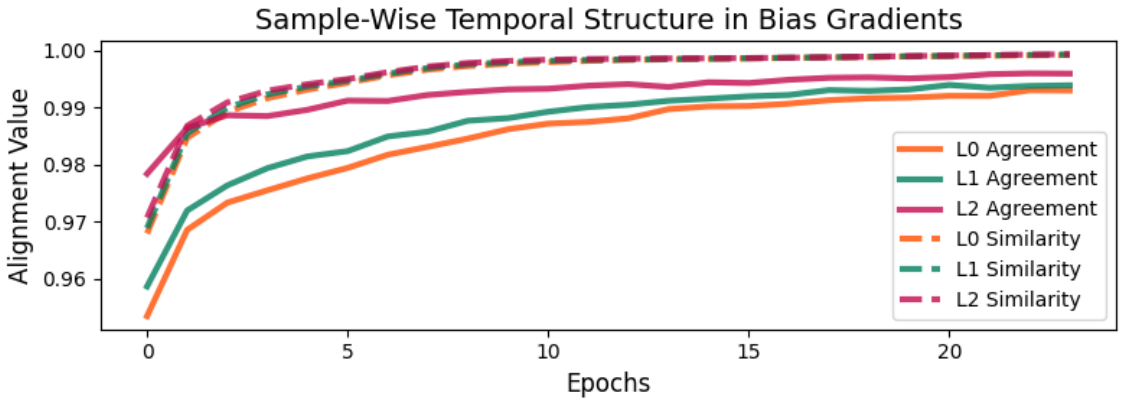


Figure 5.6: Ternary agreement (solid lines) and full-precision cosine similarities (dashed lines) are measured for bias gradients from one epoch to the next during a convergent training procedure on MNIST.

Fortunately, perturbative methods are robust to noisy conditions, as already demonstrated in Sec. 5.2. Rather than compute the gradient for every component, sample, and epoch, why not learn suitable intermediate representations? If gradients have learnable features, perhaps there can be shortcuts to calculating them analytically. There is precedence for feature analysis of neural network gradients themselves [186–190], but because noisy gradients are not generally satisfactory, as we have already seen with SignSGD, these sorts of efforts have not yet been effectively directed toward gradient computation shortcuts. With respect to MGD-requirements, we assess gradient representation quality in time and space by utilization of our agreement measure defined in Equation 5.2 for ternary vectors, and cosine similarity for full-precision.

5.4.1 Sample-Wise Temporal Stability

The first finding we present is that the gradient vector for any given sample does not change drastically over time for a well-conditioned dataset. This is true both of the ternary and full-precision gradients, as seen in Fig. 5.6. While this does allude to the convexity of the loss landscape with respect to that sample, this measure is not descriptive of the gradient

with respect to the entire dataset. Given this finding, we may already implement another compression to the gradient estimation problem by a factor of N_{T_δ}/N_{epoch}, where N_{T_δ} is the number of times the representation of a bias gradient might need to be updated, and $N_{T_\delta} < N_{epoch}$. The assumption is that these vectors might be sufficiently stable over time such that they need not be computed every epoch.

5.4.2 Class-Wise Stability

We see in the left quadrants of Fig. 5.7 that there is no apparent ternary or full-precision similarity across samples in a batch. However, a surprising richness of features becomes salient when these same bias-gradient vectors are organized according to class. Class-common features are cleanly distinguishable even by visual inspection. We can also see the strongest gradient components in the full precision plot (dark purple and bright yellow), tend to align most with these class-wise representations, and should therefore be well-captured in approximations. This offers a massive potential for gradient representation reduction by a factor of $N_{classes}/N_{samples}$. This finding, along with work such as [187–189], suggests that gradient dynamics are largely driven by the structure of the objective function, which requires the minimization of error to one-hot-encoded class labels. While nuanced input data features related to fundamental differences in class are essential for learning [191], these are quickly mapped to more stable gradient targets (formed by gradient directions that maximize class-wise alignment [187]) to guide separable representations of class. This sentiment may also relate to feedback alignment literature [170], which finds that even arbitrary mappings of cost to gradient estimations can produce network learning, as well as recent work on learned feedback mappings [192].

5.4.3 Class-Wise Temporal Stability

Finally, we find that the aforementioned temporal stability of bias gradients applies directly to class-wise representations, measuring $\Lambda = 0.92$ between first and last epoch of a convergent network trained on MNIST. Together with the previous structure-enable gradient compressions, we arrive at the final reduction

$$\mathcal{O}(N_\Theta \times N_{samples} \times N_{epochs}) \rightarrow \mathcal{O}(N_n \times N_{classes} \times N_{T_\delta}), \tag{5.4}$$

which for our sample MNIST network is

$$N_\nabla = 70,969,200,000 \rightarrow N_{\nabla_\delta} = 319,200$$

corresponding to $222,335\times$ *fewer gradient component calculations* N_∇.

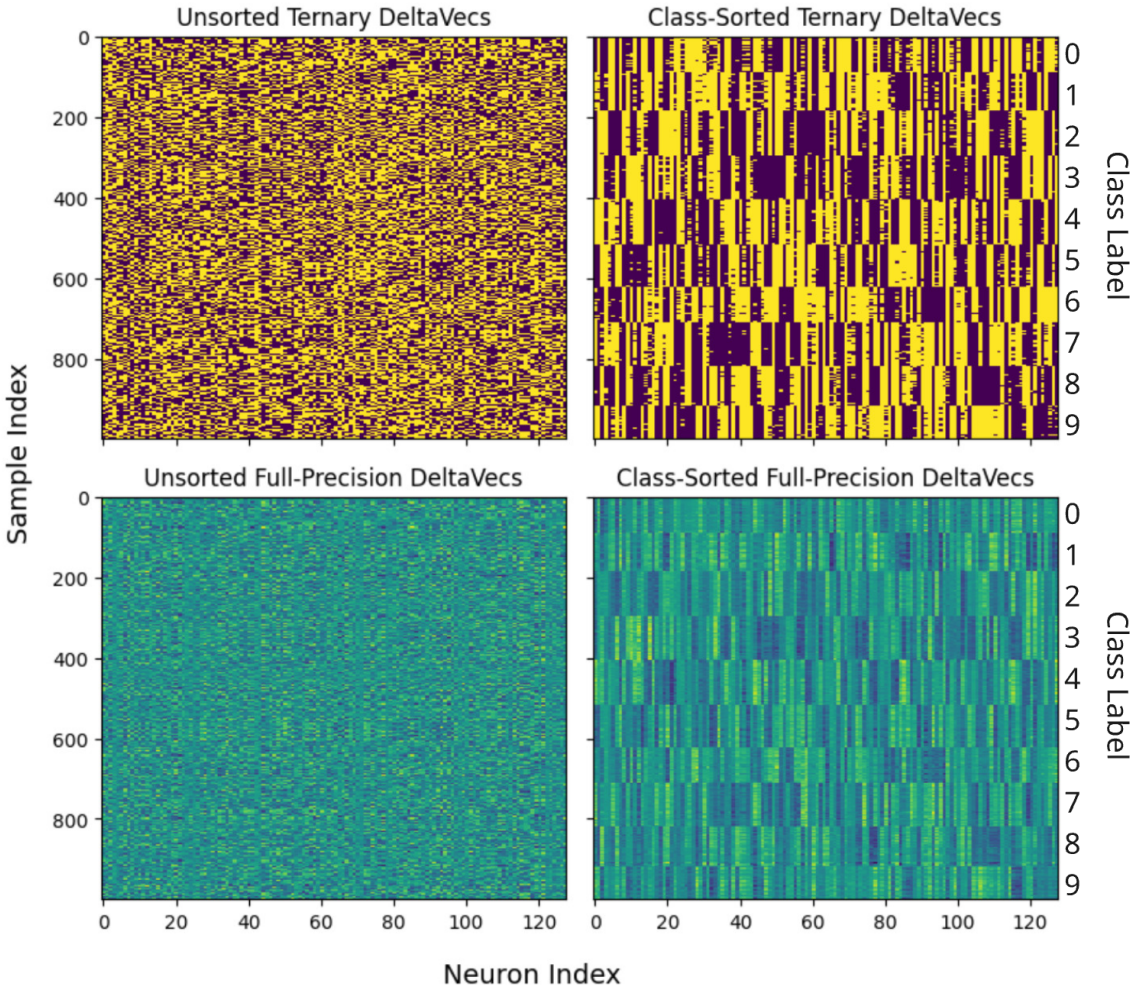


Figure 5.7: (Top Left) Unsorted ternary bias gradients for a batch of training examples. Rows correspond to a sample, columns to a gradient component. (Bottom left) Unsorted full-precision bias gradients. (Top Right) Class-sorted ternary bias gradients. Strong per-class structure is now salient. (Bottom Right) Class-sorted full-precision bias gradients. Class-wise structure is also salient in the **magnitude** of components, implying strong gradient correlation across samples for a given class.

5.4.4 DeltaVecs

We therefore introduce *DeltaVecs*, temporally robust class-wise representations of post-synaptic deltas (bias gradients) as an asset to efficient gradient computation methods. Equipped with massive savings in gradient estimation by way of DeltaVecs and Hebbian MGD, we next propose a way to learn, store, and produce DeltaVecs by neuromorphic means.

5.5 Astrocytes: Intelligent Perturbation-Generating Subnetworks

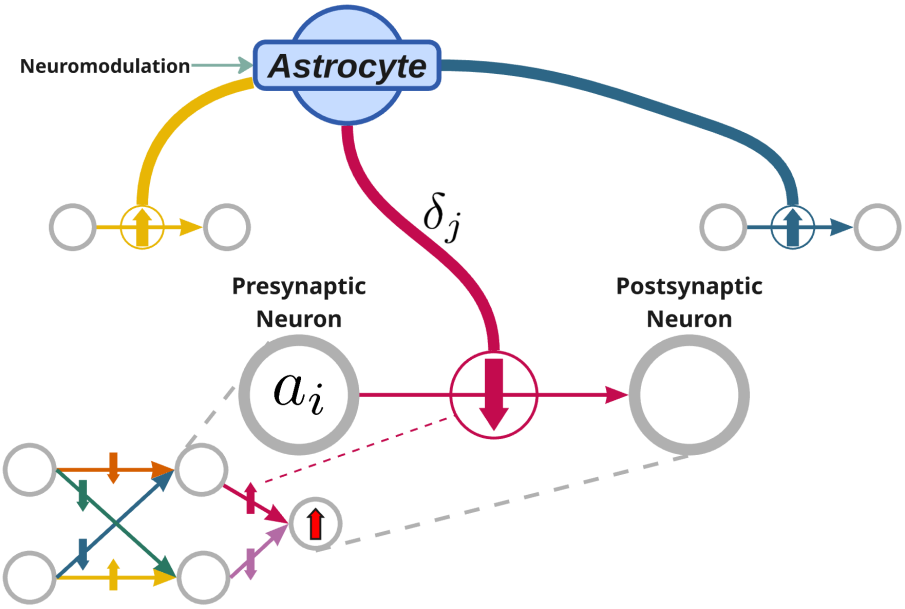


Figure 5.8: Astrocytic Model for Driving Perturbations: A mock-up for potential astrocytic relationship with perturbative learning. An astrocyte may connect to multiple synapses and be responsible for modulating their synaptic excitability (perturbing their weights). They furthermore may be involved with perturbing in favorable directions according to locally available pre-and-postsynaptic information, and even DeltaVec-like gradient information.

We present two findings, (1) that subnetworks offer a viable method for learning, storing, and estimating DeltaVecs by simple means, and (2) that this configuration maps well onto known and proposed astrocytic operations in the brain. These two findings complete an end-to-end stack that can guide perturbative learning to be orders of magnitude more efficient in a fully neuromorphic way.

5.5 Astrocytes: Intelligent Perturbation-Generating Subnetworks

Astrocytes are known to modulate synaptic excitability as a function of network activity and may participate in learning mechanics such as three-factor learning [63, 73–75]. Moreover, a single astrocyte may connect with thousands, or even millions, of synapses [76, 77]. Similarly, the learning approaches described thus far in this work aim to drive perturbations as a function of network activity and class context in order to incite Hebbian-like three-factor learning. While there are myriad parallels to draw, we outline here only one of many such possible astromorphic computing paradigms.

Fig. 5.8 depicts a proposed model whereby an astrocyte is responsible for parameter perturbations, driven according to local activation and estimated deltas. In a multi-astrocyte scheme, each astrocyte might be responsible for its own class. Indeed, astrocytes have been shown to have context-awareness even when unsupervised [193,194] and respond to the dynamical nature of stimulus [63]. The subset of synapses to which a class-specific astrocyte is connect could be related to the same subset of nonzero values produced by our aforementioned thresholding mechanism. A fact that supports this suggestion is that astrocytes have been shown to make local adjustments to their synaptic groups, moving a connection from one synapse to another as needed. In a supervised setting, neuromodulation might aid with class awareness, as well as gradient estimation.

A more sophisticated vision for astromorphic computation is where the onus of learning to estimate DeltaVecs is also an astrocytic responsibility. Due to the strong class-wise correlations of DeltaVecs, this should not prove inconceivable. The task can be as simple as each astrocyte sampling its corresponding class-DeltaVec every τ_δ iterations, and storing them in its subnetwork weights, which perform the linear mapping $\mathcal{A} : y \rightarrow \vec{\delta}$. These DeltaVecs could be sparsely generated only every τ_δ samples by perturbative learning. Another version of this implementation could be that the astrocytes participate in online representation forming of DeltaVecs, with only sparse training examples as determined by τ_δ. This hypothesis is tested and it is found that subnetworks are able to learn DeltaVecs and estimate them online, achieving $\Lambda > 90\%$ across all layers. Finally, astrocytes could potentially perform a second order optimization by learning to estimate gradient dynamics. While all of these possibilities are compelling, we show in the next section that the *simplest* proposed implementation (periodic DeltaVec-sampling) is able to realize superior neuromorphic performance.

5.6 $\vec{\delta}$-MGD: A Neuromorphic Learning Algorithm

Finally, we arrive at a fully neuromorphic algorithm, and we find it reliably performs within 1% of backpropagation with $> 10^5$ reduction factors in N_∇ gradient component computations using only simple hardware-plausible operations across a number of architectures. We call this method $\vec{\delta}$-MGD, formally described in Alg. 2. In addition to

Algorithm 2 The $\vec{\delta}$-MGD Algorithm

Initialize network parameters Θ
for i in $N_{samples} \cdot N_{epochs}$ iterations **do**
  Input new training example x, y
  if i mod $\tau_\delta = 0$ **then**
    for class in $N_{classes}$ **do**
      Compute $\vec{\delta}$
      Update representation of $\vec{\delta}_{class}$
    end for
  end if
  Compute forward pass
  Store activations a locally, and cost C_0 globally
  Estimate gradient: $\hat{\nabla}_i \leftarrow a_i \cdot \vec{\delta}_i$
  Define $\tilde{\theta}$ according to $\hat{\nabla}_i$ and store locally
  Perturb all network parameters Θ according to $\tilde{\theta}$
  Perform forward pass
  Measure change in global cost: $\Delta C = \tilde{C} - C_0$
  Update parameters: $\Theta_i \leftarrow \eta \Delta C \cdot \tilde{\theta}_i \cdot \gamma C_0$
end for=0

incorporating DeltaVecs, there are few changes to previous MGD variants. Firstly, full precision gradient estimates are permitted as perturbation vectors because they are likely to estimate the sign correctly, thereby adding better-than-random (or uniform) magnitude information and offering an alternative to thresholding for the treatment of component-wise variance. This also implies that when a new DeltaVec is sampled, the update for that sample will be equivalent to backpropagation. Second, the final update is weighted by the cost C_0, according to coefficient and hyperparameter γ, to better represent inter-sample variance in gradient magnitude. The added complexity of these additions is negligible, but if hardware constraints become an issue, activation-thresholded ternary gradient estimations work nearly as well to drive perturbations. The performance of $\vec{\delta}$-MGD is the best

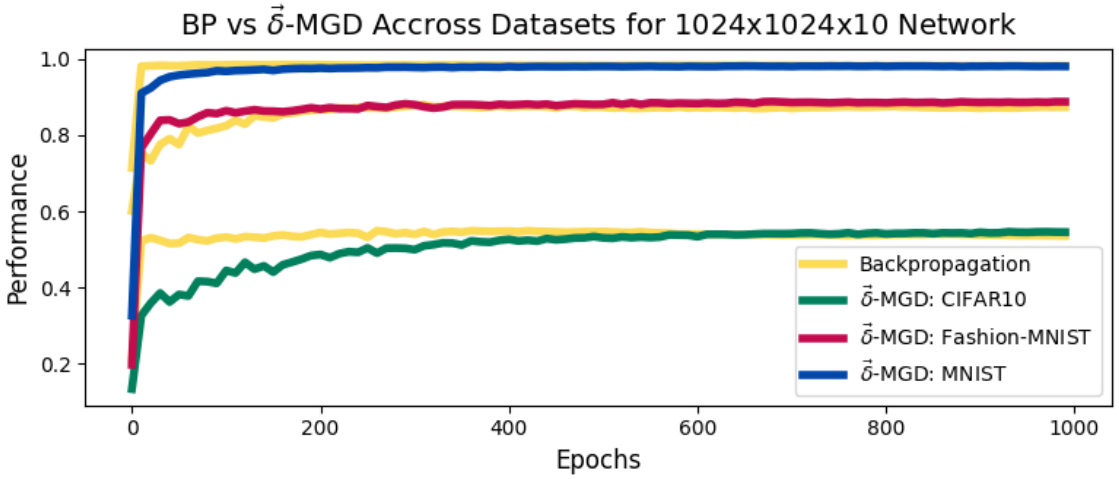


Figure 5.9: We find that $\vec{\delta}$-MGD converges to backpropagation performance on all three datasets, with smooth and stable learning. While backpropagation reaches similar performance more quickly, it is requiring thousands of times more gradient component computations, which would not be implementable in neuromorphic hardware. $\vec{\delta}$-MGD uses only sparse hardware-friendly gradient estimation methods for the same performance.

of any in the MGD family of algorithms, is entirely neuromorphic, and closely adheres to backpropagation, as seen in Fig. 5.9 and enumerated in terms of performance and savings in Sec. 5.7.

5.7 Results

See Table 5.2 for an overview of performance results. We find $\vec{\delta}$-MGD is poised to cope with millions of parameters effectively with orders of magnitude computational savings. It is worth noting that these savings may be even more dramatic depending on the hardware-specific expense of forward versus backward passes. Efficiencies outlined by our N_∇ calculations could be considerably compounded by the speed and energy efficiencies of dedicated hardware. While listed savings are exactly true with respect to the factor fewer gradient component calculations required by $\vec{\delta}$-MGD, a complete comparison of total speed, efficiency, and performance, whether with respect to backpropagation or neuromorphic means, can only be fully understood on a per-hardware basis and as a function of implementation optimization. We expect $\vec{\delta}$-MGD to benefit from competitive hardware implementations.

Furthermore, these mechanisms map to more sophisticated network architectures such as CNNs and transformers. For CNNs, DeltaVecs are computed in the convolutional layers

Table 5.2: $\vec{\delta}$-MGD vs Backpropagation, Performance and Efficiency

Network	Dataset	BP	$\vec{\delta}$-MGD	% BP	×Savings
DenseNet 3* 128x128 N_Θ=411,146	MNIST	97.9%	97.2%	**99.3%**	$\mathbf{2 \cdot 10^5}$
	Fashion	87.9%	87.2%	**99.2%**	$\mathbf{2 \cdot 10^5}$
	CIFAR	51.9%	51.2%	**98.7%**	$\mathbf{8 \cdot 10^5}$
DenseNet 3* 1024x1024 N_Θ=855,050	MNIST	98.6%	98.2%	**99.6%**	$\mathbf{5 \cdot 10^5}$
	Fashion	88.0%	88.8%	**100.9%**	$\mathbf{5 \cdot 10^5}$
	CIFAR	55.2%	54.8%	**98.0%**	$\mathbf{1 \cdot 10^6}$
CNN 3* 6C+4D $N_\Theta \approx 10^6$	MNIST	99.4%	99.2%[a]	**99.8%**	–
	Fashion	92.3%	90.3%[a]	**97.8%**	–
	CIFAR	74.0%	70.9%[a]	**95.8%**	–
Transformer 3* NanoGPT $N_\Theta \approx 10^8$	Shake-speare	val loss 3.0	val loss 3.[b]	**95.0%**	–

[a]For implementation simplicity, Hebbian MGD used here.

[b]Reduced training set size, only projection matrices trained w/ $\vec{\delta}$-MGD, remainder w/ backpropagation.

for kernel weights rather than biases because they are less numerous, and then applied toward Hebbian MGD. Transformer results are with respect to $\vec{\delta}$-MGD training projection matrices in NanoGPT [195] on the Shakespeare next token prediction task [196]. The remainder of parameters are trained with backpropagation. We find that all but the the K-cache parameters have stable DeltaVecs, but that per-sample implementation is sufficiently complex to save for future work, though these results encourage the potential for energy savings in large language models.

5.8 Methods

All simulations are implemented on GPUs using the JAX Python package [197]. A functional repository and code to produce all plots can be found at: `https://github.com/ryangitsit/delta-mgd`. Anytime two algorithms are compared, the best of their performances over a relevant hyperparameter sweep are presented. Adam optimizer and decay are used for final results only.

5.9 Discussion: Gradient Dynamics, Astrocytic Learning, and Hardware Implementations

We began this work with the empirical finding that a perturbation is improved by increasing sign-agreement with the true ternary gradient, thereby founding MGD-aligned. We then determined a hardware plausible method for computing the ternary gradient with locally available information using Hebbian-MGD. Next, by introducing DeltaVec class-wise representations of postsynaptic bias gradients, we show the problem space of ascertaining the postsynaptic information required by Hebbian MGD is reduced by several orders of magnitude, and we then show this is a task well-suited for astrocyte-like subnetworks. Finally, we observe that this stack of hardware-viable neuro-inspired mechanisms achieves backpropagation-like performance at a minuscule fraction of required gradient-component calculations for dense networks, CNNs, and transformers. These findings realize a number of implications across neuroscience, physical hardware, and machine learning.

Neuroscience: Astrocytes and perturbative methods may be strongly related in terms of their phenomenology and computational role in learning. Future work ought to focus on more tightly coupling known biological dynamics with perturbative methods.

Hardware: Global broadcast (reward signal), local perturbations, local memory, minimal finite difference, Hebbian-like mechanisms, and subnetworks are all plausible in neuromorphic hardware. Mapping this suite of operations should be investigated in known neuromorphic hardware [15–17, 19, 20], as well as explored in emerging systems that could opt to tailor physical structures to this modest recipe of operations.

Machine learning: Gradient dynamics are well-studied, but applying DeltaVecs toward short-cutting gradient computations is a novel finding. Our preliminary results on CNNs and transformers ought to be extended to pure $\vec{\delta}$-MGD implementations and on larger scales. Second-order optimization methods, by which gradient dynamics are learned early

in training and utilized thereafter for continued network training, also present an area offering of potentially massive computational savings. The viability of learning gradient dynamics is supported by Fig. 5.6, which suggests smooth bias gradient Hessians.

In conclusion, we have presented a complete neuromorphic algorithm that is neuro-inspired, hardware-plausible, and machine learning performant. This work offers a number of further investigations to pursue, many of which may interest a diversity of academic and industry communities.

Acknowledgment

This work was made possible by the institutional support from the National Institute of Standards and Technology and the University of Colorado Boulder.

Disclaimer

Commercial hardware are mentioned to contextualize algorithm applicability, but this does not imply endorsement of any product or service by NIST.

6

Closing

6.1 Discussion

6.1.1 Retrospective

In Chapter 1 we introduced neuromorphic computing, motivated its importance, and outlined the sorts of advancements that could maximize its potential impact, namely on-chip neuromorphic learning algorithms. We then presented two original works in Chapters 2 and 3 that developed the arbor update rule, an *in situ* learning algorithm for superconducting optoelectronic networks and any other dendritically-enabled neuromorphic hardware. In Chapter 4 we apply the existing model-free MGD to realize hardware-agnostic gradient descent in recurrent neural networks, showing promise for signal processing and temporal computing. Finally, in Chapter 5, we introduce the novel $\vec{\delta}$-MGD algorithm, realizing backpropagation-competitive gradient descent using only hardware-friendly operations. In short, we have posited a pertinent domain in neuromorphic computing (on-chip gradient descent) and presented considerable efforts toward the advancement of this domain. Moreover, we feel these contributions merit future work toward the continuing pursuit of bio-inspired algorithms for hardware native neural computing.

6.1.2 Future Work

The hardware-plausible results for gradient descent algorithms across applications, function-types, and network architectures presented in this dissertation encourage pursuits in on-chip neuromorphic learning.

Machine Learning: This dissertation concluded with a proposed backpropagation-competitive algorithm, demonstrating commensurate performance on a number of learning tasks and ar-

chitectures. These results, however, are not but a drop in the bucket of the immense demonstrations of backpropagation-driven intelligence across industry and academia. Therefore, all works presented here must apply considerable work to adapt the solid foundations of these proof-of-concept results toward industry-standard impact in AI and computing.

Hardware: In particular, the proposed algorithms should proceed from simulated results to physical in-hardware implementations, with real-world applications in mind. That progress has been shown on learning with only hardware-feasible operations, full neuromorphic implementations of intelligence appear increasingly feasible, offering promise in energy efficient edge computing and large scale distributed compute. Although presented results have been specially tailored to functions conducive for physical implementation, further work on these same algorithms should seek to expand into the spiking domain to harness all the speed, sparsity, and temporal sophistication that neuromorphic hardware has to offer.

Neuroscience: Investigations in neuroscience with respect to feasible learning methods over dendrites and astrocytic subnetworks would furthermore be of interest to connect presented results with biological realism. While serving academia and industry with useful advancements in computing is of paramount interest, it does not supersede the greater question with which this dissertation was introduced: *How does intelligence arise in the brain?* There are many approaches to progress on this query, but neuromorphic computing may best suit Feynman's final challenge to the scientific community, *"What I cannot create, I do not understand"* (Feynman 1988). We have thus labored on brain-inspired intelligence in neuromorphic systems not only for gains in AI computing, but dually toward bearing within the greater scheme of pursing understanding in natural intelligence.

Bibliography

[1] Carver Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, 2002. 1

[2] Giacomo Indiveri and Shih-Chii Liu. Memory and information processing in neuromorphic systems. *Proceedings of the IEEE*, 103(8):1379–1397, 2015. 1, 25

[3] Catherine D Schuman, Thomas E Potok, Robert M Patton, J Douglas Birdwell, Mark E Dean, Garrett S Rose, and James S Plank. A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint arXiv:1705.06963*, 2017. 1, 2

[4] Dhireesha Kudithipudi, Catherine Schuman, Craig M Vineyard, Tej Pandit, Cory Merkel, Rajkumar Kubendran, James B Aimone, Garrick Orchard, Christian Mayr, Ryad Benosman, et al. Neuromorphic computing at scale. *Nature*, 637(8047):801–812, 2025. 1

[5] Ryan O'Loughlin, Bryce Primavera, and Jeffrey Shainline. Dendritic learning in superconducting optoelectronic networks. In *Proceedings of the 2023 International Conference on Neuromorphic Systems*, pages 1–8, 2023. 1, 31

[6] Ryan O'Loughlin, Bryce Primavera, and Jeffrey Shainline. Spatiotemporal dendritic processing in superconducting optoelectronic networks. In *2024 International Conference on Neuromorphic Systems (ICONS)*, pages 234–241. IEEE, 2024. 1, 31

[7] Ryan O'Loughlin, Nick Skuda, Bakhrom Oripov, Bradley Hayes, Adam McCaughan, and Sonia Buckley. Model-free multiplexed gradient descent: Neuromorphic learning in recurrent networks. In *2025 International Conference on Neuromorphic Systems (ICONS)*, pages 104–111. IEEE, 2025. 1, 30, 31

BIBLIOGRAPHY

[8] Ryan O'Loughlin, Bakhrom Oripov, Nick Skuda, Noah Chongsiriwatana, Ian Whitehouse, Wolfgang Losert, Bradley Hayes, Adam McCaughan, and Sonia Buckley. multiplexed gradient descent: Perturbative learning with astrocytes. In *2026 Neuro-Inspired Computational Elements (NICE) Conference*. IEEE, 2026. 1, 8, 18, 31, 97

[9] Alan Mathison Turing et al. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936. 2

[10] John Von Neumann. First draft of a report on the edvac. *IEEE Annals of the History of Computing*, 15(4):27–75, 1993. 2

[11] Gordon E Moore et al. Cramming more components onto integrated circuits, 1965. 2, 5

[12] Chris A Mack. Fifty years of moore's law. *IEEE Transactions on semiconductor manufacturing*, 24(2):202–207, 2011. 2

[13] Fahad Javed, Qing He, Lance E Davidson, John C Thornton, Jeanine Albu, Lawrence Boxt, Norman Krasnow, Marinos Elia, Patrick Kang, Stanley Heshka, et al. Brain and high metabolic rate organ mass: contributions to resting energy expenditure beyond fat-free mass. *The American journal of clinical nutrition*, 91(4):907–912, 2010. 2

[14] Yuzhuo Li, Mariam Mughees, Yize Chen, and Yunwei Ryan Li. The unseen ai disruptions for power grids: Llm-induced transients. *arXiv preprint arXiv:2409.11416*, 2024. 2, 6

[15] Mike Davies, Andreas Wild, Garrick Orchard, Yulia Sandamirskaya, Gabriel A Fonseca Guerra, Prasad Joshi, Philipp Plank, and Sumedh R Risbud. Advancing neuromorphic computing with loihi: A survey of results and outlook. *Proceedings of the IEEE*, 109(5):911–934, 2021. 2, 5, 22, 34, 96, 117

[16] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, et al. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE transactions on computer-aided design of integrated circuits and systems*, 34(10):1537–1557, 2015. 2, 5, 34, 117

[17] JB Pedersen et al. Spinnaker 2: A 10 million core processor system for brain simulation and machine learning. *Communicating Process Architectures 2017 & 2018: WoTUG-39 & WoTUG-40*, 70:277, 2019. 2, 5, 13, 117

[18] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee micro*, 38(1):82–99, 2018. 2, 5, 11, 13

[19] Christian Pehle, Sebastian Billaudelle, Benjamin Cramer, Jakob Kaiser, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, Aron Leibfried, Eric Müller, and Johannes Schemmel. The brainscales-2 accelerated neuromorphic system with hybrid plasticity. *Frontiers in Neuroscience*, 16:795876, 2022. 2, 5, 22, 34, 96, 117

[20] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R Chandrasekaran, Jean-Marie Bussat, Rodrigo Alvarez-Icaza, John V Arthur, Paul A Merolla, and Kwabena Boahen. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, 102(5):699–716, 2014. 2, 5, 117

[21] Chit-Kwan Lin, Andreas Wild, Gautham N Chinya, Yongqiang Cao, Mike Davies, Daniel M Lavery, and Hong Wang. Programming spiking neural networks on intel's loihi. *Computer*, 51(3):52–61, 2018. 5

[22] Garrick Orchard, Cedric Meyer, Ralph Etienne-Cummings, Christoph Posch, Nitish Thakor, and Ryad Benosman. Hfirst: A temporal approach to object recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(10):2028–2040, 2015. 5

[23] Chris Yakopcic, Nayim Rahman, Tanvir Atahary, Tarek M Taha, and Scott Douglass. Solving constraint satisfaction problems using the loihi spiking neuromorphic processor. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1079–1084. IEEE, 2020. 5

[24] Steve B Furber, Francesco Galluppi, Steve Temple, and Luis A Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014. 5, 11

BIBLIOGRAPHY

[25] Javier Navaridas, Mikel Luján, Jose Miguel-Alonso, Luis A Plana, and Steve Furber. Understanding the interconnection network of spinnaker. In *Proceedings of the 23rd international conference on Supercomputing*, pages 286–295, 2009. 5

[26] Evangelos Stromatias, Francesco Galluppi, Cameron Patterson, and Steve Furber. Power analysis of large-scale, real-time neural networks on spinnaker. In *The 2013 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2013. 5

[27] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014. 5

[28] Steve K Esser, Rathinakumar Appuswamy, Paul Merolla, John V Arthur, and Dharmendra S Modha. Backpropagation for energy-efficient neuromorphic computing. *Advances in neural information processing systems*, 28, 2015. 5

[29] Giacomo Indiveri, Bernabé Linares-Barranco, Tara Julia Hamilton, André van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, Piotr Dudek, Philipp Häfliger, Sylvie Renaud, et al. Neuromorphic silicon neuron circuits. *Frontiers in neuroscience*, 5:73, 2011. 5

[30] Johannes Schemmel, Daniel Brüderle, Andreas Grübl, Matthias Hock, Karlheinz Meier, and Sebastian Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *2010 ieee international symposium on circuits and systems (iscas)*, pages 1947–1950. IEEE, 2010. 5

[31] Jeffrey M Shainline, Sonia M Buckley, Richard P Mirin, and Sae Woo Nam. Superconducting optoelectronic circuits for neuromorphic computing. *Physical Review Applied*, 7(3):034013, 2017. 5, 34

[32] Bryce A Primavera, Saeed Khan, Samuel R Adler, and Jeffrey M Shainline. Programmable synapses and dendritic circuits for superconducting optoelectronic neuromorphic computing. In *2024 International Conference on Neuromorphic Systems (ICONS)*, pages 277–281. IEEE, 2024. 5, 96

[33] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943. 7

[34] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958. 7

[35] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016. 8, 13, 22

[36] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. 8, 13

[37] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989. 8, 13

[38] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010. 8

[39] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006. 8

[40] John Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. *Advances in neural information processing systems*, 2, 1989. 8

[41] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 2002. 8

[42] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA, 2013. 8

[43] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 4(5):11, 2015. 8

[44] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. 8

[45] Louis Lapicque. Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. *Journal de physiologie et de pathologie générale*, 9:620–635, 1907. 8

BIBLIOGRAPHY

[46] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002. 8, 9

[47] Peter Dayan and Laurence F Abbott. *Theoretical neuroscience: computational and mathematical modeling of neural systems*. MIT press, 2005. 9, 10

[48] F Rieke, D Warland, and W Bialek. Coding efficiency and information rates in sensory neurons. *EPL (Europhysics Letters)*, 22(2):151–156, 1993. 10

[49] Wulfram Gerstner, Werner M Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014. 10, 21

[50] AL Hodgkin and AF Huxley. A quantitative description of membrane current. *J. Physiol*, 1952. 11

[51] Christof Koch. *Biophysics of computation: information processing in single neurons*. Oxford university press, 2004. 12

[52] Bertil Hille. Ionic channels in excitable membranes. current problems and biophysical approaches. *Biophysical journal*, 22(2):283–294, 1978. 12

[53] S Ramón Y Cajal, DR Azoulay, et al. Histology of the nervous system: of man and vertebrates. 1995. 12

[54] Eric R Kandel, James H Schwartz, Thomas M Jessell, Steven Siegelbaum, A James Hudspeth, Sarah Mack, et al. *Principles of neural science*, volume 4. McGraw-hill New York, 2000. 12

[55] Michael London and Michael Häusser. Dendritic computation. *Annu. Rev. Neurosci.*, 28(1):503–532, 2005. 12

[56] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology press, 2005. 13, 21

[57] Guo-qiang Bi and Mu-ming Poo. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of neuroscience*, 18(24):10464–10472, 1998. 13, 21

[58] Sen Song, Kenneth D Miller, and Larry F Abbott. Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nature neuroscience*, 3(9):919–926, 2000. 13, 21

[59] Peter U Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience*, 9:149773, 2015. 13, 21, 22

[60] Robert S Zucker and Wade G Regehr. Short-term synaptic plasticity. *Annual review of physiology*, 64(1):355–405, 2002. 14

[61] Nicolas Frémaux and Wulfram Gerstner. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in neural circuits*, 9:85, 2016. 14, 22, 108

[62] Wulfram Gerstner, Marco Lehmann, Vasiliki Liakoni, Dane Corneil, and Johanni Brea. Eligibility traces and plasticity on behavioral time scales: experimental support of neohebbian three-factor learning rules. *Frontiers in neural circuits*, 12:53, 2018. 14, 22, 108

[63] Kate M O'Neill, Emanuela Saracino, Barbara Barile, Nicholas J Mennona, Maria Grazia Mola, Spandan Pathak, Tamara Posati, Roberto Zamboni, Grazia P Nicchia, Valentina Benfenati, et al. Decoding natural astrocyte rhythms: dynamic actin waves result from environmental sensing by primary rodent astrocytes. *Advanced Biology*, 7(6):2200269, 2023. 14, 19, 113

[64] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019. 17, 25

[65] Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke. The heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 33(7):2744–2757, 2020. 17

[66] Aaron Voelker, Ivana Kajić, and Chris Eliasmith. Legendre memory units: Continuous-time representation in recurrent neural networks. *Advances in neural information processing systems*, 32, 2019. 17

[67] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33:1474–1487, 2020. 17, 80, 84, 87

[68] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021. 17

[69] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First conference on language modeling*, 2024. 17

[70] Herbert Jaeger. Adaptive nonlinear system identification with echo state networks. *Advances in neural information processing systems*, 15, 2002. 18

[71] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002. 18

[72] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 2002. 18

[73] Min Wu, Xudong Wu, and Pietro De Camilli. Calcium oscillations-coupled conversion of actin travelling waves to standing oscillations. *Proceedings of the National Academy of Sciences*, 110(4):1339–1344, 2013. 19, 113

[74] Gertrudis Perea, Marta Navarrete, and Alfonso Araque. Tripartite synapses: astrocytes process and control synaptic information. *Trends in neurosciences*, 32(8):421–431, 2009. 19, 113

[75] Alfonso Araque, Vladimir Parpura, Rita P Sanzgiri, and Philip G Haydon. Tripartite synapses: glia, the unacknowledged partner. *Trends in neurosciences*, 22(5):208–215, 1999. 19, 113

[76] Nancy Ann Oberheim, Takahiro Takano, Xiaoning Han, Wei He, Jane HC Lin, Fushun Wang, Qiwu Xu, Jeffrey D Wyatt, Webster Pilcher, Jeffrey G Ojemann, et al. Uniquely hominid features of adult human astrocytes. *Journal of Neuroscience*, 29(10):3276–3287, 2009. 19, 113

[77] Eric A Bushong, Maryann E Martone, Ying Z Jones, and Mark H Ellisman. Protoplasmic astrocytes in ca1 stratum radiatum occupy separate anatomical domains. *Journal of Neuroscience*, 22(1):183–192, 2002. 19, 113

[78] TM Mitchell. Version spaces: An approach to concept learning[ph. d. thesis]. 1979. 20

[79] Stephen Grossberg. Adaptive pattern classification and universal recoding: I. parallel development and coding of neural feature detectors. *Biological cybernetics*, 23(3):121–134, 1976. 21

[80] Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982. 21

[81] Natalia Caporale and Yang Dan. Spike timing–dependent plasticity: a hebbian learning rule. *Annu. Rev. Neurosci.*, 31(1):25–46, 2008. 21

[82] Friedemann Zenke and Wulfram Gerstner. Hebbian plasticity requires compensatory processes on multiple timescales. *Philosophical transactions of the royal society B: biological sciences*, 372(1715), 2017. 21

[83] Steve Furber and Petruț Bogdan. *Spinnaker-a spiking neural network architecture.* Emerald Group Publishing, 2020. 22, 34, 96

[84] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951. 23, 86

[85] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006. 24

[86] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. 24

[87] Stephen Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive science*, 11(1):23–63, 1987. 25, 81

[88] Francis Crick. The recent excitement about neural networks. *Nature*, 337(6203):129–132, 1989. 25

[89] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020. 25

[90] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 25, 89

[91] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7(1):13276, 2016. 27

[92] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. *Advances in neural information processing systems*, 29, 2016. 27

[93] Brian Crafton, Abhinav Parihar, Evan Gebhardt, and Arijit Raychowdhury. Direct feedback alignment with sparse connections for local learning. *Frontiers in neuroscience*, 13:525, 2019. 27

[94] Charlotte Frenkel, Martin Lefebvre, and David Bol. Learning without feedback: Fixed random learning signals allow for feedforward training of deep neural networks. *Frontiers in neuroscience*, 15:629892, 2021.

[95] Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24, 2017. 27, 28

[96] John J Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the national academy of sciences*, 81(10):3088–3092, 1984. 27

[97] Jérémie Laydevant, Maxence Ernoult, Damien Querlioz, and Julie Grollier. Training dynamical binary neural networks with equilibrium propagation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4640–4649, 2021. 28

[98] Axel Laborieux, Maxence Ernoult, Benjamin Scellier, Yoshua Bengio, Julie Grollier, and Damien Querlioz. Scaling equilibrium propagation to deep convnets by drastically reducing its gradient estimator bias. *Frontiers in neuroscience*, 15:633674, 2021. 28

[99] Jack Kendall, Ross Pantone, Kalpana Manickavasagam, Yoshua Bengio, and Benjamin Scellier. Training end-to-end analog neural networks with equilibrium propagation. *arXiv preprint arXiv:2006.01981*, 2020. 28

[100] Lewis Fry Richardson. Ix. the approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. *Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character*, 210(459-470):307–357, 1911. 28, 82, 99, 100

[101] James C Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE transactions on automatic control*, 37(3):332–341, 2002. 28, 29, 30

[102] Adam N McCaughan, Bakhrom G Oripov, Natesh Ganesh, Sae Woo Nam, Andrew Dienstfrey, and Sonia M Buckley. Multiplexed gradient descent: Fast online training of modern datasets on hardware neural networks without backpropagation. *APL Machine Learning*, 1(2), 2023. 30

[103] Bakhrom G Oripov, Andrew Dienstfrey, Adam N McCaughan, and Sonia M Buckley. Scaling of hardware-compatible perturbative training algorithms. *APL Machine Learning*, 3(2), 2025. 30, 82, 83, 86, 95, 99, 100, 103

[104] Ila R Fiete and H Sebastian Seung. Gradient learning in spiking neural networks by dynamic perturbation of conductances. *Physical review letters*, 97(4):048104, 2006. 30

[105] Justin Werfel, Xiaohui Xie, and H Seung. Learning curves for stochastic gradient descent in linear feedforward networks. *Advances in neural information processing systems*, 16, 2003. 30

[106] Ila R Fiete, Michale S Fee, and H Sebastian Seung. Model of birdsong learning based on gradient estimation by dynamic perturbation of neural conductances. *Journal of neurophysiology*, 98(4):2038–2057, 2007. 30

[107] Vincent F Koosh and Rodney M Goodman. Analog vlsi neural network with digital perturbative learning. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 49(5):359–368, 2002. 30

[108] Ryan O'Loughlin, Bryce Primavera, and Jeffrey Shainline. Dendritic learning in superconducting optoelectronic networks. In *Proceedings of the 2023 International Conference on Neuromorphic Systems*, 2023. 33, 58, 60, 62, 64, 65

[109] Saber Moradi, Ning Qiao, Fabio Stefanini, and Giacomo Indiveri. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *IEEE transactions on biomedical circuits and systems*, 12(1):106–122, 2017. 34

BIBLIOGRAPHY

[110] Jeffrey M Shainline, Sonia M Buckley, Adam N McCaughan, Jeff Chiles, Amir Jafari-Salim, Richard P Mirin, and Sae Woo Nam. Circuit designs for superconducting optoelectronic loop neurons. *Journal of Applied Physics*, 124(15):152130, 2018. 34, 35

[111] Jeffrey M Shainline, Sonia M Buckley, Adam N McCaughan, Jeffrey T Chiles, Amir Jafari Salim, Manuel Castellanos-Beltran, Christine A Donnelly, Michael L Schneider, Richard P Mirin, and Sae Woo Nam. Superconducting optoelectronic loop neurons. *Journal of Applied Physics*, 126(4):044902, 2019. 34

[112] Shuangming Yang, Tian Gao, Jiang Wang, Bin Deng, Benjamin Lansdell, and Bernabe Linares-Barranco. Efficient spike-driven learning with dendritic event-based processing. *Frontiers in Neuroscience*, 15:601109, 2021. 34

[113] Jordan Guerguiev, Timothy P Lillicrap, and Blake A Richards. Towards deep learning with segregated dendrites. *Elife*, 6:e22901, 2017. 34

[114] Jeff Hawkins and Subutai Ahmad. Why neurons have thousands of synapses, a theory of sequence memory in neocortex. *Frontiers in neural circuits*, page 23, 2016. 34

[115] Panayiota Poirazi, Terrence Brannon, and Bartlett W Mel. Pyramidal neuron as two-layer neural network. *Neuron*, 37(6):989–999, 2003. 34

[116] Fabian A Mikulasch, Lucas Rudelt, Michael Wibral, and Viola Priesemann. Where is the error? hierarchical predictive coding through dendritic error computation. *Trends in Neurosciences*, 46(1):45–59, 2023. 34

[117] Fabian A Mikulasch, Lucas Rudelt, Michael Wibral, and Viola Priesemann. Dendritic predictive coding: A theory of cortical computation with spiking neurons. *arXiv preprint arXiv:2205.05303*, 2022. 34

[118] Fabian A Mikulasch, Lucas Rudelt, and Viola Priesemann. Local dendritic balance enables learning of efficient representations in networks of spiking neurons. *Proceedings of the National Academy of Sciences*, 118(50):e2021925118, 2021. 34

[119] Saeed Khan, Bryce A Primavera, Adam N McCaughan, Sonia M Buckley, Jeff Chiles, Alexander N Tait, Richard P Mirin, Sae Woo Nam, and Jeffrey M Shainline. Demonstration of single-photon synapses. In *CLEO: Science and Innovations*, pages SF3G–3. Optica Publishing Group, 2022. 35

[120] Saeed Khan, Bryce A Primavera, Jeff Chiles, Adam N McCaughan, Sonia M Buckley, Alexander N Tait, Adriana Lita, John Biesecker, Anna Fox, David Olaya, et al. Superconducting optoelectronic single-photon synapses. *Nature Electronics*, 5(10):650–659, 2022. 35, 96

[121] Sonia Buckley, Jeffrey Chiles, Adam N McCaughan, Galan Moody, Kevin L Silverman, Martin J Stevens, Richard P Mirin, Sae Woo Nam, and Jeffrey M Shainline. All-silicon light-emitting diodes waveguide-integrated with superconducting single-photon detectors. *Applied Physics Letters*, 111(14):141101, 2017. 35

[122] Jeffrey M Shainline, Bryce A Primavera, and Saeed Khan. Phenomenological model of superconducting optoelectronic loop neurons. *Physical Review Research*, 5(1):013164, 2023. 36, 47, 48

[123] Eugene M Izhikevich. *Dynamical systems in neuroscience*. MIT press, 2007. 39, 40

[124] Bryce A Primavera and Jeffrey M Shainline. An active dendritic tree can mitigate fan-in limitations in superconducting neurons. *Applied Physics Letters*, 119(24):242601, 2021. 43

[125] Ryan O'Loughlin, Bryce Primavera, and Jeffrey Shainline. Spatiotemporal dendritic processing in superconducting optoelectronic networks. In *2024 International Conference on Neuromorphic Systems (ICONS)*, pages 234–241. IEEE, 2024. 57

[126] Steve Furber and Petruț Bogdan. *SpiNNaker - A spiking neural network architecture*. Now Publishers, 2020. 58

[127] Mike Davies et al. Advancing neuromorphic computing with loihi: A survey of results and outlook. *Proceedings of the IEEE*, 109(5):911–934, 2021. 58

[128] C. Pehle et al. The brainscales-2 accelerated neuromorphic system with hybrid plasticity. *Frontiers*, 2022. 58

[129] Saber Moradi et al. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *IEEE Transactions on Biomedical Circuits and Systems*, 12(1):106–122, 2017. 58

[130] Filipp Akopyan et al. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10):1537–1557, 2015. 58

BIBLIOGRAPHY

[131] Jeffrey M. Shainline. Optoelectronic intelligence. *Applied Physics Letters*, 118(16), 2021. 58

[132] Saeed Khan et al. Superconducting optoelectronic single-photon synapses. *Nature Electronics*, 5(10):650–659, 2022. 58

[133] Jordan Guerguiev, Timothy P. Lillicrap, and Blake A. Richards. Towards deep learning with segregated dendrites. *eLife*, 6:e22901, 2017. 58

[134] Jeff Hawkins and Subutai Ahmad. Why neurons have thousands of synapses: A theory of sequence memory in neocortex. `https://arxiv.org/pdf/1511.00083`, 2015. 58, 61

[135] Eugene M. Izhikevich. *Dynamical Systems in Neuroscience*. MIT Press, 2007. 58

[136] Fabian A. Mikulasch, Lucas Rudelt, and Viola Priesemann. Local dendritic balance enables learning of efficient representations in networks of spiking neurons. *Proceedings of the National Academy of Sciences*, 118(50):e2021925118, 2021. 58

[137] Fabian A. Mikulasch et al. Dendritic predictive coding: A theory of cortical computation with spiking neurons. `https://arxiv.org/abs/2205.05303`, 2022. 58

[138] Fabian A. Mikulasch et al. Where is the error? hierarchical predictive coding through dendritic error computation. *Trends in Neurosciences*, 46(1):45–59, 2023. 58

[139] Panayiota Poirazi, Terrence Brannon, and Bartlett W. Mel. Pyramidal neuron as two-layer neural network. *Neuron*, 37(6):989–999, 2003. 58

[140] Shuangming Yang et al. Efficient spike-driven learning with dendritic event-based processing. *Frontiers in Neuroscience*, 15:601109, 2021. 58

[141] Johannes Leugering, Pascal Nieters, and Gordon Pipa. Dendritic plateau potentials can process spike sequences across multiple time-scales. *Frontiers in Cognition*, 2:1044216, 2023. 58

[142] Simone D'Agostino et al. Denram: Neuromorphic dendritic architecture with rram for efficient temporal processing with delays. *Nature Communications*, 15(1):3446, 2024. 58

[143] Mark Plagge, Suma G. Cardwell, and Frances S. Chance. Expanding spiking neural networks with dendrites for deep learning. In *Machine Learning with New Compute Paradigms*, 2024. 58

[144] Ary L. Goldberger et al. Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, 2000. 58, 72

[145] Yanping Chen et al. A general framework for never-ending learning from time series streams. *Data Mining and Knowledge Discovery*, 29:1622–1664, 2015. 58, 72

[146] Jeffrey M. Shainline. Fluxonic processing of photonic synapse events. *IEEE Journal of Selected Topics in Quantum Electronics*, 26(1):1–15, 2019. 58

[147] Ryan O'Loughlin. sim-soens. `https://github.com/ryangitsit/sim_soens`, 2024. 60

[148] Jeffrey M. Shainline, Bryce A. Primavera, and Saeed Khan. Phenomenological model of superconducting optoelectronic loop neurons. *Physical Review Research*, 5(1):013164, 2023. 60

[149] Bryce A. Primavera et al. Programmable superconducting optoelectronic single-photon synapses with integrated multi-state memory. *APL Machine Learning*, 2(2), 2024. 60, 62

[150] Albert Gidon et al. Dendritic action potentials and computation in human layer 2/3 cortical neurons. *Science*, 367(6473):83–87, 2020. 61

[151] Lou Beaulieu-Laroche et al. Enhanced dendritic compartmentalization in human cortical neurons. *Cell*, 175(3):643–651, 2018. 61

[152] Stefan Scholl. Classification of radio signals and hf transmission modes with deep learning. `https://arxiv.org/abs/1906.04459`, 2019. 77

[153] Bakhrom Oripov Bradley Hayes Adam McCaughan Sonia Buckley Ryan O'Loughlin, Nick Skuda. Next-generation neural interfaces. In *2025 International Conference on Neuromorphic Systems (ICONS)*. IEEE, 2025. Forthcoming, accepted for publication. 79

[154] Rodney J Douglas and Kevan AC Martin. Recurrent neuronal circuits in the neocortex. *Current biology*, 17(13):R496–R500, 2007. 80

[155] Xiao-Jing Wang. Synaptic reverberation underlying mnemonic persistent activity. *Trends in neurosciences*, 24(8):455–463, 2001. 80

[156] Henry Markram, Eilif Muller, Srikanth Ramaswamy, Michael W Reimann, Marwan Abdellah, Carlos Aguado Sanchez, Anastasia Ailamaki, Lidia Alonso-Nanclares, Nicolas Antille, Selim Arsever, et al. Reconstruction and simulation of neocortical microcircuitry. *Cell*, 163(2):456–492, 2015. 80

[157] Floris Takens. Lecture notes in mathematics. In *DA Rand and LS Young Springer, Berlin*, volume 898, page 366. 1981. 80

[158] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002. 80

[159] Herbert Jaeger. Adaptive nonlinear system identification with echo state networks. *Advances in neural information processing systems*, 15, 2002. 80, 89

[160] Xiaoran He, Ting Liu, Fatemeh Hadaeghi, and Herbert Jaeger. Reservoir transfer on analog neuromorphic hardware. In *2019 9th International IEEE/EMBS Conference on Neural Engineering (NER)*, pages 1234–1238. IEEE, 2019. 80

[161] Alejandra Patiño-Saucedo, Hugo Rostro-González, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. Liquid state machine on spinnaker for spatio-temporal classification tasks. *Frontiers in Neuroscience*, 16:819063, 2022. 80

[162] Rishabh Patel, Vivek Saraswat, and Umesh Ganguly. Liquid state machine on loihi: Memory metric for performance prediction. In *International Conference on Artificial Neural Networks*, pages 692–703. Springer Nature Switzerland, 2022. 80

[163] Aaron Voelker, Ilja Kajić, and Chris Eliasmith. Legendre memory units: Continuous-time representation in recurrent neural networks. *Advances in neural information processing systems*, 32, 2019. 80

[164] Rahul Gaurav, Terrence C Stewart, and Yiran C Yi. Spiking reservoir computing for temporal edge intelligence on loihi. In *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*, pages 526–530. IEEE, 2022. 80

[165] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021. 80, 84

[166] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023. 80

[167] Jeremy Appleyard, Tomas Kocisky, and Phil Blunsom. Optimizing performance of recurrent neural networks on gpus. *arXiv preprint arXiv:1604.01946*, 2016. 81

[168] Eriko Nurvitadhi, Jaewoong Sim, David Sheffield, Asit Mishra, Srivatsan Krishnan, and Debbie Marr. Accelerating recurrent neural networks in analytics servers: Comparison of fpga, cpu, gpu, and asic. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4. IEEE, 2016. 81

[169] Catherine D Schuman, Thomas E Potok, Robert M Patton, J Douglas Birdwell, Mark E Dean, Garrett S Rose, and James S Plank. A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint arXiv:1705.06963*, 2017. 81

[170] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7:13276, 2016. 82, 110

[171] Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24, 2017. 82

[172] Peter U Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience*, 9:99, 2015. 82

[173] Charlotte Frenkel and Giacomo Indiveri. Reckon: A 28nm sub-mm2 task-agnostic spiking recurrent neural network processor enabling on-chip learning over second-long timescales. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pages 1–3. IEEE, 2022. 82

[174] Uicheol Shin, Masatoshi Ishii, Atsuya Okazaki, Megumi Ito, Malte J Rasch, Wanki Kim, Akiyo Nomura, Wonseok Choi, Dooyong Koh, Kohji Hosokawa, et al. Pattern training, inference, and regeneration demonstration using on-chip trainable neuromorphic chips for spiking restricted boltzmann machine. *Advanced Intelligent Systems*, 4(8):2200034, 2022. 82

[175] Qingtian Zhang, Huaqiang Wu, Peng Yao, Wenqiang Zhang, Bin Gao, Ning Deng, and He Qian. Sign backpropagation: An on-chip learning algorithm for analog rram neuromorphic computing systems. *Neural Networks*, 108:217–223, 2018. 82

[176] ERW Van Doremaele, X Ji, J Rivnay, and Y Van De Burgt. A retrainable neuromorphic biosensor for on-chip learning and classification. *Nature Electronics*, 6(10):765–770, 2023. 82

[177] Adam N McCaughan et al. Multiplexed gradient descent: Fast online training of modern datasets on hardware neural networks without backpropagation. *APL Machine Learning*, 1(2), 2023. 82, 99

[178] James C Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE transactions on automatic control*, 37(3):332–341, 1992. 82, 83, 99, 100

[179] Matei-Ioan Stan and Oliver Rhodes. Learning long sequences in spiking neural networks. *Scientific Reports*, 14(1):21957, 2024. 96

[180] Amir Dembo and Thomas Kailath. Model-free distributed learning. *IEEE Transactions on Neural Networks*, 1(1):58–70, 1990. 99, 105

[181] Paul Züge, Christian Klos, and Raoul-Martin Memmesheimer. Weight versus node perturbation learning in temporally extended tasks: Weight perturbation often performs similarly or better. *Physical Review X*, 13(2):021006, 2023. 99, 102, 105

[182] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signsgd: Compressed optimisation for non-convex problems. In *International conference on machine learning*, pages 560–569. PMLR, 2018. 104

[183] Gert Cauwenberghs. A fast stochastic error-descent algorithm for supervised learning and optimization. *Advances in neural information processing systems*, 5, 1992. 105

[184] Sander Dalm, Marcel van Gerven, and Nasir Ahmad. Effective learning with node perturbation in multi-layer neural networks. *arXiv preprint arXiv:2310.00965*, 2023. 105

[185] Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 2007. 108

[186] Angela H Jiang, Daniel L-K Wong, Giulio Zhou, David G Andersen, Jeffrey Dean, Gregory R Ganger, Gauri Joshi, Michael Kaminksy, Michael Kozuch, Zachary C Lipton, et al. Accelerating deep learning by focusing on the biggest losers. *arXiv preprint arXiv:1910.00762*, 2019. 109

[187] Satrajit Chatterjee. Coherent gradients: An approach to understanding generalization in gradient descent-based optimization. *arXiv preprint arXiv:2002.10657*, 2020. 109, 110

[188] Satrajit Chatterjee and Piotr Zielinski. On the generalization mystery in deep learning. *arXiv preprint arXiv:2203.10036*, 2022. 109, 110

[189] Vardan Papyan, XY Han, and David L Donoho. Prevalence of neural collapse during the terminal phase of deep learning training. *Proceedings of the National Academy of Sciences*, 117(40):24652–24663, 2020. 109, 110

[190] Yanzhou Pan, Huawei Lin, Yide Ran, Jiamin Chen, Xiaodong Yu, Weijie Zhao, Denghui Zhang, and Zhaozhuo Xu. Alinfik: Learning to approximate linearized future influence kernel for scalable third-party llm data valuation. *arXiv preprint arXiv:2503.01052*, 2025. 109

[191] Guodong Zhang, James Martens, and Roger B Grosse. Fast convergence of natural gradient descent for over-parameterized neural networks. *Advances in Neural Information Processing Systems*, 32, 2019. 110

[192] Brian S Robinson, Isaac Western, Erik C Johnson, and Matthew J Roos. Optimizing generalized feedback paths for credit assignment. In *Proceedings of the International Conference on Neuromorphic Systems*, pages 135–140, 2025. 110

[193] Justin Lines, Eduardo D Martin, Paulo Kofuji, Juan Aguilar, and Alfonso Araque. Astrocytes modulate sensory-evoked neuronal network activity. *Nature communications*, 11(1):3689, 2020. 113

[194] Rune Nguyen Rasmussen, Antonis Asiminas, Eva Maria Meier Carlsen, Celia Kjaerby, and Nathan Anthony Smith. Astrocytes: integrators of arousal state and sensory context. *Trends in neurosciences*, 46(6):418–425, 2023. 113

[195] Andrej Karpathy. Nanogpt. `https://github.com/karpathy/nanoGPT`, 2023. 116

[196] William Shakespeare. The complete works of william shakespeare. Public Domain Text, 1623. Used as a dataset for next-token prediction in modern machine learning experiments. 116

[197] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. Jax: composable transformations of python+numpy programs, 2018. 117