ADAPTING NEURAL NETWORK LEARNING ALGORITHMS FOR NEUROMORPHIC IMPLEMENTATIONS

by

Jason M. Allred

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

School of Electrical and Computer Engineering

West Lafayette, Indiana

August 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Prof. Kaushik Roy, Chair
School of Electrical and Computer Engineering

Prof. Anand Raghunathan
School of Electrical and Computer Engineering

Prof. Vijay Raghunathan
School of Electrical and Computer Engineering

Prof. Okan Ersoy
School of Electrical and Computer Engineering

Approved by:
Prof. Dimitrios Peroulis

to Celisse

ACKNOWLEDGMENTS

I would like to thank my wife, Celisse, for her unwavering commitment to our family, for raising children far from home (during a pandemic!), and for going on this incredible adventure with me. She has worked so hard. This dissertation is as much hers as it is mine. I also thank my parents, in-laws, and siblings for their encouragement, support, visits, example, and love.

I place a special note here for my young children: I love you! You have spent your childhood here in Indiana so that I could pursue this degree. Thank you. I want you to know that while this was a big accomplishment for me, YOU are much more important to me. Even if I had failed here at Purdue, I would still have viewed the last several years as my greatest accomplishment–being your dad.

I am sincerely grateful to my advisor, Professor Kaushik Roy, not only for his consistent availability, resources, guidance, and encouragement, but also for his understanding, support, and accommodation for my family responsibilities and my desired career path. I thank my committee members, Professors Anand Raghunathan, Vijay Raghunathan, and Okan Ersoy for their guidance, feedback, and recommendations. I also gratefully acknowledge valuable discussions with Professor Jonathon Peterson and Dr. Gerard (Rod) Rinkus.

Additionally, my heartfelt thanks goes to Professor Sanghamitra Roy and Professor Koushik Chakraborty, my advisor and co-advisor from Utah State University during my master's degree who encouraged, enabled, and recommended me to continue my research and education. Similarly, I owe so much to Brother Ron Jones, my advisor at Brigham Young University–Idaho during my bachelor's studies, for being an exemplary teacher, a kind mentor, an enabling employer, and for being my role model.

I give credit to my collaborators during my doctoral studies, Professor Priyadarshini Roy, Dr. Gopalakrishan Srinivasan, Steven Spencer, Professor Shriram Ramanathan, Dr. Zhen Zhang, and Dr. Sandip Mondal. I also thank Nitin Rathi for help with his hybrid SNN conversion and training code, and I thank Wachirawit (Mint) Ponghiran and Timur Ibrayev for Python and GPU help and for being great lab mates. With that, I thank all other past and current members of our Nanoelectronics Research Laboratory (NRL) and, more generally, all those involved in the Center for Brain-Inspired Computing (C-BRIC). I gratefully recognize Nicole Piegza, the C-BRIC Research Operations Manager, and her predecessor Lynne Dahmen for their organizational and logistical support and assistance.

Lastly, I thank the members of the Wea Creek Ward (previously the Lafayette 3rd Ward) of the Lafayette Indiana Stake of The Church of Jesus Christ of Latter-day Saints for being a family away from home, for being my friends, and for being there for us when we needed your help and service. You have been the Christ-like neighbors that anyone could hope for.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Computing with Artificial Neural Networks (ANNs) is a branch of machine learning that has seen substantial growth over the last decade, significantly increasing the accuracy and capability of machine learning systems. ANNs are connected networks of computing elements inspired by the neuronal connectivity in the brain. Spiking Neural Networks (SNNs) are a type of ANN that operate with event-driven computation, inspired by the "spikes" or firing events of individual neurons in the brain. Neuromorphic computing—the implementation of neural networks in hardware—seeks to improve the energy efficiency of these machine learning systems either by computing directly with device physical primitives, by bypassing the software layer of logical implementations, or by operating with SNN event-driven computation. Such implementations may, however, have added restrictions, including weight-localized learning and hard-wired connections. Further obstacles, such as catastrophic forgetting, the lack of supervised error signals, and storage and energy constraints, are encountered when these systems need to perform autonomous online, real-time learning in an unknown, changing environment.

Adapting neural network learning algorithms for these constraints can help address these issues. Specifically, corrections to Spike Timing-Dependent Plasticity (STDP) can stabilize local, unsupervised learning; accounting for the statistical firing properties of spiking neurons may improve conversions from non-spiking to spiking networks; biologically-inspired dopaminergic and habituation adjustments to STDP can limit catastrophic forgetting; convolving temporally instead of spatially can provide for localized weight sharing with direct synaptic connections; and explicitly training for spiking sparsity can significantly reduce computational energy consumption.

1. INTRODUCTION

Engineering is all about creating effective tools, systems, or products, while also balancing competing objectives such as cost, time, and energy. These competing objectives open up many research avenues, helping us to explore not just *what* our creations can do, but also *how efficiently* they operate. As a part of the Center for Brain-Inspired Computing (C-BRIC), I have considered the modifications to neural network machine learning algorithms that must occur for the associated computations to be performed more efficiently, to be computed on more efficient hardware, and to be more efficient in their data usage over time.

1.1 Machine Learning with Neural Networks

Computing with neural networks is a branch of machine learning that has seen substantial growth over the last decade, significantly increasing the accuracy and capability of machine learning systems to do previously unattainable tasks, such as accurate image and voice recognition.

1.1.1 Artificial Neural Networks (ANNs)

An Artificial Neural Network (ANN), loosely based on neuronal connectivity in the brain, is a network of *neuron* computing elements with *soma* units that integrate incoming signals received from other neurons via weighted *synapses*. Non-linear neuron activation functions applied to the outgoing signal of each neuron in multi-layered networks enable them to approximate solutions to more complex problems. Deeper networks, however, can confront over-fitting difficulties [1], [2] because of the over-parameterization that enables this capability. In such cases, a network's ability to generalize can be improved by parametric reduction via "weight sharing" techniques [3] such as convolutions [4], [5].

Convolutional Neural Networks (CNNs) are ANNs that have one or more convolutional layers which comprise a set of *kernels* or filter weights that convolve over the values from the preceding layer, actualizing a feature map. At earlier layers, feature maps help the network identify small, local structure in the data. At later layers, the feature maps identify larger, more complex data patterns. Such feature maps involve a high degree of weight replication, as kernels are shared between the windows of convolution. This sharing of convolutional kernels is central to a later portion of this work and is discussed further in Chapter 5. Convolutional layers are often followed by pooling layers, which provide sub-sampling, further reducing overfitting while also improving spacial and noise invariance [6].

1.1.2 Spiking Neural Networks (SNNs)

For many years, the dominant neural networks in machine learning have been deep convolutional ANNs. Spiking Neural Networks (SNNs), a type of ANN that operate with event-driven computation, have been touted as the "third generation" of neural networks [7]. SNNs are inspired by the *spikes* or firing events of individual neurons in the brain. Each spiking neuron only fires when its membrane potential reaches a specified threshold, similar to biological models, creating an asynchronicity that adds a timing component to the network. This firing event then sends an all-or-nothing spike to the other neurons to which it is connected through unidirectional, weighted synapses. SNNs have been gaining traction due to their biological plausibility [8] and other potential benefits, discussed next.

1.1.3 Why *spiking*?

Neural networks with asynchronous, time-based signals have been shown to be computationally more powerful than threshold- or sigmoidal-gated neural network models by additionally encoding information in the temporal dimension [7]. The stochasticity of spiking events may also prove beneficial for system robustness.

In addition, SNNs can be more energy efficient by performing event-driven computation triggered by synaptic action potentials. The inherent computational sparsity and additive nature of integrating spikes versus synaptic multiplication on logically analog inputs is a further source of potential energy efficiency [9].

Further, spiking neurons can learn with local, unsupervised learning rules. This type of learning has been shown to occur with beyond-CMOS spintronic devices, resulting in power consumption equivalent to biological counterparts [10].

1.2 Online Neuromorphic Implementations

As Dennard scaling of traditional CMOS devices continues to break down in the nano-transistor regime, non-traditional computational models are gaining ground to expand the computing domain and decrease the energy footprint of VLSI circuits. Neuromorphic implementations of neural networks are one such area of exploration.

1.2.1 What is *neuromorphic*?

In a broad sense, "neuromorphic computing" encompasses all efforts of implementing neural networks in hardware. More specifically, however, in the literature there are three different ways to define the term *neuromorphic* [11]:

- computing directly with device physical primitives, sometimes on analog or time-based signals;
- bypassing the software layer of logical implementation with dedicated components; or
- operating with SNN event-driven computation.

Such neuromorphic designs are not just more biologically plausible—they may significantly improve computational energy efficiency and reduce inference latency–enabling real-time, low power machine learning on online systems. The goals and inherent properties of online neuromorphic systems, however, place certain restrictions on the learning algorithms that they can operate. We list eight such restrictions in the next two subsections.

1.2.2 Neuromorphic restrictions

In general, the above defining characteristics of neuromorphic computing imply that such hardware implementations have one or more of the following requirements:

1. local learning (synaptic weight changes occurring with only local information);
2. fixed size and configuration;

3. dedicated units and hardwired connections; and/or
4. asynchronous, spiking, or analog computation.

1.2.3 Online restrictions

There are also several goals associated with operating a neuromorphic system online. Ideally, an online system would have the following characteristics:

5. autonomous and real-time (no offline retraining);
6. unsupervised learning (able to learn without supervised error signals);
7. lifelong/dynamic learning (able to adapt to a changing environment while retaining memory); and
8. energy efficient (able to operate with a limited power supply).

Each of these characteristics also introduces restrictions on the neural network learning algorithms for such systems.

1.3 Adapting for Online Neuromorphic Systems

Traditional neural network learning algorithms generally assume offline learning with up-front access to all training data, together with their desired responses or labels. Additionally, training and inference are typically computed on von-Neumann machines, performing digital computations with generic computing components that load and store from shared memory. These traditional algorithms are, therefore, not ideally suited for online neuromorphic implementations.

This work presents several proposed adaptations to traditional neural network learning algorithms that each help address one or more of the previously-mentioned restrictions associated with online neuromorphic computing, extracted from my work in [12]–[18]. Specifically, Chapter 3 proposes two corrections to spiking learning algorithms: the first (Section 3.2) in regards to online local, unsupervised learning [12] (©2020 Allred and Roy)[1]; and the second

[1]↑Used with permission.

(Section 3.3) in regards to conversion from a non-spiking to a spiking network [13]. Chapter 4 presents proposed augmentations to online local, unsupervised learning that enable lifelong learning [14] (©2016 IEEE)[2], [12], [15] (©2018 IEEE)[3], [16]. Chapter 5 presents a proof-of-concept for convolutional neural networks that accomplish traditional weight sharing while still satisfying weight-localized computation on statically-connected synapses [17] (©2017 IEEE)[4]. And finally, Chapter 6 demonstrates a method for achieving significant computational energy efficiency improvements in SNNs by explicitly training for spiking sparsity [18]. Before presenting these works, I first provide the models they employ, detailed next in Chapter 2.

[2]↑Used with permission. In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Purdue's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.
[3]↑See footnote 2.
[4]↑See footnote 2.

2. SPIKING SYSTEMS AND MODELS

As spiking networks are central to this work, this chapter preemptively presents the basic SNN models and systems used in the subsequent chapters. Table 2.1 lays out the different model and system characteristics of each proposed method, which are subsequently detailed. The three simulation platforms employed were Brian [19] based on the code in [20], Matlab using a built-from-scratch event-driven simulation, and PyTorch based on the code in [21].

2.1 Synapse Models

For the methods that use the Brian spiking simulator, the synaptic connections between neurons are modeled as inducing alpha responses, with incoming spikes inducing an exponentially decaying current that potentiates an exponentially decaying membrane potential (discussed next). The magnitude of the initial spike in current is weighted by the synapse value. Alpha responses are more difficult to model in an event-driven system because they create an non-instantaneous potentiation.

For the other methods, synapses are modeled as a delta spike from its pre-synaptic neuron, scaled by the synaptic weight, and then added to the membrane potential of its post-synaptic neuron, creating a exponential kernel response. Later in this work, we represent the weight of the synapse connecting input i to neuron j as w_{ij} and the vector of all inputs to neuron j as $\vec{w}_{\mathrm{j}}$.

Table 2.1. Model and system characteristics employed by each proposed spiking method.

Method (Section)	Platform	Synapse	Homeo.	Computation	Lateral
Stabilized STDP (3.2)	Matlab	delta	yes	event-driven	direct
L^4-Norm Adjustment (3.3)	PyTorch	delta	N/A	time steps	N/A
Forced Firing (4.3)	Brian	alpha	no	time steps	separate
Dopaminergic (4.4)	Matlab	delta	no	event-driven	direct
ASP (4.7)	Brian	alpha	yes	time steps	separate
Explicit Sparsity (6)	PyTorch	delta	N/A	time steps	N/A

2.2 Spiking Neuron Models

For all methods, we use the common Leaky-Integrate-and-Fire (LIF) neuron model, in which a neuron's membrane potential v_{mem} undergoes a continuous decay according to the differential equation in (2.1), where τ_{mem} is the membrane decay constant and v_{rest} is the resting potential. The membrane potential is also potentiated or depressed by incoming excitatory or inhibitory signals, respectively. If the membrane potential reaches or surpasses the neuron's firing threshold v_{th} then the neuron fires, producing an output spike and resetting its potential to v_{reset}. This is known as a hard reset, which is used throughout unless otherwise specified. Section 3.3 and Chapter 6 employ a soft reset, in which the membrane is simply reduced by the threshold value, retaining any residual potential over the threshold. After each firing event, there may be a refractory period during which the neuron is incapable of firing even if additional input spikes are received. Without loss of generality, we set v_{rest} to zero as a reference voltage for the mathematical analysis in the next chapter. Except for the Brian simulations, for model and evaluation simplicity we also set v_{reset} to zero and have no refractory periods. Figure 2.1 shows an example LIF neuron responding to incoming spikes via delta synapses.

$$\dot{v}_{mem} = \frac{-(v_{mem} - v_{rest})}{\tau_{mem}} \tag{2.1}$$

2.3 Rate-encoded Spike Train Inputs

Input samples are encoded as Poisson spike trains, following the mathematical model of a Poisson point process (PPP), where the spike rate λ_{i} of an input neuron is proportional to the pixel intensity of input i. Thus, the number of spikes in a given time window follows the distribution of a Poisson random variable with an expectation proportional to the input value. For perception tasks on static images, there is no temporal information in a single sample, and thus rate encoding is one of the most common encoding methods for SNN image perception implementations as it maintains statistical independence between individual input spikes,

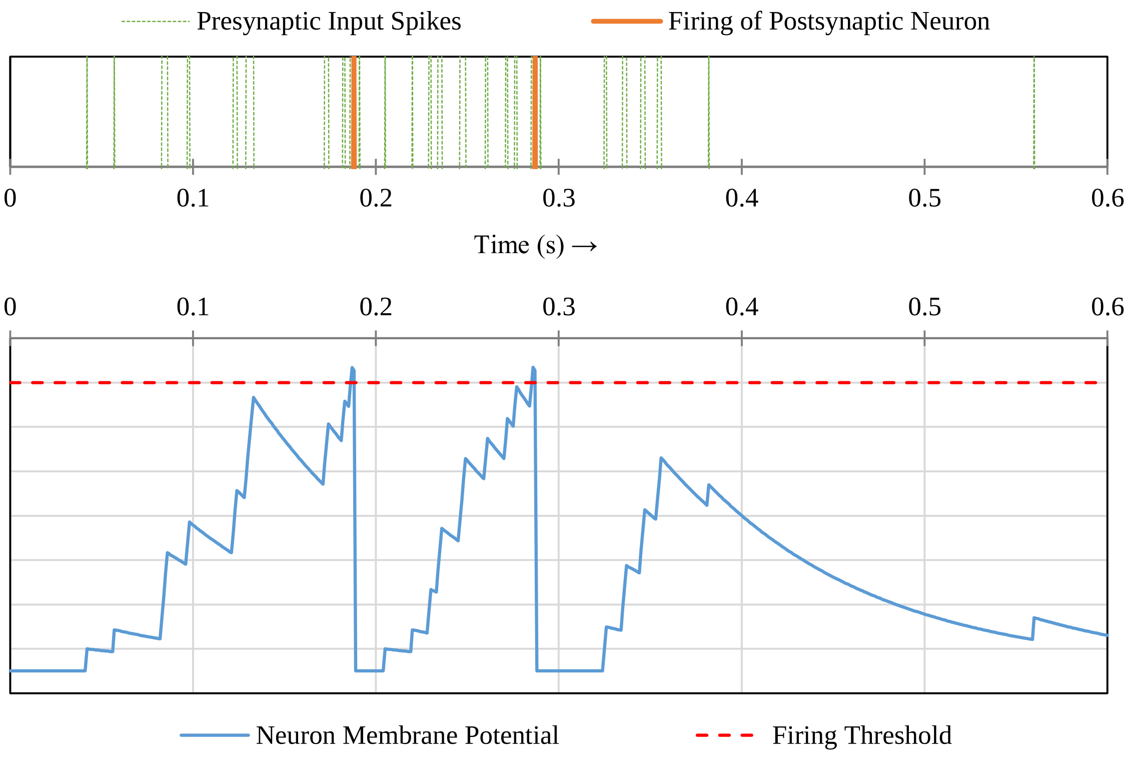


Figure 2.1. Leaky-Integrate-and-Fire model for spiking neurons in an SNN. (©2016 IEEE [14])

which is useful for the computationally less expensive one-sided STDP curve, discussed later in Section 2.4.2.[1]

Each spike is modeled as a time-shifted delta function. The precise time of the k^{th} most recent spike from input i is represented as $t_{\mathrm{i}k}$. Being a PPP, the timing between two sequential spikes on a given input channel are drawn from an exponential random distribution, also with rate λ_{i}. The time passed since the k^{th} most recent spike from i at time t is represented as $t_{|\mathrm{i}k|} = t - t_{\mathrm{i}k}$ and follows the distribution of a gamma random variable $T_{|\mathrm{i}k|} \sim gamma(\alpha = k; \beta = \lambda_{\mathrm{i}})$. The vector of all input rates for each dimension of the given sample is represented as $\vec{\lambda}$.

2.4 Unsupervised Learning in SNNs

Unsupervised learning is performed in four of the spiking works in this proposal (the first four in Table 2.1). The clustering tasks in these works are based on the architecture and learning process by Diehl and Cook [22].

2.4.1 Architecture for competitive unsupervised learning

Competitive learning is enabled by an architecture based on [22], which consists of two layers: an excitatory layer and an inhibitory layer. Each of the excitatory neurons are fully connected to the input and represents competing reference vector neurons during the clustering task.

Lateral inhibition

There is a one-to-one connection from each excitatory neuron to its respective inhibitory neuron in the inhibitory layer. The inhibitory neurons send inhibiting signals to every neuron except the neuron that triggered it. This layer provides competition by limiting the firing

[1]↑Other input encodings that use time-encoding such as rank-order may provide energy efficiency improvements but at the moment provide no obvious additional benefit to the addressed problems and are thus beyond the scope of this work.

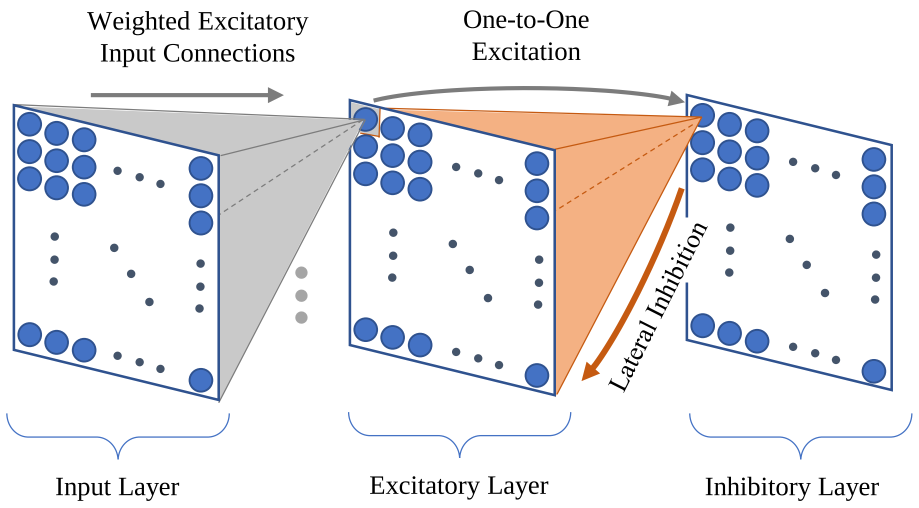


Figure 2.2. Competitive architecture with lateral inhibition for unsupervised learning.

of the other reference vector neurons, causing each to learn different input patterns. Figure 2.2 illustrates this architectural setup.

For the Stabilized STDP and the Dopaminergic methods, lateral inhibition was enacted instead with direct inhibitory connections from each reference vector neuron rather than a separate layer of inhibitory neurons, effecting the same behaviour.

Homeostasis

With STDP (discussed next), relevant synaptic weights are strengthened when the post-synaptic neuron fires. A larger weight, in turn, enables the neuron to fire more easily. This setup can create a positive feedback loop during training, potentially causing a single neuron to dominate. This problem is often solved with homeostasis, distributing firing activity between neurons.

One method of homeostasis based on [23] is implemented with adaptive thresholding—dynamically increasing a neuron's firing threshold each time it fires, with a decay over time. The temporarily larger threshold prevents a single neuron from firing too frequently. For two of the proposed methods, Forced Firing and Dopaminergic Learning, this form of homeostasis is not employed because it is not effective on the targeted lifelong learning tasks.

2.4.2 Spike-Timing Dependant Plasticity (STDP)

Some components of biological neural learning can be modeled with Spike Timing Dependent Plasticity (STDP) [24]. In STDP the weights of synaptic connections between neurons are strengthened or weakened based on the timing of neuron spikes.

Traditional STDP

The STDP learning rule is a correlation-based Hebbian learning rule that achieves unsupervised clustering by strengthening causal connections and weakening anti-causal connections. This is done by potentiating the weight when the post-synaptic neuron fires shortly after the pre-synaptic neuron has fired, with the magnitude of the weight change exponentially decaying with the increase of the time difference. Thus, the more that the given pre-synaptic neuron contributed to causing the post-synaptic neuron to fire, the more easily it will be able to do so in the future.

On the other hand, depression occurs if the order is reversed and the post-synaptic neuron fires shortly *after* the pre-synaptic neuron has fired.

One-sided STDP

As the input information in the systems in this work is encoded only in the spike rate, we can instead employ the computationally less-expensive one-sided version of STDP, evaluated only at the post-synaptic firing event:

$$\Delta w = \alpha(pre - offset) \tag{2.2}$$

where α is the learning rate, pre is a trace of pre-synaptic firing events, and $offset$ is the value to which the pre-synaptic traces are compared, determining potentiation or depression. When the post-synaptic neuron fires shortly after the pre-synaptic neuron, the synaptic

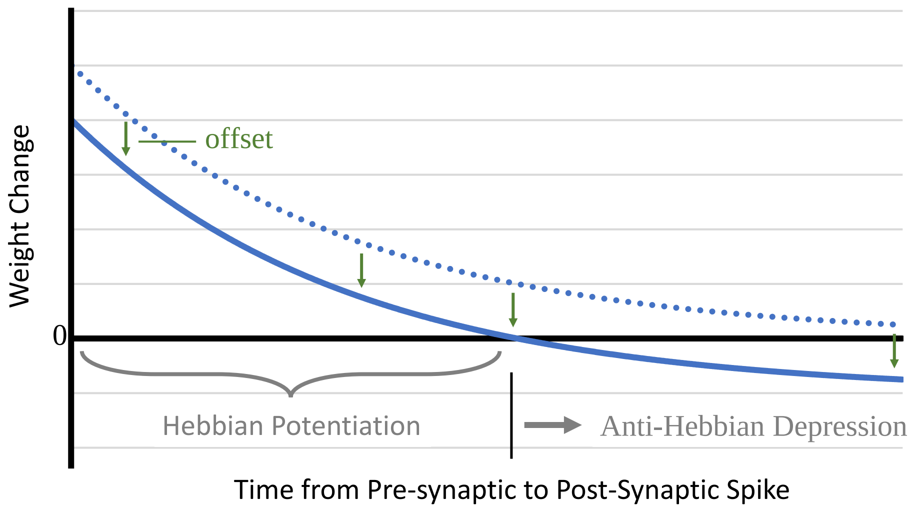


Figure 2.3. One-sided STDP curve.

weight is strengthened, with shorter time intervals resulting in an exponentially larger weight change. As opposed to the two-sided rule, depression occurs when this time interval is larger, as dictated by the $offset$.

Correlated potentiations in the direction of $p\vec{r}e$ therefore provide Hebbian learning by angularly migrating $\vec{w}$ toward the angle of the input vector $\vec{\lambda}$. Anti-Hebbian depression reduces weights from uncorrelated inputs and is provided by subtracting the $offset$ term for one-sided STDP rather than performing additional weight processing at pre-synaptic firing events. Figure 2.3 illustrates this learning rule.

2.4.3 Training and testing process

With unsupervised learning algorithms, it is important to carefully design the training and testing processes to avoid having the training labels influence model parameters while still considering how network predictions are to be identified. These processes are described here.

Training:

Samples are presented one-by-one to the network. For the current sample, input neurons fire at the sample rate until the system registers at least five output spikes, as followed by Diehl and Cook, which is generally enough to confidently identify the input in view of the stochasticity in the SNN. For the baseline, if a given sample does not produce enough output spike, the input spike rates are incrementally scaled up until the requirement is met. After five output spikes are registered, all membrane potentials are allowed to reset to avoid one sample interfering with the next, and then the next sample is presented. This training process is modified by the methods in Chapter 4 to represent sequentially presented tasks in a lifelong learning scenario (see Section 4.1.4).

Label assignment and testing:

As training is performed entirely without supervision, the final network outputs must be assigned class labels for evaluation. While the network is frozen, label assignment is done by inference on the training set, followed by evaluation on the testing set. The MNIST dataset is already highly clustered in its input space, and therefore a supervised linear classifier is already capable of competitive accuracy. Because of this, no final linear readout classification layer is added to avoid the label assignment process acting as a traditional supervised linear classifier. Instead, following the unsupervised evaluation method of Diehl and Cook, each trained neuron is *directly* assigned a class label and no linear combination of these neuron outputs is performed. Rather, the class decision is winner-take-all, choosing the class of the neuron that spiked the most for that sample.

3. LEARNING RULE AND MODEL CORRECTIONS FOR SPIKING NEURAL NETWORKS

Spiking networks differ from non-spiking feed-forward networks in several ways: (1) they have a built-in temporal component internal to each neuron creating a recurrent dependency over time; (2) there is some form of temporal encoding for the input, output and internal data signals; (3) they can operate with event-driven computation; (4) their non-linear activation functions are generally non-differentiable; and (5) the outputs of individual neurons are non-continuous, often being represented as binary–active (spike) or inactive (no spike).

Yet, much of the initial exploration of SNNs has followed the same paths as non-spiking networks, often ignoring some or all of these differences. The neuron and synapse models, input encoding, and learning rules must be considered jointly to more effectively and efficiently transfer the success of non-spiking network to SNNs. This chapter first examines the statistical membrane potential distribution of spiking neurons (Section 3.1) before using that information to correct local, unsupervised SNN learning (Section 3.2) and perform more accurate transferring of trained non-spiking models to the spiking domain (Section 3.3).

3.1 Pre-Firing Membrane Potential Distribution

To estimate the relative firing distributions of competing LIF neurons, it is useful to understand the distribution of their pre-firing membrane potentials. These derivations and discussions are extracted from and expanded on my work in [12] (©2020 Allred and Roy)[1].

3.1.1 Expected value of pre-firing membrane potential

Let $V_\mathrm{j}(t)$ be the random variable representing the potential of neuron j at time t. First, we consider the effect of the most recent incoming spike k which is received from input i at time $t_{\mathrm{i}k}$. If at the time immediately before the spike was received, $t_{\mathrm{i}k}^{-}$, the voltage of neuron

[1]↑Used here with permission. The material in this section is an expansion of [12], Section 2.3.4 therein. The bulk of my work in [12] is included later in Chapter 4, specifically in Sections 4.4, 4.5, and 4.6. The expanded discussion on these derivations are to be included in [13], manuscript in preparation, which contains my work in Section 3.3. I gratefully acknowledge assistance from Professor Jonathon Peterson of Purdue University regarding these derivations.

j over the resting potential v_{rest} is given by $v_{\mathrm{j}}(t_{\mathrm{i}k}^{-})$, then, assuming neuron j has not since fired, the potential of neuron j at time t is given by:

$$v_{\mathrm{j}}(t) = \Big(v_{\mathrm{j}}(t_{\mathrm{i}k}^{-}) + w_{\mathrm{ij}}u(t - t_{\mathrm{ik}})\Big)\mathrm{e}^{-(t-t_{\mathrm{ik}})/\tau_{mem}} + v_{rest} \tag{3.1}$$

where $u(t)$ is the unit step function. If we assume that the input spike has already occurred, i.e. $t > t_{\mathrm{i}k}$, then the unit step function may be removed. Provided that the decay rate τ_{mem} remains constant, we see that the value of the potential immediately preceding the spike, $v_{\mathrm{j}}(t_{\mathrm{i}k}^{-})$, and the weighted potential increase induced by the spike, w_{ij}, may be linearly separated. Without loss of generality, this linear separability may be extended to the residual potential increases induced by all spikes received by a neuron since its last firing event and refractory period, given by:

$$v_{\mathrm{j}}(t) = v_{rest} + \sum_{\mathrm{i}}\sum_{k} w_{\mathrm{ij}}\mathrm{e}^{-(t-t_{\mathrm{ik}})/\tau_{mem}} \tag{3.2}$$

As discussed in the text, we assume that $v_{reset} = v_{rest}$. In cases where this does not hold, the reset voltage is also linearly separable, and its residual effect may simply be added as $+(v_{reset} - v_{rest})\mathrm{e}^{-(t-t_{last\_ref})/\tau_{mem}}$ where t_{last_ref} is the time since the last refractory period.

Since $t_{|\mathrm{i}k|} = t - t_{\mathrm{i}k}$ follows the distribution of gamma random variable $T_{|\mathrm{i}k|} \sim gamma(\alpha = k; \beta = \lambda_{\mathrm{i}})$, let

$$V_{|\mathrm{i}k|} = u(T_{|\mathrm{i}k|}) = \mathrm{e}^{-T_{|\mathrm{i}k|}/\tau_{mem}} \tag{3.3}$$

be the random variable representing the unweighted portion of the potential increase from the k^{th} most recent spike from i. This allows us to rewrite (3.2) as a random variable:

$$V_{\mathrm{j}} = v_{rest} + \sum_{\mathrm{i}}\sum_{k} w_{\mathrm{ij}}V_{|\mathrm{i}k|} \tag{3.4}$$

Because $T_{|\mathrm{i}k|}$ is a gamma random variable, we know its CDF is

$$F_{T_{|\mathrm{i}k|}}(t) = \frac{\gamma(k, \lambda_{\mathrm{i}}t)}{\Gamma(k)} \tag{3.5}$$

where $\Gamma(s)$ and $\gamma(s, x)$ are the gamma function and the lower incomplete gamma function, respectively.

We can use (3.5) and invert (3.3) as (3.6) to solve for the transformation from $T_{|\mathrm{i}k|}$ to $V_{|\mathrm{i}k|}$ in (3.7).

$$t_1 = u^{-1}(v) = -\tau_{mem} \ln v \tag{3.6}$$

$$F_{V_{|\mathrm{i}k|}}(v) = 1 - F_{T_{|\mathrm{i}k|}}(t_1) = \frac{\Gamma(k, -\lambda_{\mathrm{i}}\tau_{mem} \ln v)}{\Gamma(k)} \tag{3.7}$$

where $\Gamma(s, x)$ is the upper incomplete gamma function.

We take the derivative of the CDF in (3.7) to give us the following PDF for $V_{|\mathrm{i}k|}$:

$$f_{V_{|\mathrm{i}k|}}(v) = \frac{(\lambda_{\mathrm{i}}\tau_{mem})^k v^{\lambda_{\mathrm{i}}\tau_{mem}-1}(-\ln v)^{k-1}}{(k-1)!} \tag{3.8}$$

We use the pdf of $V_{|\mathrm{i}k|}$ to calculate its expected value in (3.9), and since expectation is a linear operator, we solve for the expectation of V_{j} in (3.10) from (3.4) and (3.9).

$$E(V_{|\mathrm{i}k|}) = \int_0^1 v f_{V_{|\mathrm{i}k|}}(v) dv = \frac{(\lambda_{\mathrm{i}}\tau_{mem})^k}{(1 + \lambda_{\mathrm{i}}\tau_{mem})^k} \tag{3.9}$$

$$E(V_{\mathrm{j}}) = v_{rest} + \sum_{\mathrm{i}} \sum_{k} w_{\mathrm{ij}} \frac{(\lambda_{\mathrm{i}}\tau_{mem})^k}{(1 + \lambda_{\mathrm{i}}\tau_{mem})^k} \tag{3.10}$$

If we take the sum of spikes to infinity to get the steady state, which is a reasonable approximation since only the most recent spikes have a significant impact on the potential, then the inner sum converges to:

$$\begin{aligned} E(V_{\mathrm{j}}) &\approx v_{rest} + \sum_{\mathrm{i}} w_{\mathrm{ij}}\lambda_{\mathrm{i}}\tau_{mem} \\ &\approx v_{rest} + \tau_{mem}(\vec{w_{\mathrm{j}}} \cdot \vec{\lambda}) \end{aligned} \tag{3.11}$$

which includes a simple scaled dot product of neuron j's weight vector w_{j} and the input rate vector λ, as you would find in a non-spiking neuron. The appropriateness of this approximation is strengthened by the fact that in paper, the equation is used not to determine the precise spiking rate of an individual neuron but rather to compare relative spiking rates between competing neurons.

3.1.2 Function over a stochastic process

The entire input spike train may also be treated as a stochastic process. Assuming a firing event has yet to occur, the effect of a Poisson spike train on a neuron's membrane potential with exponential leakage may be viewed as a shot-noise process [25]. This allows for not only a more elegant derivation of the mean, but also an extension of that derivation to any moment of the distribution.

A Poisson spike train from input i is the summation of many spikes represented as delta functions:

$$N_{\mathrm{i}} = \sum_k \delta_{T_{|\mathrm{i}k|}} \tag{3.12}$$

This stochastic process produces the following pre-firing membrane potential induced on neuron j by the spike train from input i:

$$V_{\mathrm{ij}}(t) = \int f_{\mathrm{ij}}(t)N(dt) = \sum_k f_{\mathrm{ij}}(t - T_k), \tag{3.13}$$

where $f_{\mathrm{ij}}(t) = w_{\mathrm{ij}}\mathrm{e}^{-t/\tau_{mem}}$. The Laplace transform of this shot-noise process is:

$$\mathcal{L}(\theta) = E[\mathrm{e}^{-\theta V_{\mathrm{ij}}(t)}] = \mathrm{e}^{g(\theta)} \tag{3.14}$$

where $g(\theta) = \lambda_{\mathrm{i}} \int_0^t (\mathrm{e}^{-\theta f_{\mathrm{ij}}(v)} - 1)dv$.

First moment

The 1st moment, which is the mean pre-firing potential caused by input channel i, is given by:

$$\begin{aligned}
E[V_{\mathrm{ij}}(t)] &= -\left[\frac{d\mathcal{L}(\theta)}{d\theta}\right]_{\theta=0} = -\left[\frac{d\mathrm{e}^{g(\theta)}}{d\theta}\right]_{\theta=0} \\
&= -\left[\mathrm{e}^{g(\theta)}\right]_{\theta=0}\left[\frac{dg(\theta)}{d\theta}\right]_{\theta=0} \\
&= -\lambda_{\mathrm{i}}\left[\int_0^t(-f_{\mathrm{ij}}(v)\mathrm{e}^{-\theta f_{\mathrm{ij}}(v)})dv\right]_{\theta=0} \\
&= \lambda_{\mathrm{i}}\int_0^t f_{\mathrm{ij}}(v)dv = \lambda_{\mathrm{i}} w_{\mathrm{ij}}\tau_{mem}(1-\mathrm{e}^{-t/\tau_{mem}}) \qquad (3.15)
\end{aligned}$$

For all inputs, represented as the rate vector $\vec{\lambda}$, the mean combined pre-firing potential of neuron j is:

$$\begin{aligned}
E[V_{\mathrm{j}}(t)] &= \tau_{mem}\sum_{\mathrm{i}}\lambda_{\mathrm{i}} w_{\mathrm{ij}}(1-\mathrm{e}^{-t/\tau_{mem}}) \\
&= \tau_{mem}(\vec{w}_{\mathrm{j}}\cdot\vec{\lambda})(1-\mathrm{e}^{-t/\tau_{mem}}) \qquad (3.16)
\end{aligned}$$

In steady-state this converges to: $\tau_{mem}(\vec{w}_{\mathrm{j}}\cdot\vec{\lambda})$, which is important for discussions later in Sections 3.2 and 4.5.3.

Second moment and variance

Continuing to the second moment, we can calculate the variance of the pre-firing membrane potential that is induced on neuron j by incoming spikes received from input i:

$$\begin{aligned}
Var(V_{\mathrm{ij}}(t)) &= E[V_{\mathrm{ij}}(t)^2] - E[V_{\mathrm{ij}}(t)]^2 \\
&= \left[\frac{d^2\mathcal{L}(\theta)}{d\theta^2}\right]_{\theta=0} - E[V_{\mathrm{ij}}(t)]^2 \\
&= [\mathrm{e}^{g(\theta)}(g(\theta)^2+g(\theta))]_{\theta=0} - E[V_{\mathrm{ij}}(t)]^2 \\
&= E[V_{\mathrm{ij}}(t)]^2 + \lambda_{\mathrm{i}}\int_0^t f_{\mathrm{ij}}(v)^2dv - E[V_{\mathrm{ij}}(t)]^2 \\
&= \frac{1}{2}\lambda_{\mathrm{i}}\tau_{mem}w_{\mathrm{ij}}^2(1-\mathrm{e}^{-2t/\tau_{mem}}) \qquad (3.17)
\end{aligned}$$

The combined variance of the potential induced by all inputs is:

$$\begin{aligned} Var(V_{\mathrm{j}}(t)) &= \frac{1}{2}\tau_{mem}\sum_{\mathrm{i}}\lambda_{\mathrm{i}}w_{\mathrm{ij}}^2(1-\mathrm{e}^{-2t/\tau_{mem}}) \\ &= \frac{1}{2}\tau_{mem}(\vec{\lambda}\cdot\vec{w}_{\mathrm{j}}^{\circ 2})(1-\mathrm{e}^{-2t/\tau_{mem}}) \end{aligned} \tag{3.18}$$

where $\vec{w}_{\mathrm{j}}^{\circ 2}$ represents the *Hadamard square*, or element-wise square, of the weight vector. This equation is important for discussions later in Section 3.3.

Approximate distribution

The summation of the exponential decay of values drawn from a uniform distribution may be approximated with a lognormal distribution. Because the mean and variance can be calculated given the input rate vector, the neuron weight vector, and its membrane decay constant, we can use these values to generate the estimated PDF of a lognormal distribution.

And indeed, simulations of many spike trains with varying input rate vectors being applied to an LIF neuron with varying weight vectors produces distributions that match very well to lognormal distributions calculated this way.

3.2 Stabilizing One-Sided STDP

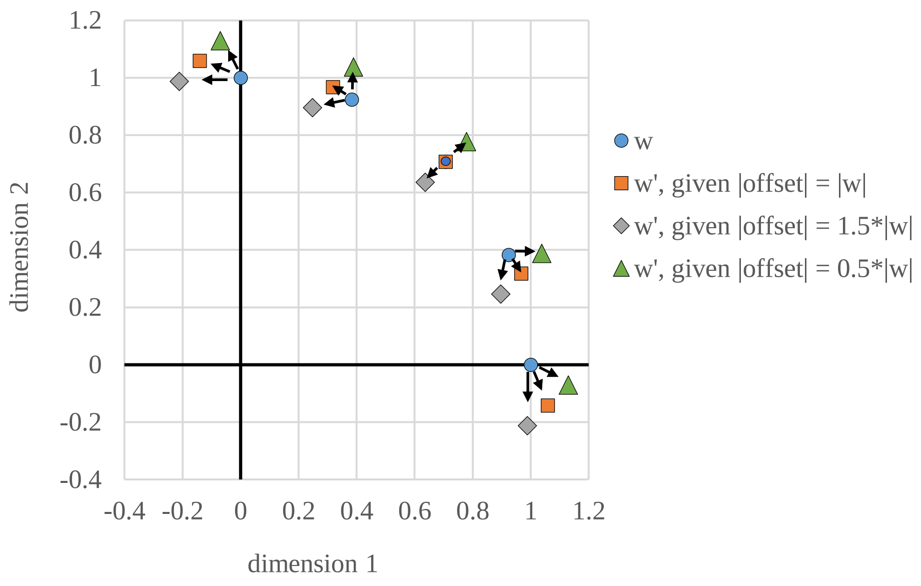
This section presents a proposed adjustment to the one-sided STDP learning rule that stabilizes the weight update equation for unsupervised clustering tasks–extracted from and expanded upon a portion of my work in [12] (©2020 Allred and Roy)[2]. In the baseline one-sided STDP learning rule (see Section 2.4.2), the pre trace follows a similar distribution as the membrane potential (see Section 3.1), only with a different time constant and without being weighted by the synapse, and so its expected value is also proportional to the input spike rate (e.g. $E[pre_{\mathrm{i}}] = \lambda_{\mathrm{i}}\tau_{pre}$). This is how the weight vector migrates toward the input vector. The $offset$ term is a constant value identical across all dimensions and can be thought of as a scaled ones vector, applying uniform anti-Hebbian depression by reducing the weights from uncorrelated inputs. Such uniform depression does not, however, create a weight change in exactly the direction desired (see Figure 3.1) and causes instability. Figure 3.1a shows that with a static offset in each dimension, even scaled to the appropriate magnitude, the weight vectors do not stabilize on the target and instead migrate toward the axes, creating binarized weights when capped at zero.

3.2.1 Tying the offset to the weight

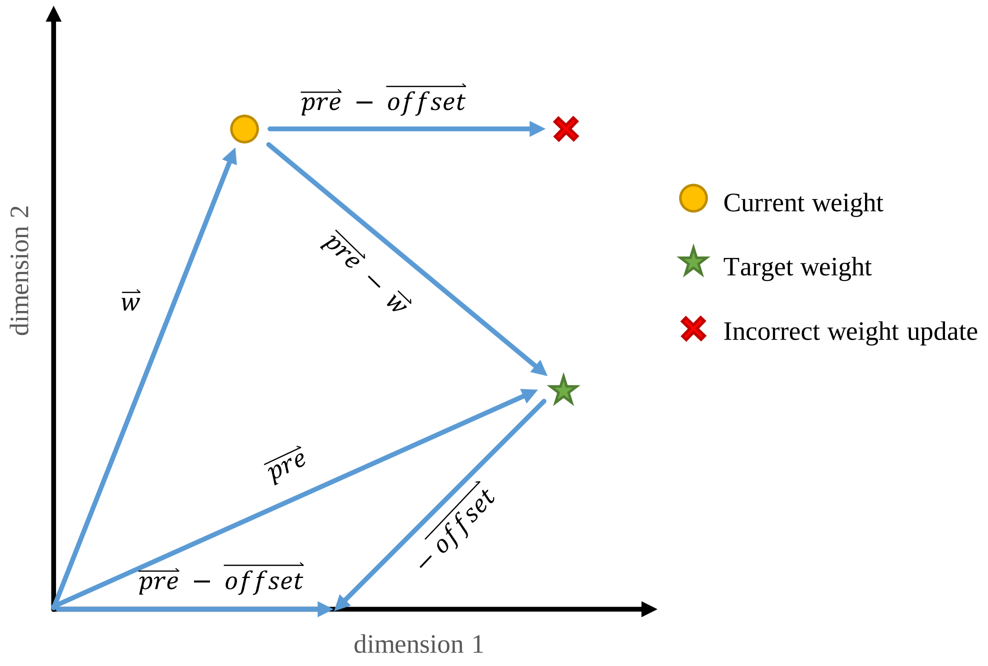
We provide the required stability to this STDP learning rule by correcting the direction of the weight change. Rather than a constant offset, we dynamically tie $offset$ to the current weight value, which is an adaptation based on Oja's rule [26]. The goal is to migrate $\vec{w}$ toward the $p\vec{r}e$ trace, which is proportional to the input $\vec{\lambda}$. Figure 3.1b shows that the $of\vec{f}set$ vector that is subtracted from $p\vec{r}e$ must be dynamically tied in each dimension to $\vec{w}$, rather than being the same in every dimension. To place pre and the weight on the same scale, we scale the pre-synaptic trace by the inverse of its decay rate τ_{pre}, changing (2.2) to:

$$\Delta w = \alpha\left(\frac{pre}{\tau_{pre}} - w\right) \tag{3.19}$$

[2]↑Used here with permission. The material in this section is an expansion of [12], Section 2.4.2.2 therein. The bulk of my work in [12] is included later in Chapter 4, specifically in Sections 4.4, 4.5, and 4.6.



(a) Weight change results for various starting positions where the target vector is equal to the current weight vector, which would ideally result in no weight change.



$\overrightarrow{pre} - \overrightarrow{offset}$
$\vec{w}$
$\overrightarrow{pre} - \vec{w}$
$\overrightarrow{pre}$
$-\overrightarrow{offset}$
$\overrightarrow{pre} - \overrightarrow{offset}$
dimension 2
dimension 1
Current weight
Target weight
Incorrect weight update

(b) Example vectors showing how a static offset does not result in a correct weight change, but should instead by tied to the current weight.

Figure 3.1. Instability of one-sided STDP. (©2020 Allred and Roy [12])

3.2.2 Stabilization results

Using setups with differing homeostatic hyperparameters, this stabilized version STDP learning rule was compared to the traditional which uses a static $offset$. For each, SNNs of various sizes performed unsupervised learning on the MNIST [27] dataset. Figure 3.2 shows that the correction consistently provides an approximate 1% accuracy improvement.

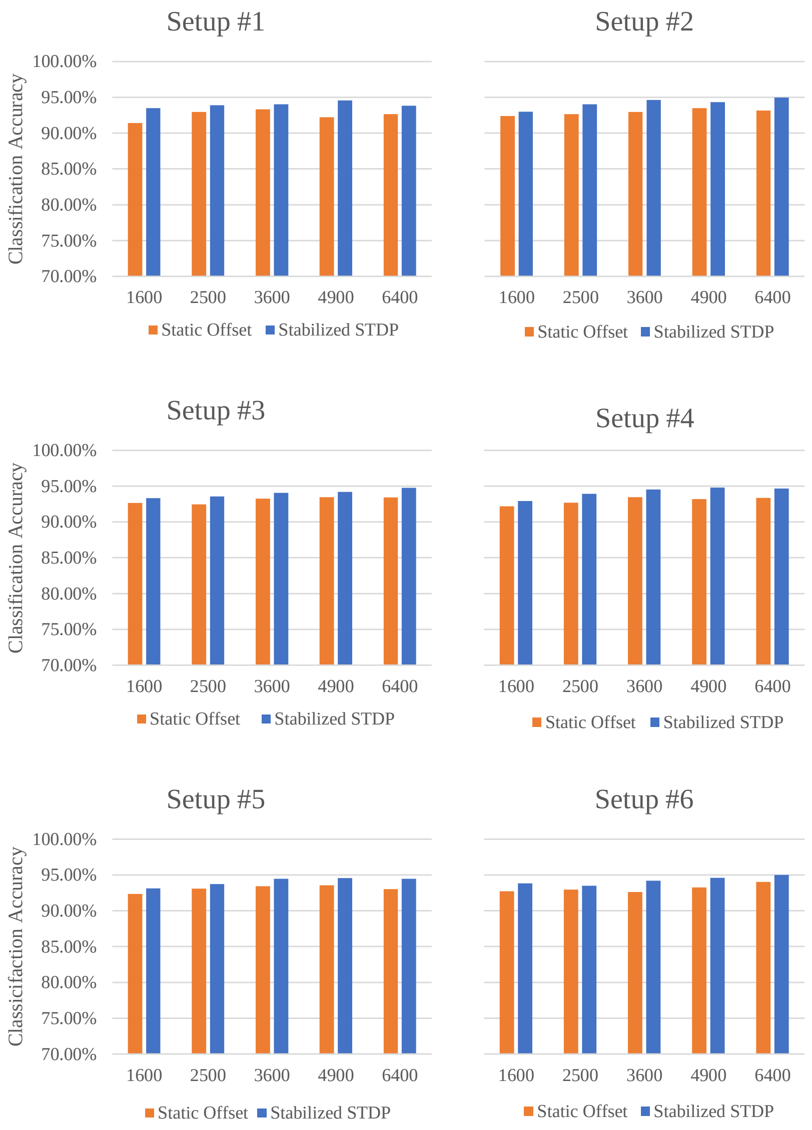


Figure 3.2. Unsupervised classification accuracy on MNIST using one-sided STDP with a static offset compared to stabilized on-sided STDP.

3.3 L^4-Norm Weight Adjustments

An efficient way of training spiking neural networks is by training a topologically homogeneous non-spiking network and then converting the weights over to a spiking network, replacing ReLUs with integrate-and-fire neurons and balancing the weights with the thresholds [28], [29]. One issue with this method is that SNNs operate in the temporal domain, and this training process completely ignores any temporal variance. By accounting for the temporal variance, the conversion process can be improved [13][3].

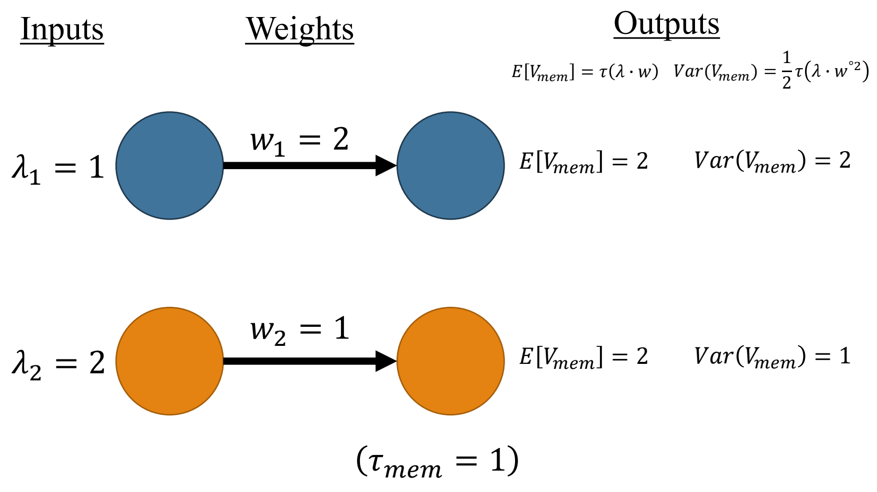
3.3.1 Why account for membrane potential variance?

Although the mean pre-firing membrane potential of an LIF neuron is proportional to the dot product of the input vector and the weight vector, we cannot assume that a neuron that has a larger mean pre-firing membrane potential will have a higher spiking probability. To illustrate this claim, consider two membrane potential distributions which have equivalent means but different variances (see Section 3.1.2), for example two neurons each with a single input and $\tau_{mem} = 1$: the first with input rate $\lambda_1 = 1$ and synaptic weight $w_1 = 2$, and the second with input rate $\lambda_2 = 2$ and synaptic weight $w_2 = 1$. Both have the same mean membrane potential, $E[v_{mem1}] = E[v_{mem2}] = 2$, however the first neuron has a larger variance, $Var(v_{mem1}) = \frac{1}{2}(1)(1)(2)^2 = 2$; and $Var(v_{mem2}) = \frac{1}{2}(1)(2)(1)^2 = 1$ (see Figure 3.3).

Given a certain threshold higher than the mean, the distribution with the larger variance will have a higher probability of surpassing the threshold than the distribution with a smaller variance (see Figure 3.4).

The mismatch between means and variances isn't solely based on weight vector magnitudes, either. Consider another simple example expanded into another dimension, with two inputs and weights per neuron (see Figure 3.5). Based on the input/weight vector dot products, we would expect the decision boundary between these two neurons to be at the angular midpoint. However, in the spiking domain Neuron B dominates in much of the

[3]↑This section contains my work on a manuscript that is still under preparation [13]. It may or may not undergo a copyright transfer in the future, but will retain permission to be included here. This version should be considered a pre-print that is not necessarily endorsed by any potential future publisher in its current form.



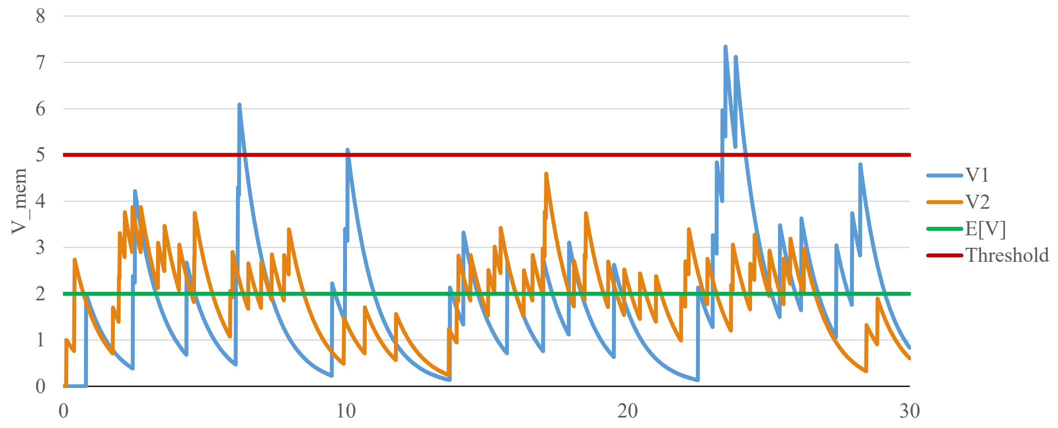
Inputs
Weights
Outputs
$E[V_{mem}] = \tau(\lambda \cdot w)$ $Var(V_{mem}) = \frac{1}{2}\tau(\lambda \cdot w^{\circ 2})$
$\lambda_1 = 1$
$w_1 = 2$
$E[V_{mem}] = 2$ $Var(V_{mem}) = 2$
$\lambda_2 = 2$
$w_2 = 1$
$E[V_{mem}] = 2$ $Var(V_{mem}) = 1$
$(\tau_{mem} = 1)$

Figure 3.3. A simple example of the same mean membrane potential, but different variances.

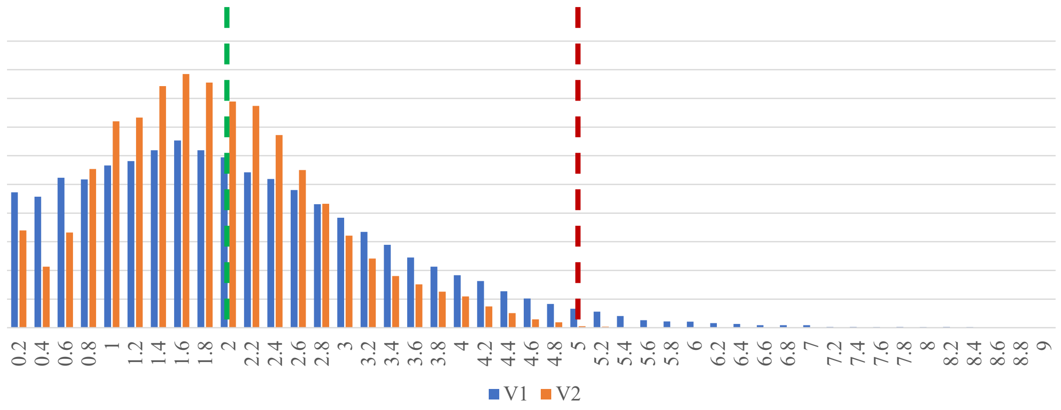
input space that is angularly closer to Neuron A. Neuron B was able to dominate more of the input space than expected because its membrane potential at those points in the input space had a larger variance than did the membrane potential of Neuron A. The difference in variance at those points is because while both neurons have weight vectors with the same magnitude, using a euclidean or L^2-norm, the Hadamard or element-wise squares of their respective weight vectors do not have the same magnitude.

3.3.2 Relation of v_{mem} standard deviation to weight L^4-norm

We know from Equation 3.18 that the variance of the pre-firing membrane potential is proportional to the dot product of the input and the Hadamard (or element-wise) square of the weight vector. Therefore, the variance is proportional to the L^2-norm, or euclidean magnitude of $w^{\circ 2}$, which can be rewritten as:



(a) Pre-firing membrane potentials over time.



(b) Histogram of pre-firing membrane potentials.

Figure 3.4. The temporal responses and distributions of the examples in Figure 3.3.

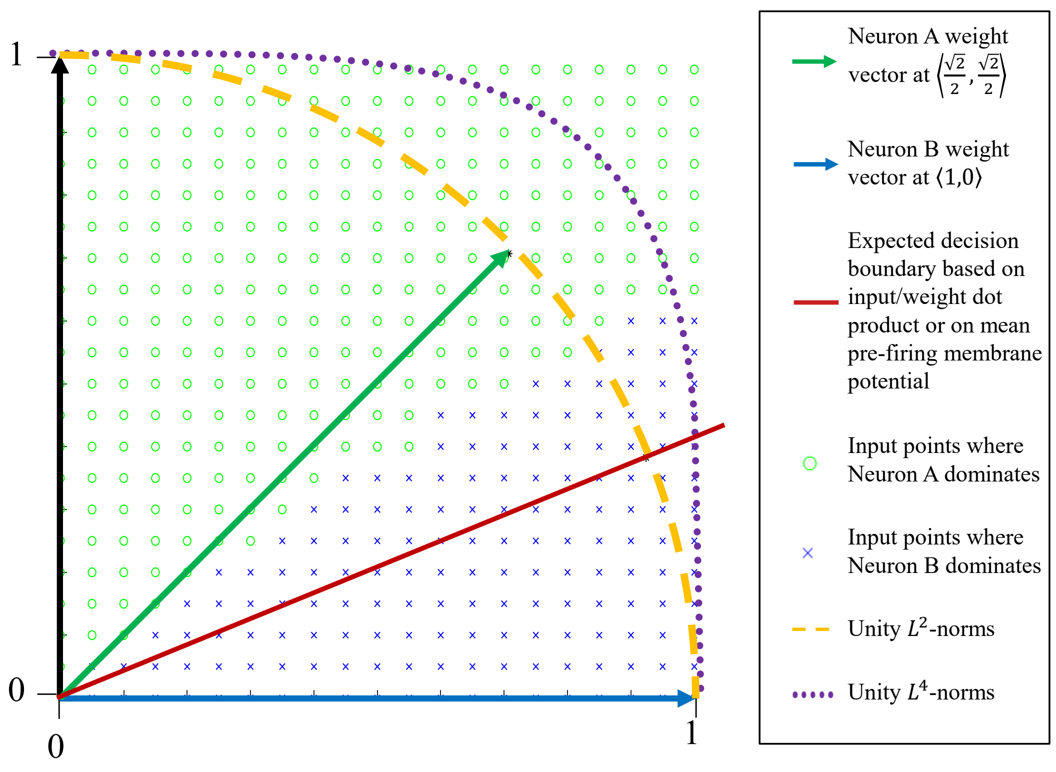


Figure 3.5. A two-dimensional example of neurons with the same weight magnitude (same L^2-norm), showing the input points at which each dominates in a spiking domain.

$$\begin{aligned}
|w^{\circ 2}|_{L^2} &= \sqrt{(w_1^2)^2 + (w_2^2)^2 + (w_3^2)^2 + ... + (w_n^2)^2} \\
&= \sqrt{w_1^4 + w_2^4 + w_3^4 + ... + w_n^4} \\
&= \left(\sqrt[4]{w_1^4 + w_2^4 + w_3^4 + ... + w_n^4}\right)^2 \\
&= (|w|_{L^4})^2 \qquad (3.20)
\end{aligned}$$

And since the standard deviation is the square root of the variance, the standard deviation of the membrane potential becomes directly proportional to the L^4-norm of the weight vector. Going back to the example in Figure 3.5, we now see that the standard deviation of the membrane potential of Neuron B was larger than the membrane potential of Neuron A at the points in question because Neuron B's weight vector had a larger L^4-norm than did Neuron A's, despite them having equivalent L^2-norms.

3.3.3 L^4-norm weight adjustments for SNNs conversions

For a given L^2-norm, a weight vector that is closer to an axis will have a larger L^4 norm. These are weight vectors that are more sparse or have a few larger individual components rather than many fairly equivalent individual components. For example, the vectors $< \frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2} >$ and $< 1, 0 >$ have the same L^2-norm, but their respective L^4 norms are ≈ 0.84 and 1.

Current conversion methods from trained non-spiking ANNs to SNNs assume that a neuron's output spike rate is proportional to the dot product of the input rate vector and the neuron's weight vector. Failing to account for the wider reach of neurons with a larger L^4-norm can cause non-ideal conversions. We proposed scaling a neuron's threshold, or equivalently its weight magnitude, by some function of the L^4-norm of the pre-conversion weight vector so that systems that operate on a dot-product assumption incur less of a "conversion penalty."

As a reminder, the mathematical modeling of the statistical distribution of the neuron pre-firing membrane potential that we have been using was built on the *Poisson* rate-encoding of the input. We note here that after the input layer, the spike trains may no longer be considered Poisson point processes, as the potential has a transient state, meaning that the probability of a neuron that is within the network firing is dependent on how recently it has previously fired and reset its potential. This observation would imply that the following adjustments are more effective at the first layer of the network. Our simulations verify that, and so the following adjustments are applied to the first layer only.

3.3.4 Adjustment methodology

For clarity, we will begin by discussing how an individual neuron's firing threshold v_{th} should be scaled. In practice, we instead scale the weights by the inverse amount, which has the exact same functionality but still allows us to maintain a single threshold voltage for all neurons in the layer. The reason for calculating the threshold change instead of the weight magnitude change is so that the neuron's pre-firing membrane potential distribution remains consistent throughout the calculation process.

As discussed above, because training in a non-spiking domain is based on the dot product of the input and weight vectors, the mean pre-firing membrane potential of neurons in the converted network will be proportional to that dot product. For more accurate use of those parameters in the converted SNN, we instead want those computed values to be reflected in each neuron's respective probability of having its potential reach its firing threshold. We propose scaling each neuron's threshold to a new value $\widehat{v}_{th_{\mathrm{j}}}$ so that the number of *standard deviations* it is over the neuron's mean membrane potential is proportional to the difference between the original threshold v_{th_0} and the mean.

$$\frac{\widehat{v}_{th_{\mathrm{j}}} - \mu_{V_{\mathrm{j}}(t)}}{\sigma_{V_{\mathrm{j}}(t)}} \propto v_{th_0} - \mu_{V_{\mathrm{j}}(t)} \tag{3.21}$$

In other words, we want the reach of the threshold over the mean to be scaled by the standard deviation, which is proportional to the L^4-norm:

$$\frac{\widehat{v}_{th_\mathrm{j}} - \mu_{V_\mathrm{j}(t)}}{v_{th_0} - \mu_{V_\mathrm{j}(t)}} \propto \sigma_{V_\mathrm{j}(t)} \propto |\vec{w}_\mathrm{j}|_{L^4} \tag{3.22}$$

To keep the parameters in the same range as training, we scale it with the average L^4-norm of the neurons in that layer, μ_{L^4}:

$$\frac{\widehat{v}_{th_\mathrm{j}} - \mu_{V_\mathrm{j}(t)}}{v_{th_0} - \mu_{V_\mathrm{j}(t)}} = \frac{|\vec{w}_\mathrm{j}|_{L^4}}{\mu_{L^4}}$$

which allows us to calculate the new threshold for each neuron:

$$\widehat{v}_{th_\mathrm{j}} = \mu_{V_\mathrm{j}(t)} + \frac{|\vec{w}_\mathrm{j}|_{L^4}}{\mu_{L^4}}(v_{th_0} - \mu_{V_\mathrm{j}(t)}) \tag{3.23}$$

The mean membrane potential is input-dependent, and cannot be calculated only with the weight values. We sample it by running a forward pass on a single batch of the training set through only the first layer to measure the mean membrane potential of each neuron. This extra single batch of inference need not add much computation time to the conversion because the single-batch inference is already performed during the threshold balancing step of the conversion, and we simply record the mean potential within the first layer during that inference.

Then, as 3.23 gives us the new threshold, we instead us its proportion to the original threshold inversely as the L^4-norm weight adjustment scalar, s_{L^4}, as mentioned at the start of this subsection:

$$s_{L^4} = 1/(\frac{\widehat{v}_{th}}{v_{th}}) \tag{3.24}$$

Smoothing ratio, r_s

The variance is also input dependent. Although the magnitude of a given input doesn't have an affect of the conversion penalty–since it scales all the potentials of all competing neurons equivalently–the angle of the input plays a role. The variance is proportional to

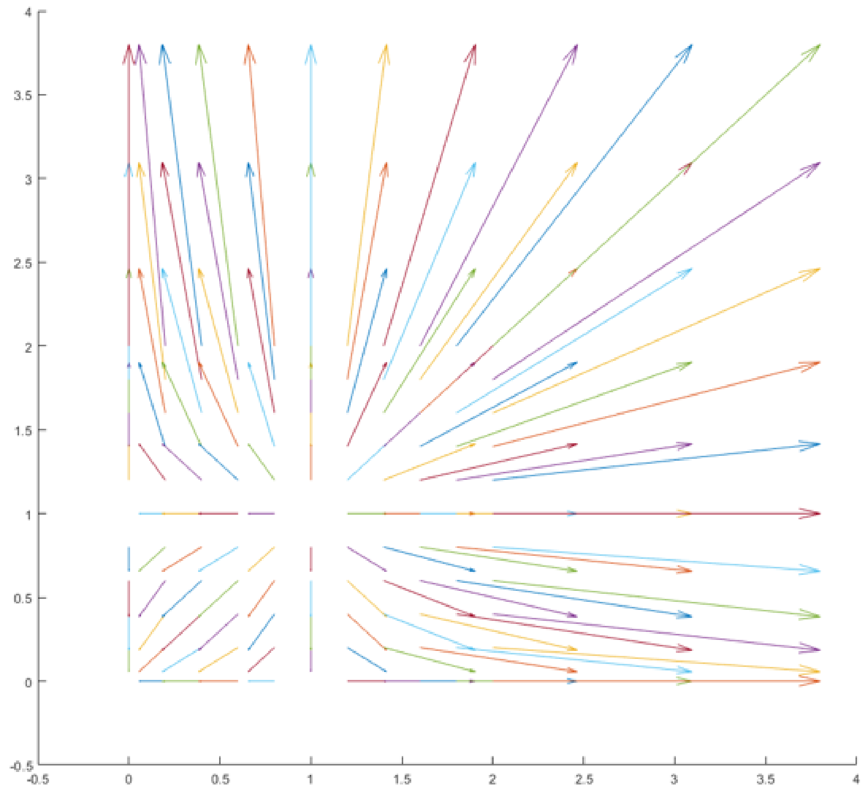


Figure 3.6. The Hadamard angle drift. In this figure, each vector represents the translation of performing an element-wise square–the shown start point being an original vector endpoint, and the shown endpoint being the endpoint of the element-wise square of the original vector.

the cosine of the angle between the input vector $\vec{\lambda}$ and the Hadamard square of the weight vector $\vec{w}_{\mathrm{j}}^{\circ 2}$. Figure 3.6 shows a 2-d plot of how performing the element-wise square on a vector changes not only its magnitude, but also its direction.

Vectors that are along an axis or are along the ones vector do not change their angle from the origin, but every other vector drifts away from the ones vector toward the axes. This means that for a given angle between $\vec{\lambda}$ and $\vec{w}_{\mathrm{j}}$, the corresponding angle between $\vec{\lambda}$ and $\vec{w}_{\mathrm{j}}^{\circ 2}$ will either be smaller or larger, depending on if the input vector is located in the direction of the drift or away from it, respectively.

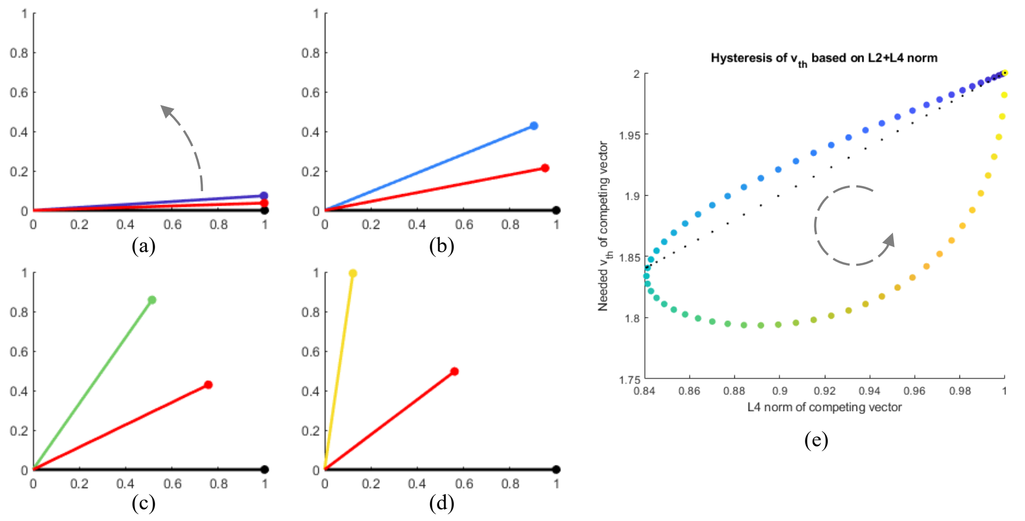


Figure 3.7. A 2-d simulation of the needed v_{th} of a sweeping weight vector (purple to blue to green to yellow) that is competing against a static weight vector (black) at <1,0> when presented with the midpoint input vector (red). The dotted gray arrow indicates the direction of the sweep. Subfigures (a), (b), (c), and (d) are snapshots of the sweep from an angle of zero to $\pi/2$. Subfigure (e) shows the corresponding color coded v_{th} of the sweeping neuron in order for it to achieve the same spiking activity as the constant neuron, whose v_{th} remains static at 2. The smaller black dots represent the v_{th} that would correspond to a mean + variance of the $L^2 + L^4$ norms. ($\tau = 1$)

This creates a hysteresis (see Figure 3.7), meaning that there cannot be a single v_{th} for a neuron that is input-independent and completely correct for the temporal variance. What this means is that in some cases, the L^4-norm correction will overshoot, and potentially move to a portion of the loss space that increases loss rather than decreases loss.

I minimize this effect by smoothing the L^4-norm variance estimations with a weighted average of all the L^4-norms in the layer, where the layer average μ_{L^4} is weighted by r_s, a hyperparameter, as follows:

$$|w_{\mathrm{i}}|_{\widehat{L^4}} = \frac{(|w_{\mathrm{i}}|_{L^4} + r_s \mu_{L^4})}{1 + r_s} \tag{3.25}$$

Then $|w_{\mathrm{i}}|_{\widehat{L^4}}$ is used in place of $|w_{\mathrm{i}}|_{L^4}$ in the weight adjustment calculation of 3.23.

Additionally, there is the additional approximation of the mean potentials that are profiled from a single batch and averaged over those inputs and across windows of convolution rather than input-dependent for each sample. We additionally apply the smoothing ratio, then, to determine how much of the correction to apply. The larger the smoothing ratio r_s, the smaller the weight adjustment.

$$\hat{s}_{L^4} = \frac{s_{L^4} + r_s}{1 + r_s} = \frac{s_{L^4} - 1}{1 + r_s} + 1 \tag{3.26}$$

Finally, after scaling the weights in the first layer by $\hat{s}_{L^4}$, we calculate the overall change in the average weight magnitude of the layer and re-scale the whole layer to match the original average weight magnitude so that the spike rate remains on-par with the baseline.

Threshold balancing

In traditional non-spiking to spiking conversion, a threshold must be assigned to each layer. Because we are changing the distribution of the membrane potentials and squashing the outliers, the same threshold that was optimal for the baseline will not be the same as threshold that is optimal for the L^4-norm adjusted networks. (Recall that the adjustment didn't actually change the threshold, but instead the weights inversely.) Therefore, for a more fair comparison, we perform the same threshold voltage sweep for both the baseline and the adjusted networks and choose the best for each.

3.3.5 Adjustment results

We now present our results, which are being prepared for publication [13]. We tested Cifar-10 on both VGG-5 and RESNET-20 as well as Cifar-100 on VGG-11 using both soft-resent and hard-reset neurons. We also tested five different leak scalars for each network, performing the threshold and smoothing ratio hyperparameter sweep on each using the training data. (Note that these values show the leak scalar at each time step rather than the corresponding leak time constant τ. For these figures, 1 means no leak, and lower values

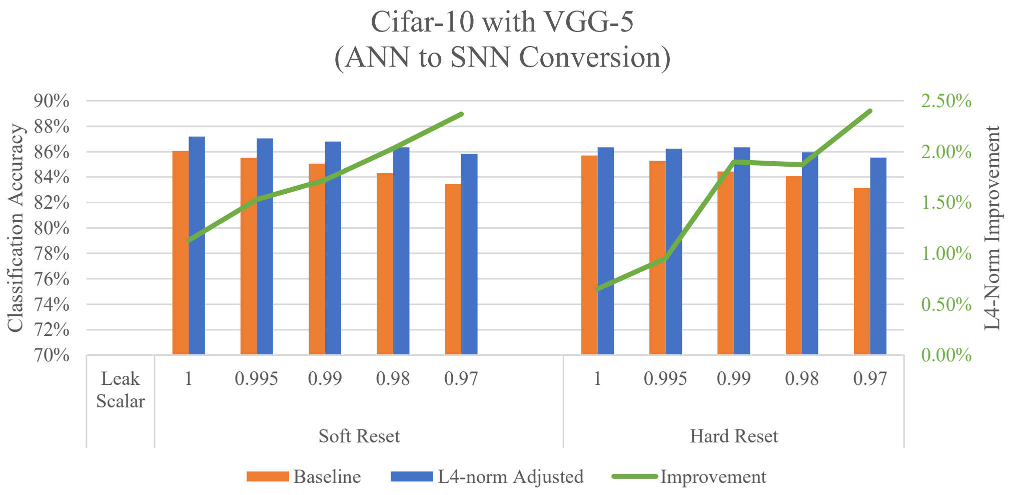


Figure 3.8. Classification accuracy of an SNN converted from a pre-trained non-spiking ANN with the VGG-5 network model on the Cifar-10 dataset, showing the improvement by adjusting weights according to the L^4-norm.

mean faster leak.) The ANN to SNN conversion followed the methodology, code, and some of the same pre-trained networks as in [30].

Cifar-10 on VGG-5

When VGG-5 is trained on Cifar-10 in a non-spiking ANN, the ANN accuracy is 89.32%. We convert that pre-trained network to a spiking network operating with 75 time steps per inference. Fig. 3.8 shows the accuracy when converting to a spiking network for both the baseline conversion and the L^4-norm adjusted conversion. The L^4-norm adjusted networks outperform the baseline in all scenarios and regains some of the accuracy lost during conversion without needing any additional training in the spiking domain (which is costly).

As expected, for all networks the accuracy drops when the leak is faster and also drops with the hard reset, in both cases because of information loss. However, it is still beneficial to explore these scenarios. Leaky neurons allow for identifying temporal patterns that can be erased by non-leaky neurons, and there are also emerging analog devices that have intrinsic parasitic sub-threshold leak. Additionally, hard resets can sometimes be cheaper to implement in hardware, as a value simply needs to be discharged to ground rather than performing a subtraction operation. Note that the L^4-norm improvement increases as the leak gets faster. This is because the temporal stochasticity and variance plays a larger roll in the spiking activity when there is more leak.

However, we still get an improvement even in the no-leak scenario, which we had originally not expected. In the no-leak scenario, the traditional conversion dropped 3.27 percentage points to 86.05% while the L^4-norm adjusted network was able to regain 1.13 of those lost percentage points. The reason this was unexpected is that because without leak, there is no steady-sate; every positively-bound membrane potential will continue to increase at an expected rate proportional to the input/weight dot product and eventually fire. In fact, mathematically the expected time to fire–and thus the spiking activity–of a no-leak neuron will always be proportional to the desired input/weight dot product, no matter the variance in that rate. The benefit here, therefore, must be that that by stabilizing the variance of that firing rate, it reduces the temporal noise in the spike rates for the subsequent layers.

Cifar-10 on RESNET-20

When RESNET-20 is trained on Cifar-10 in a non-spiking ANN, the ANN accuracy is 92.79%. We convert that pre-trained network to a spiking network operating with 250 time steps per inference. Fig. 3.9 shows the accuracy when converting to a spiking network for both the baseline conversion and the L^4-norm adjusted conversion. (Note that because of information loss, RESNET-20 failed to generalize for the faster two leaks in the hard reset scenario for both the baseline and the L^4-norm adjusted networks, and are not shown.)

In contrast to the smaller network VGG-5, RESNET-20 did not see a benefit in the no- and low-leak, soft-reset scenarios by performing the L^4-norm adjustment. We suspect that because RESNET-20 is a larger network, the de-noising of the internal spike rates was less effective because the redundancy was able to handle the noise. Yet, the benefit still exists in the faster leak scenarios, verifying that even with a larger network we need to correct for the temporal variance affecting the ability to reach the firing threshold.

We further note that in the hard reset scenarios, the L^4-norm adjustment *did* provide a benefit even in the no- and slow-leak scenarios. This benefit must because the L^4-norm adjustment reduces the overshoot of high-varying potentials over the threshold, reducing the associated information loss. Thus there are three areas in which adjusting for the L^4-norm provides a benefit: (1) in all networks with sufficient leak, by accounting for the gap between the mean and threshold in proportion to the standard deviation; (2) in no-leak networks that are smaller, or less redundant by reducing the spike rate noise past the first layer; and (3) in networks with hard-reset neurons, even without leak and with redundancy, by accounting for the statistical overshoot past the threshold, reducing information loss.

Cifar-100 on VGG-11

When VGG-11 is trained on Cifar-100 in a non-spiking ANN, the ANN accuracy is 71.21%. We convert that pre-trained network to a spiking network operating with 125 time steps per inference. Fig. 3.10 shows the accuracy when converting to a spiking network for both the baseline conversion and the L^4-norm adjusted conversion.

With this network and dataset, we see similar areas of improvement as with RESNET-20. While there is little to no benefit with soft reset and no- or slow- leak, there is still the expected benefit at faster leaks and for hard reset. The overall improvement for VGG-11 is higher than that of RESNET-20, likely because CIFAR-100 is a more complicated dataset and so the corrections play a bigger role.

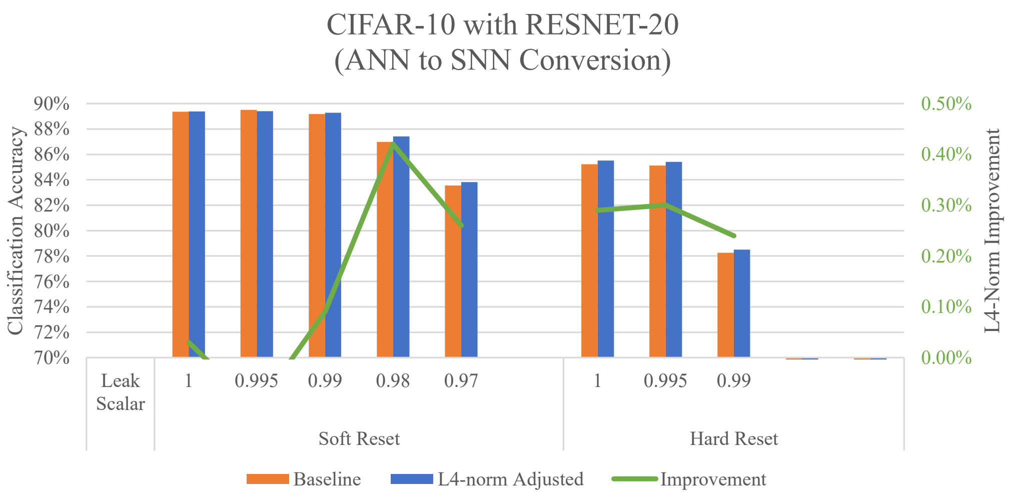


Figure 3.9. Classification accuracy of an SNN converted from a pre-trained non-spiking ANN with the ResNet-20 network model on the Cifar-10 dataset, showing the improvement by adjusting weights according to the L^4-norm.

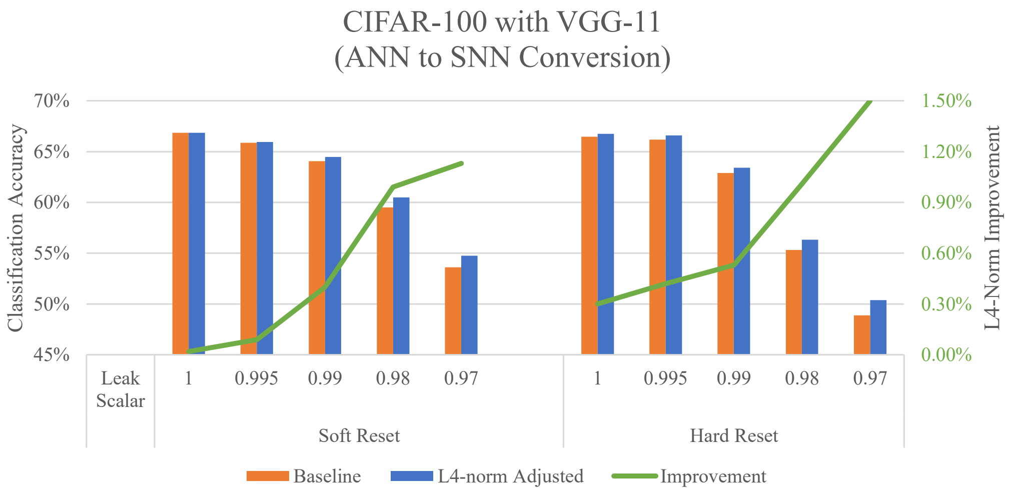


Figure 3.10. Classification accuracy of an SNN converted from a pre-trained non-spiking ANN with the VGG-11 network model on the Cifar-100 dataset, showing the improvement by adjusting weights according to the L^4-norm.

4. LIFELONG LEARNING IN ONLINE SPIKING NEURAL NETWORKS

One of the current obstacles preventing fully autonomous, unsupervised learning in dynamic environments while maintaining efficiency is the *stability-plasticity dilemma*, or the challenge of ensuring that the system can continue to quickly and successfully learn from and adapt to its current environment while simultaneously retaining and applying essential knowledge from previous environments [31]. This chapter begins by introducing and discussing this problem, as extracted from my work in [12] (©2020 Allred and Roy)[1].

There have been a handful of terms used in literature to describe the process of learning from data that is temporally distributed inhomogeneously, such as the terms incremental learning, sequential learning, continual learning, and lifelong learning. In this work, we will use the term "lifelong learning." *Lifelong learning* is the process of successfully learning from new data while retaining useful knowledge from previously encountered data that is statistically different, often with the goal of sequentially learning differing tasks while retaining the capability to perform previously learned tasks without requiring retraining on data for older tasks. When traditional artificial neural networks are presented with changing data distributions, more rigid parameters interfere with adaption, while more flexibility causes the system to fail to retain important older information, a problem called *catastrophic interference* or *catastrophic forgetting*. Biological neuronal systems don't seem to suffer from this dilemma. We take inspiration from the brain to help overcome this obstacle.

To avoid catastrophic forgetting, important information from older data must be protected while new information is learned from novel data. Non-local learning rules may not provide such isolation. Localized learning such as STDP, on the other hand, may provide the desired segmentation while also being able to perform unsupervised learning, which is critical for lifelong learning in unknown environments.

However, even though STDP learning is localized, it is still susceptible to catastrophic forgetting because the algorithms that employ STDP are traditionally designed for randomized input ordering. Certain features, such as homeostasis, attempt to distribute the effect

[1]↑Used with permission.

of input groupings globally in order to benefit from the full network. Without a temporally uniform distribution of classes, traditional STDP algorithms still lose important older information, which is either replaced by or corrupted with information from newer samples.

Many recent papers have tackled the challenge of lifelong learning without catastrophic forgetting, but they are not designed to target the goal of this work, which is autonomous learning on an online neuromorphic system. This goal requires real-time unsupervised learning, energy efficiency, and fixed network resources. The works in [32]–[43] all employ supervised or reinforcement learning methods, in some way provide the network with the knowledge of when a task change occurs, or provide access to previous samples for retraining. For example, the work by [43] requires that the system be allowed a parameter-"importance update" period on the older task(s) before proceeding to a new task. Additionally, [32]–[37], [44] are also not applicable to localized learning rules that may be employed on spiking networks. And [41], [45], [46] are morphological systems that do not work with static-sized networks, which would exclude them from direct mapping onto physical hardware implementations.

This chapter begins with a discussion on the challenge of lifelong learning (Section 4.1) and a motivation for the need to address catastrophic forgetting (Section 4.2). I then detail three proposed methods for lifelong learning: forced firing [14] (Section 4.3), dopaminergic learning [12] (Sections 4.4, 4.5, and 4.6), and adaptive synaptic plasticity [15] (Section 4.7). This chapter concludes with a discussion on a relationship between these lifelong learning methods and the properties of an NiO Mott insulator material [16] (Section 4.8).

4.1 The Challenge of Lifelong Learning

Backpropagation has proven a successful learning algorithm for deep neural networks. The accuracy of this approach depends on proper stochastic gradient descent or SGD, also known as incremental gradient descent, in which many small, global adjustments to network weights are performed while iterating over samples from a training dataset. These samples, however, must be drawn from a random distribution of the dataset—hence the name "stochastic" gradient descent—intermixing the classes so that each class can affect the direction of descent for correct error minimization throughout the entire training process.

The need to draw training samples from a random distribution is an obstacle for on-line learning, especially when the system encounters novel data. Backpropagation in an on-line system for real-time learning proves difficult when the input from the environment is uncontrolled and unknown. With traditional SGD, the system typically has three choices to attempt learning from novel data: (1) train normally on inputs in the order seen; (2) periodically go offline and retrain from an updated dataset; or (3) maintain an online storage of previous samples to intermix with the new samples, providing a simulated random sampling. The latter two choices are costly and inhibit real-time learning, while the first catastrophically violates SGD.

4.1.1 Catastrophic forgetting due to global interference

If a uniformly randomized order is not provided, e.g. samples are grouped by class and classes are presented sequentially to the network, then the gradient descent followed by latter samples will likely disagree with the direction from previous samples. This conflict causes the network to fail to reach an error minimum that respects older tasks, as at each period of time in the training process the network essentially attempts to globally optimize for only the current tasks, agnostic as to whether or not that particular direction increases the error for older tasks. Latter samples erase or corrupt the information learned from previous samples, causing catastrophic forgetting.

One of the largest underlying causes of catastrophic forgetting in backpropagation algorithms is the reliance on a global error. Calculating weight updates from the current sample's global error means that the current sample may globally affect network weights. Biological neuronal learning, on the other hand, appears to be significantly localized, with synaptic weight updates being a function of local activity, causing different regions to be responsible for different tasks. While distributed representations promote generalization in neural networks, rapid learning of novel information may not require significant modifications to low-level distributed representations in a sufficiently trained network. It has been shown that the IT cortex contains a large-scale spatial organization, or "shape map," that remains significantly stable over time [47], even while learning novel information. Lee and DiCarlo

[48] have shown that the stable earlier levels of the visual cortex are capable of representing the generic structure and composition of never-before-seen inputs with an already-learned understanding of the physical world that remains constant through the remainder of life–for example, an understanding of lines, edges, curves, and colors at the lowest levels and an understanding of rotations, shading, and physical properties at subsequent levels. Thus, it is likely that lifelong learning need only occur in the last one or two layers of a neural network, where local learning may sufficiently classify from a read-out of the higher-dimensional generalizations that have been learned previously.

4.1.2 Catastrophic forgetting in localized learning due to homeostasis

Many leading STDP-trained SNNs employ adaptive thresholding, in which a neuron's firing threshold increases each time it fires and otherwise decays, preventing specific neurons from dominating the receptive field (see 2.4.1). Adaptive thresholding helps achieve homeostasis by distributing the firing activity between neurons. However, adaptive thresholding assumes a temporally random distribution of input samples and often causes catastrophic interference when the environment changes [14]. For lifelong learning, adaptive thresholding must be modified to account for long-term variations in spiking activity that would occur when processing temporally variant input distributions.

4.1.3 The need for forgetting in online systems

For successful lifelong learning, there must be network resources available to learn new information. In an online system with finite resources, some forgetting of older knowledge is required to make room for information from new data. As mentioned earlier, there are morphological systems that logically grow the network to accommodate new information, even employing pruning techniques when necessary if the network grows too large. However, for our goal of online learning on neuromorphic hardware, inserting and removing physical components of the network is not an option, and instead existing network components must be re-purposed to learn a new task when network capacity is reached, requiring some forgetting.

Additionally, in some cases, forgetting may actually be beneficial. Forgetting outlier data can improve generalizations, and forgetting stale data can allow the system to adapt to a changing environment if new information directly contradicts older information. Because some forgetting must occur, this work seeks to control the forgetting process to protect the most vital information, minimizing accuracy loss.

4.1.4 Problem description: sequential unsupervised learning

In Section 2.4.3, we described the traditional unsupervised digit recognition problem in which samples are drawn from all classes uniform randomly throughout the training process, providing a temporally consistent distribution. That is the *interleaved* scenario and represents traditional offline learning in which all training data is already available.

To test lifelong learning, the methods proposed in this chapter (forced firing, dopaminergic learning, and adaptive synaptic plasticity) are tested with the worst-case possible ordering–the *disjoint* scenario in which all samples from one digit are presented before moving to the next digit and never returning to previous digits after changing classes. The disjoint scenario with sequentially presents classes represents a changing, dynamic environment.

In both scenarios, labels are not provided during training. The network receives no external indication of when a task/digit change occurs in the disjoint scenario. Other than sequentializing the MNIST dataset in the disjoint scenario, our training and testing procedure follows closely with that of [22], who demonstrated competitive unsupervised STDP training on MNIST in the traditional interleaved scenario. The following discusses modifications to the baseline setup in Section 2.4.3 for the proposed lifelong learning methods.

Training process

In contrast to the baseline setup described previously, if a given sample does not produce enough output spikes we do not continue to increase the input firing rate during training in the Forced Firing and Dopaminergic Learning methods, since they have other methods of stimulating neurons in the absence of a good match. Because these alternative approaches are part of the learning process, they are only employed during training, and the increased input rates do occur as normal during testing.

Testing process

In the disjoint scenario, we measure effective lifelong learning over time by evaluating each network after each task change to determine its current accuracy for all classes seen up to that point. Networks in the interleaved scenario are only evaluated at the end of the training process. As in the baseline, during evaluation we pause learning and freeze network parameters to prevent samples from older classes or samples from the testing set from affecting the network.

4.2 A Motivating Example

In many real-time applications, online machine learning must continually process new data to tune its performance or alter its response when confronted with changing environments. In these situations, it is often essential to retain at least a portion of previously-learned information. A common solution to avoid overwriting old information is to provide data reinforcement both during initial training and any subsequent training. This motivating example for data reinforcement is extracted from my work in [14] (©2016 IEEE)[2].

4.2.1 Data reinforcement during initial training

The practice of data reinforcement is that of re-presenting previous information to the network together with the new information so that the old data is sufficiently retained and stays balanced with the new information. Data reinforcement is used during the initial training process by intermixing input classes and iterating repeatedly over a varied selection so that later input patterns do not replace previous patterns.

[2]↑In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Purdue's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

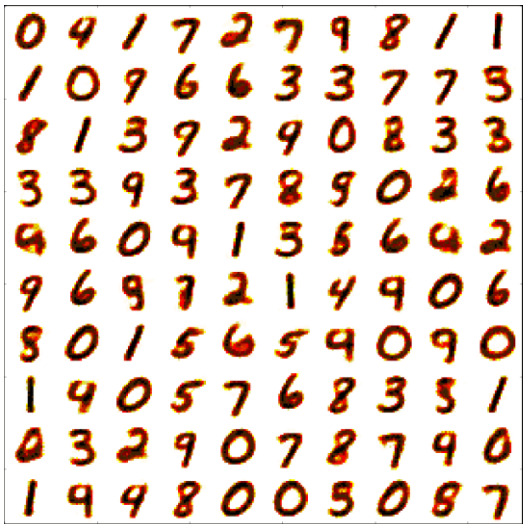
Without data reinforcement, the latter information may dominate and the former information may be lost. To emphasize this point, we show two examples of an SNN attempting to learn the digits ‘0’ through ‘9,’ one with and the other without data reinforcement. Figure 4.1a shows the first example in which the network is presented with a randomized mixture of input images from the MNIST data set, allowing data reinforcement to balance the different input patterns during the training process.

In contrast, Figure 4.1b shows the second example in which previously learned data categories were not re-presented to the network, i.e. the classes were presented sequentially: first all the images for digit ‘0,’ and then all the images for digit ‘1,’ and so on until digit ‘9.’ At each stage of the training process, the new image patterns were unacceptably learned over the top of the old image patterns, causing the learned patterns to morph together until the network became useless at correctly categorizing them. The absence of data reinforcement prevented the network from successfully learning during its initial training.

4.2.2 Data reinforcement during subsequent training

Not only is data reinforcement essential for the initial training process, but every time subsequent information is encountered, old information must traditionally be re-presented to avoid being lost. Figure 4.2a shows an SNN initially trained using data reinforcement (i.e., all classes intermixed) for the digits ‘0’ through ‘8.’ After this initial training, we subsequently presented the input image examples for the digit ‘9’ without reinforcing any of the previous digits, resulting in data being overwritten. Figure 4.2b illustrates how weights associated with each neuron were affected, with many learned digits slowly transforming into the digit ‘9,’ especially the weights for patterns of similar digits such as ‘4’ and ‘7.’

Without reinforcing previously-learned data, traditional neural networks have difficulty retaining previous knowledge while trying to continue to learn. However, in online real-time learning it is often impractical to store all previous input data for retraining each time new input data is encountered. A neural network capable of sequential learning—being trained on new data without reinforcement of old data—would naturally lend itself to online real-time machine learning.


(a) Interleaved scenario - all 10 digits.

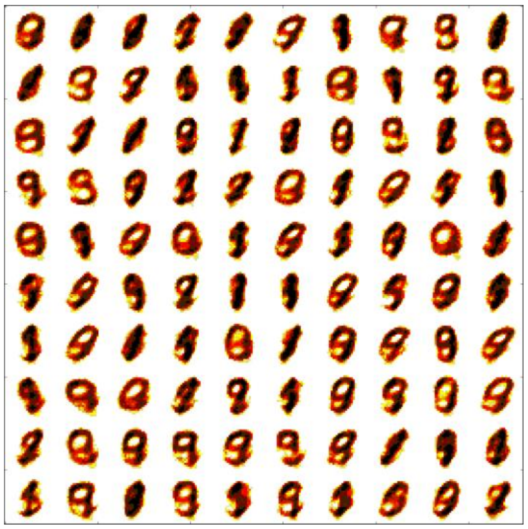
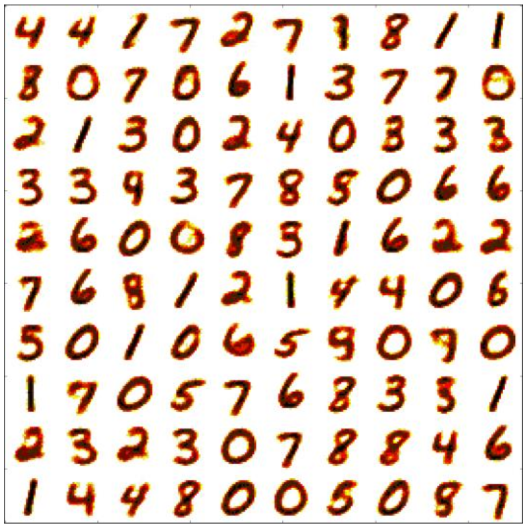

(b) Sequential scenario - '0' through '9.'

Figure 4.1. Synaptic weights of reference vector neurons in a traditional SNN after two contrasted training scenarios: (a) interleaved; and (b) sequential. Each of the 100 (10x10) excitatory neurons has 784 (28x28) synaptic weights, rearranged in a two-dimensional grid for display. Darker weights are stronger synaptic connections. (©2016 IEEE [14])


(a) Interleaved scenario - all digits except '9.'

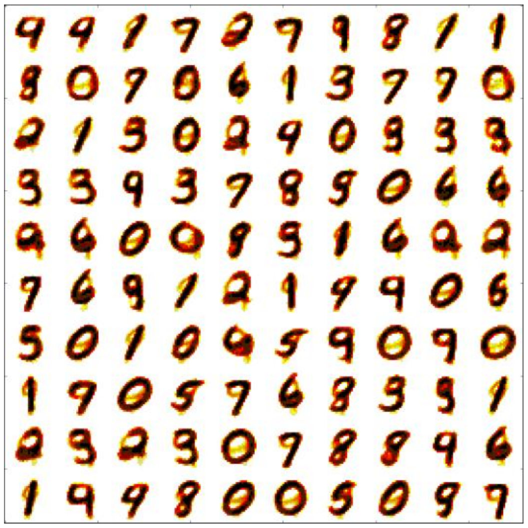

(b) After subsequently learning digit '9.'

Figure 4.2. A training scenario (before (a) and after (b)) showing the attempt to add new information (digit '9') to an already trained SNN. Without reinforcing old data, the stored information is corrupted. (©2016 IEEE [14])

4.3 An Initial Proof of Concept: Forced Firing

This section presents an initial approach for designing a sequentially learning SNN, taken from my work in [14] (©2016 IEEE)[3]. The design framework explores four main components: augmenting the network with over-provisioned neurons, guiding new input signals to these supplemental neurons, determining when additional neurons are necessary, and reassigning old neurons under network size constraints.

4.3.1 Adding neurons to an SNN

Lifelong learning of sequentially presented information requires that new information be learned with a limited or controlled effect on previous training. The only certain way to completely avoid altering previously learned information is to exclude any trained portions of the network during any subsequent training. If trained portions are excluded, then untrained portions must be made available.

To provide untrained resources for sequential learning, an SNN implemented directly in hardware may be over-provisioned with additional neuron components. These reserved or dormant neurons may remain inactive throughout initial training stages during which they are not needed. For certain hardware implementations, dormant neuron components may be power-gated [49] to reduce leakage current and overall power consumption while they are dormant. Then, when the network encounters new information after the initial training, the dormant neurons that were reserved may be activated or powered-on so that the new information can be encoded in their respective synaptic weights.

However, the naïve approach of simply augmenting the network size to learn new information encounters a few obstacles. Figure 4.3 illustrates these obstacles. An SNN of 90-neurons was augmented with 10 additional neurons that remained dormant during the training of digits '0' through '8' (see Figure 4.3a). Then the dormant neurons were activated while the training images for digit '9' were presented to the network. The untrained weights of the supplemental neurons that were added to the network were not intrinsically matched

[3]↑See previous footnote 2.

to the input spike trains associated with the training data set of the digit '9,' causing them to spike infrequently and only weakly learn the new information (see Figure 4.3b).

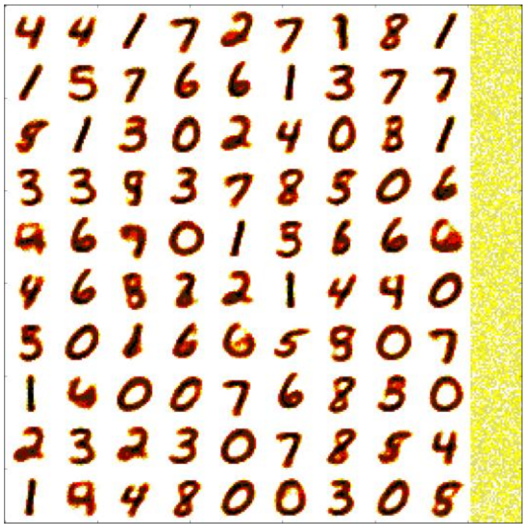
Even when the data set was applied 3X as long to allow the new neurons to learn the information, the input spike trains were better matched to the synaptic weights of previously-trained neurons, especially those associated with the digits '4' and '7' (see Figure 4.3c), significantly overwriting the old information. Without reinforcement of the previous information, merely adding neurons to this network is insufficient. The new input signals must somehow be guided through the synaptic weights of the added neurons.

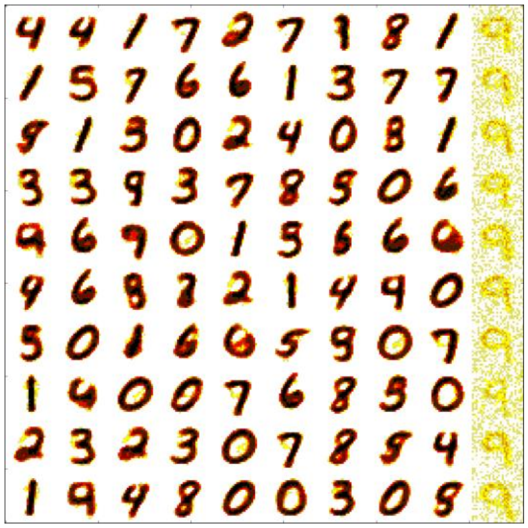
4.3.2 Guided STDP learning with forced firing

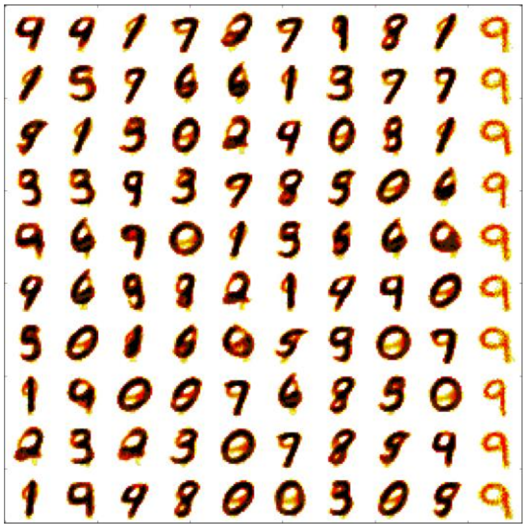
The principle element of STDP learning is the order and timing of the firing in pre- and post-synaptic neurons. As discussed in Section 2.4.2, the synaptic weight is strengthened when the post-synaptic neuron fires shortly after the pre-synaptic neuron fires. This work exploits STDP to guide input signals to the added neurons through a method called forced firing. Forced firing consists of presenting the input signals to the network and then shortly thereafter causing the added neurons to fire. By forcing the new neurons to fire, the synaptic weights that connect the input neurons to the added neurons are strengthened. An SNN with forced firing gives the added neurons the needed preference over the previously-trained neurons, providing a layer of protection to the old information. In addition, this method is enhanced by sending inhibitory signals to the previously trained output neurons, making it harder for them to fire and further protecting the old information.

Figure 4.3d shows the results of forced firing using the same setup as the previous example. The 10 neurons that were added were forced to fire while the previously-trained neurons were inhibited. Forced firing successfully limited firing to the added neurons, preserving the previous knowledge of the network while allowing the new information to be learned.

However, this example highlights several issues that need to be addressed: How many new neurons should be added to the network? How does the network know when new information is being presented? (This small example relied on supervised learning.) Many of the new neurons learned very similar patterns because they were all forced to fire. For input patterns


(a) The weights of 90 neurons that initially learned all digits except '9' with data reinforcement; 10 dormant neurons.


(b) The same SNN after then using the dormant neurons to attempt to subsequently learn digit '9.'


(c) The weights of the same SNN after further applying the input patterns for the digit '9' for 3X as long as in (b).

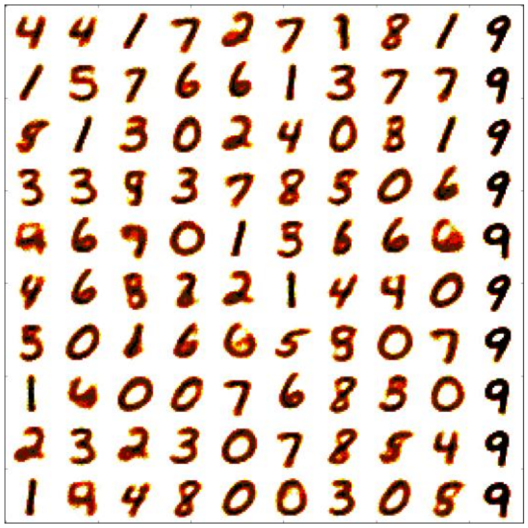

(d) The same network from (a) after instead learning the digit '9' using forced firing of the dormant neurons.

Figure 4.3. The synaptic weights of a pre-trained SNN attempting to use reserved dormant neurons to subsequently learn the digit '9.' (©2016 IEEE [14])

that vary significantly, should different neurons be forced to fire? For how long should the new neurons be forced to fire? (In this example, forced firing occurred for the entire data set of presented ‘9’s. This caused unnecessary forced firing after the input patterns had already been sufficiently learned, causing an effectively larger learning rate as manifested by the darker weights.) How should the network respond if previously-learned data is re-encountered? (Augmenting the network should only occur for new data.) Resolving these issues requires an unsupervised network capable of true sequential learning—one that can learn data in any order, with or without reinforcement, as detailed next.

4.3.3 Implementing unsupervised sequential learning

During supervised training, the data labels only provide the knowledge of when a new category or class of information is presented. Ideally, neurons should be added to the network as they are needed so that the allocation of neurons can depend on data complexity, independent of data labels. Accordingly, the defining task of such a network is to determine when new information is presented, and whether the new information merits forced firing.

In a spiking neural network, the frequency of spikes at the output layer can indicate how well the input pattern was recognized by the network. Too few output spikes suggests that there is not a good match in the network for the current input pattern. With an established threshold for output spike count (see Section 4.3.5), the network makes a decision whether or not to allocate a new neuron to help learn the current input data through forced firing.

Because neurons are added as they are needed, the network can begin this learning process without any previous offline training. In essence, the network can start with zero active neurons and no initial training, activating dormant neurons one-by-one as all information is learned sequentially. If previously learned information is encountered, the output spike count threshold will be surpassed, indicating that a new neuron is not required and causing the old data to be reinforced, as desired to refine generalizations.

This threshold may be tuned to control the sensitivity of the network to new information, which may be affected by the availability of dormant neurons. In a network where many dormant neurons are available, the threshold can be set low, providing easy access to the extra neurons for those input patterns that do not reach the threshold. On the other hand, a smaller network with few available dormant neurons should have a high threshold, restricting new neurons to only information that is very different than previously learned data. However, a finite network with limited neuron resources will eventually run out of available dormant neurons. Sequential learning proves valuable when dealing with these limited network memory scenarios.

4.3.4 Working with a static network configuration

Reserved dormant neurons may not be available to be added to the network, especially under power and area constraints. A finite network must then perform a tradeoff between learning new information and retaining old information. If new information must be learned at the expense of old information, it may be desirable to control the loss of information in a way that a small level of detail is lost from each category of information rather than an entire category being forgotten.

In such a scenario, graceful degradation of accuracy can be achieved by strategically selecting sections of the previously trained network to be reassigned to learn the new information. This selection focuses on idle neurons, which are defined by their infrequent spike count (see Section 4.3.5) because they contribute less often to the overall network computation. When the network has decided to allocate a new neuron but does not have any dormant neurons available, the neuron that has been most idle is reassigned through forced firing.

Therefore, in addition to the benefit of not requiring data reinforcement, our sequentially learning SNN provides a valuable tradeoff. High accuracy may be maintained as new information is learned at the expense of increased power and area cost due to the activation of dormant neurons (if they are available), or if network size is limited, the overall network accuracy may be gracefully degraded to expand the network's capability as new information is learned. Overall, these advantages are provided by forced firing and knowing which neurons to fire and when to fire them. Figure 4.4 delineates the system-level approach to implementing the proposed sequentially learning SNN, which we experimentally validate, as demonstrated in the next two subsections.

4.3.5 Methodology

Simulations for this experiment were performed using Python and the BRIAN spiking neural network simulator [19] to learn digit recognition with the MNIST training set [27]. The Brian simulator enables spiking neural networks to be efficiently implemented on biologically-based neuron models. We used the same neuron and synapse models, network architecture, lateral inhibition, input encoding, and training and classification methods as Diehl and Cook in [22] using portions of their code provided at [20], with the following modifications to implement a sequentially learning SNN.

Implementing forced firing

In order to accomplish forced firing, the neuron membrane potential must be caused to surpass the firing threshold potential. This requirement can be satisfied either by sending additional excitatory synaptic signals to the relevant neuron to increase its membrane potential, or by decreasing its firing threshold. In our simulations, we temporarily reduce the relevant neuron's firing threshold. The lower threshold causes the neuron to spike more easily, strengthening the appropriate synaptic connections from the input signals and encoding the new information.

Spike count threshold for new neurons

In the setup used in [22], the output spike count was used to determine if the input spike train intensity should be increased. If fewer than five output spikes were registered, then it was determined that the network was having difficulty learning the current pattern. The input intensity was incremented iteratively until at least five output spikes were registered. We use spike count to also determine if there is poor recognition, indicating the need to add a new neuron. This threshold was established as when the input intensity was increased 5X without registering the five output spikes.

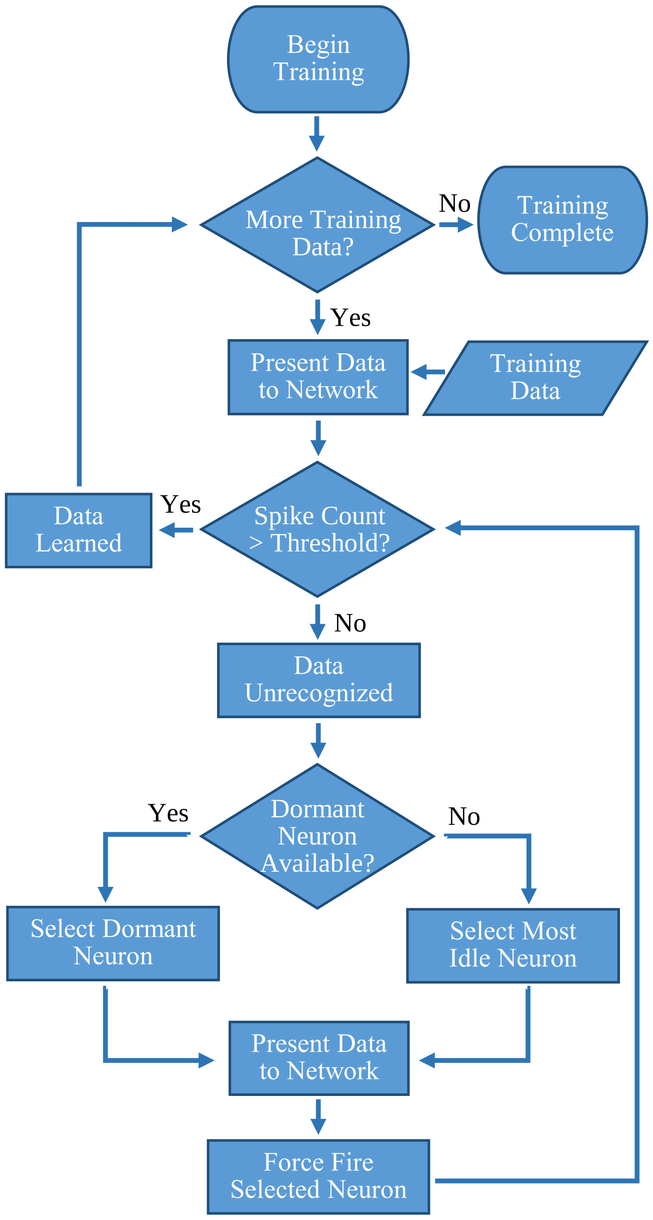


Figure 4.4. System-level approach to implement incremental STDP learning. (©2016 IEEE [14])

Output spike count is partially influenced by the pixel footprint of the patterns stored in synaptic weights. An input pattern with a larger pixel footprint can have a larger impact on output spike count because there are more synaptic connections in use. Because output spike count is an essential metric, this imbalanced preference was resolved using a weight normalization technique similar to that in [28].

Identifying idle neurons using spike count

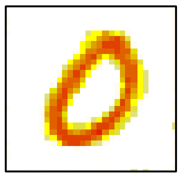
When there are no dormant neurons available, an idle neuron is identified and reassigned to learn the new information. As discussed in 4.3.4, the idle neurons can be identified by a low spike count. We expand this idea to improve accuracy by using a “win” count that is based on the spike count. Rather than count each spike for every output neuron that fires, we count how many times an output neuron fires more than the other output neurons. These are called “win” counts and can be implemented with simple hardware counters. For each input sample, the win count of the output neuron that fires most is incremented. The neuron with the smallest win count is deemed most idle. By counting wins, we also eliminate neurons that are less useful due to redundancy.

In addition, we establish a time decay for neurons that were recently added to the network, preventing them from being immediately overwritten (since their spike count resets when they are reassigned). This time allows new information the opportunity to establish its importance before becoming a candidate for deletion.

Homeostasis

Traditional homeostasis requires that input patterns be intermixed. However, iterative learning often presents input categories in groups, requiring that sections of the network be active at different times. Our method of forced firing further conflicts with homeostasis by giving preference to a specific neuron.

Our resolution to this conflict was to replace homeostasis with a combination of weight normalization (discussed previously) and complementary inhibitory input weights (discussed


(a) Excitatory synaptic weights from the input to an excitatory neuron.

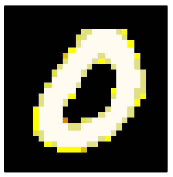

(b) Complimentary inhibitory weights to the same neuron.

Figure 4.5. Example weights from the input neurons to an excitatory neuron that has been trained for a specific pattern of the digit '0.' (©2016 IEEE [14])

next) to reduce a single neuron's dominance of all input fields while still encouraging variation between neurons.

Complementary inhibitory weights

For each excitatory synapse extending from the input neurons to the excitatory neurons, we implemented an inhibitory synapse with a complementary weight (see Figure 4.5 for an example). The inhibitory weights (w_{inh}) are calculated to nonlinearly compliment the excitatory weights (w_{exc}) as follows:

$$w_{\text{inh}} = \beta \cdot [1 - (w_{\text{exc}})^{\mu}] \tag{4.1}$$

where $\beta > 1$ and $\mu < 1$. The complementation is scaled by β to allow the inhibitory weights to have larger strength, but to only reach a high value when the corresponding excitatory weight is very small. The nonlinearity of μ allows the inhibitory weights to have less affect before the weights are sufficiently trained. The excitatory weights range from 0 to 1, while the inhibitory weights are reversely scaled from β to 0. The exact values for β and μ can depend on network size, tuning the variability of learned data. For a network of size 400, we used $\beta = 8$ and $\mu = 1/32$. An appropriate balance of excitatory and inhibitory weights can help prevent overfitting.

The concept of complementary inhibitory weights is based on the idea that some patterns are defined not only by where pixels are, but also by where they aren't. For example, a certain image for the digit '3' may have all the same pixels as an image for the digit '8' except for the pixels closing the loops on the left. In this example, the excitatory weights associated with the digit ‘3’ would be large where there are pixels and small where there is white space, while the inhibitory weights would be small where there are pixels and large where there is white space. The inhibitory weights prevent the neurons associated with the digit ‘3’ from firing when the digit ‘8’ is presented. Complimentary inhibitory weights allow forced firing to successfully enable sequential learning, as demonstrated next.

4.3.6 Preliminary results

Using the methodology described, we simulated an sequentially learning spiking neural network with 400 excitatory neurons. The data was presented sequentially, and no training image is re-shown after subsequent images are shown (i.e. no reinforcement). While repeating input patterns within a category is not prohibited by sequential learning, we avoid it to better illustrate the capability of the network.

The accuracy of the network was measured using a testing set of images from the MNIST data set which were not used during training. Figure 4.6 shows the accuracy measurements at each stage of the training process, which we explain below.

We began the training process by presenting only digit ‘0’ to the network. Figure 4.7a shows the network weights at that stage. Notice that the network activated 315 out of the 400 total neurons, leaving 85 still dormant. The darker weights correspond to those input patterns that were learned more intensely, indicating that patterns similar to those patterns were more common. Lighter weights indicate input patterns that were less frequent and are consequently better candidates for reassignment when the dormant neurons run out.

Figure 4.7b shows the network weights after subsequently presenting only the training images associated with digit ‘1.’ We note that much fewer neurons were required to learn the digit ‘1’ because it is a less complex pattern. Next, the training images for digit ‘2’ were presented, as shown in Figure 4.7c. It is important to see that the weights of several neurons which had learned the digits ‘0’ or ‘1’ were overwritten to learn digit ‘2’ because there were insufficient dormant neurons. The idlest neurons, by win count, were selected to be reassigned.

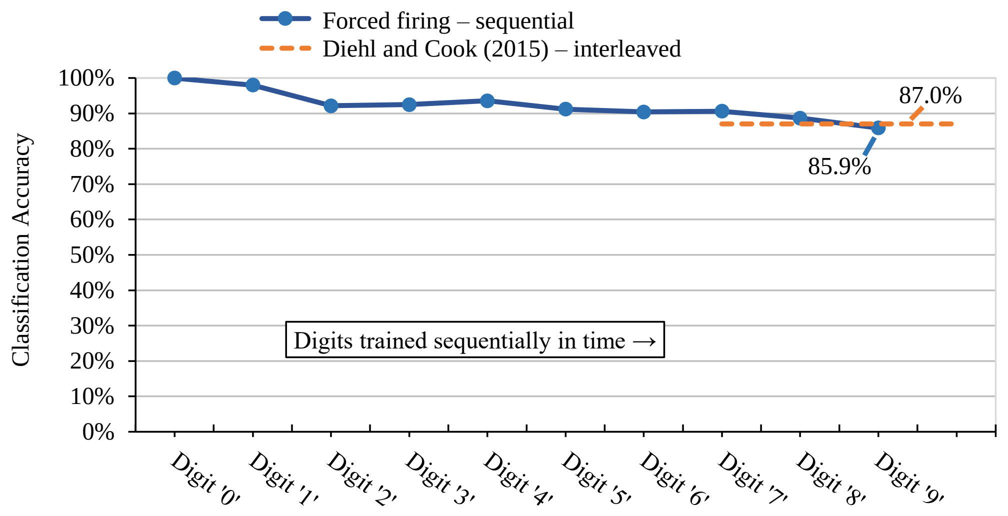
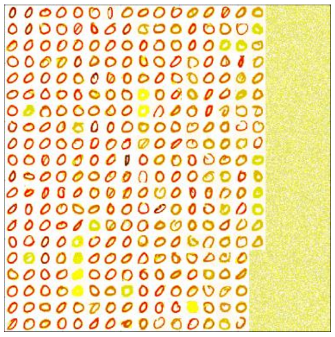
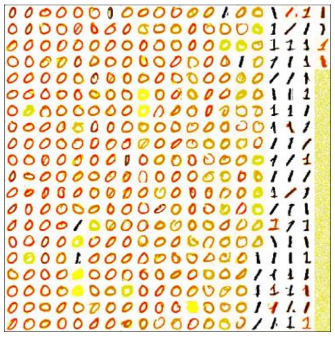


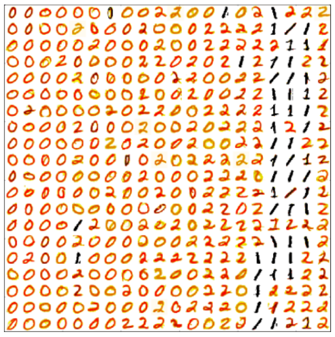
Figure 4.6. Graceful degradation of accuracy as new information is added to an incrementally learning SNN of size 400, compared to an equivalent SNN which is incapable of incremental learning and requires data reinforcement. (©2016 IEEE [14])

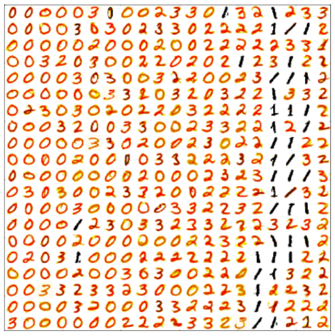
This process was continued in order for the remaining digits, one by one, until all digits '0' through '9' had been learned. Although the data was presented in groups, the training process is unsupervised, and the information may be learned in any order. The advantage of this network is that even though the old data is not reinforced, the network still successfully preserves substantial amounts of old information across data categories. Note how the final weights (see Figure 4.7j) show many image patterns for each digit (early or late). Rather than all of the old information being overwritten, less significant portions across the data categories were selectively reassigned, providing graceful adaptation.

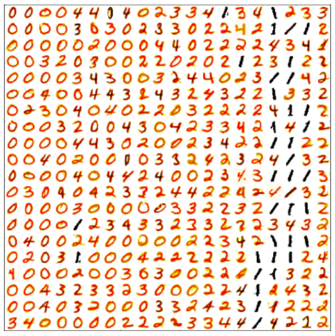
As mentioned, the trade-off for learning broader information across more categories with a limited network size is a gradual decrease in accuracy. A good comparison at this network size is Diehl and Cook's [22] SNN which achieves at most 87.0% accuracy for an SNN of size 400, with improved accuracy as the network size is increased. As with other traditional SNNs, their work requires that all input data be learned at once (i.e. input categories interleaved). The accuracy results for our sequentially learning SNN are comparable (refer again to Figure 4.6), even though the data is presented sequentially without any data reinforcement.

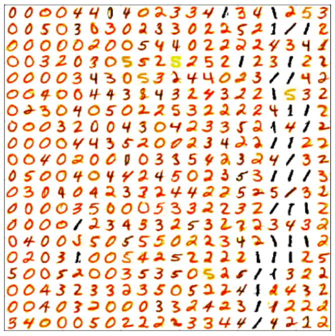

(a) After learning digit ‘0.’

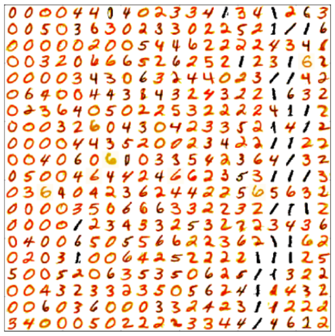

(b) After learning digit ‘1.’

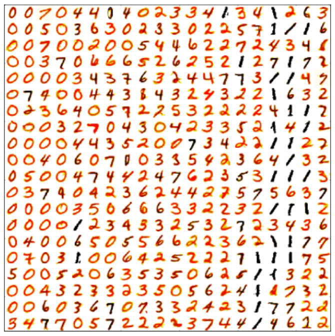

(c) After learning digit ‘2.’

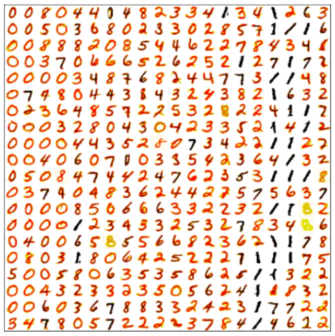

(d) After learning digit ‘3.’


(e) After learning digit ‘4.’


(f) After learning digit ‘5.’


(g) After learning digit ‘6.’


(h) After learning digit ‘7.’


(i) After learning digit ‘8.’

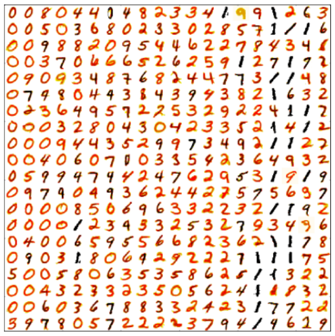

(j) After learning digit ‘9.’

Figure 4.7. Synaptic weights of a sequentially learning SNN using forced firing. (©2016 IEEE [14])

4.4 Dopaminergic Learning for "Controlled Forgetting"

This section presents a new learning paradigm, inspired by the dopamine signals in mammalian brains that non-uniformly, or heterogeneously modulate synaptic plasticity. We create Controlled Forgetting Networks (CFNs) that address the stability-plasticity dilemma with rapid/local learning from new information, rather than the traditional gradual/global approach to learning. Our approach allows fixed-size CFNs to successfully perform unsupervised learning of sequentially presented tasks without catastrophically forgetting older tasks.

The stability-plasticity dilemma can be addressed by allowing for dynamic, heterogeneously modulated plasticity. Consider the example of unsupervised clustering where neurons are trained to center on input clusters (see Figure 4.8). Temporarily making the synaptic weights of some neurons more plastic while keeping the weights of other neurons more rigid can allow for isolated adaptation by the plastic parameters while protecting the information associated with the rigid parameters. The challenge then becomes how to dynamically control the plasticity and for which parameters.

STDP embeds local, generalized representations of correlated inputs within the synaptic weights of individual neurons. Lateral inhibition between neurons, similar to the architecture in [22], creates competition that prevents multiple neurons from learning the same information. We seek to control the forgetting process by harnessing the segmentation of localized and distinct representations that are created by STDP with competition. Interference from novel information may be isolated by stimulating specific network elements to adapt to that information, protecting the remainder of the network from change. The forgetting caused by this interference may be minimized and controlled by targeting network elements associated with less useful information. We draw on inspiration from biology to heterogeneously modulate STDP learning to perform such isolated adaptation, creating Controlled Forgetting Networks (CFNs).

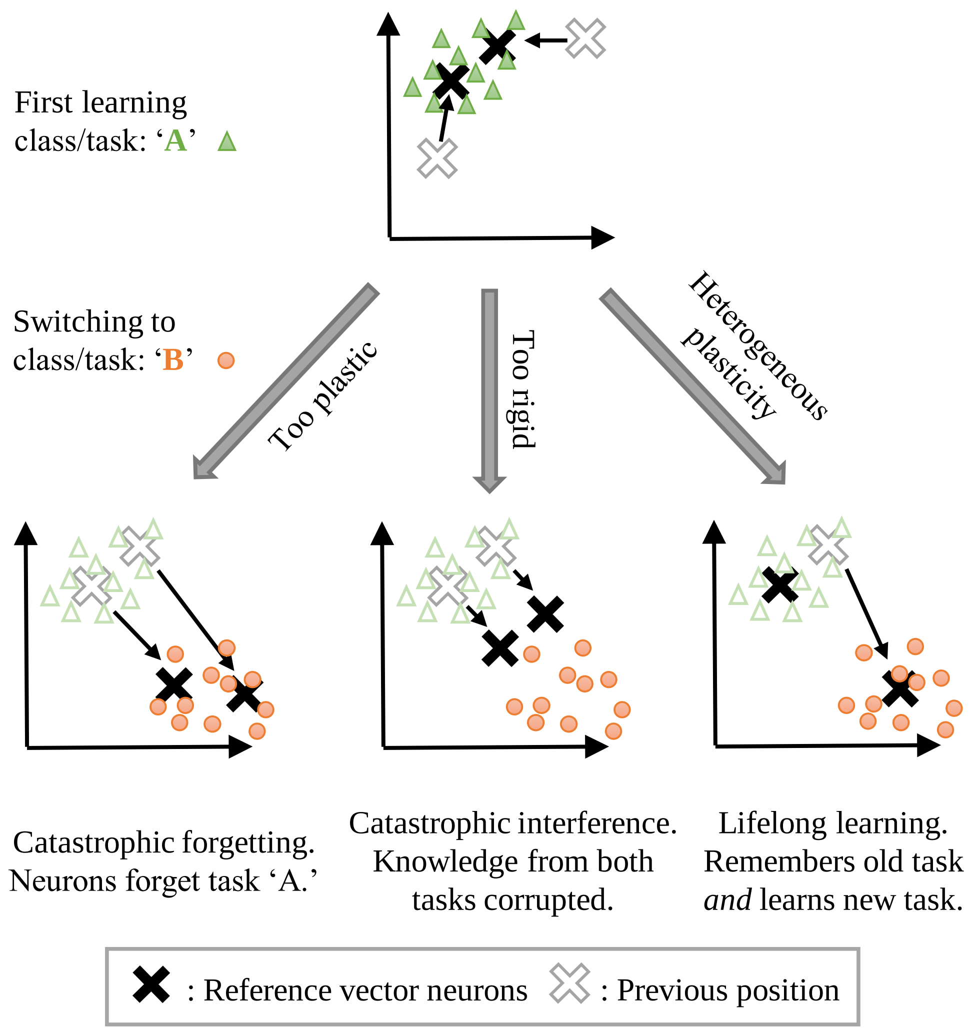


Figure 4.8. The stability-plasticity dilemma in unsupervised clustering. Lifelong learning is achieved with a strategic heterogeneous modulation of synaptic plasticity. (©2020 Allred and Roy [12])

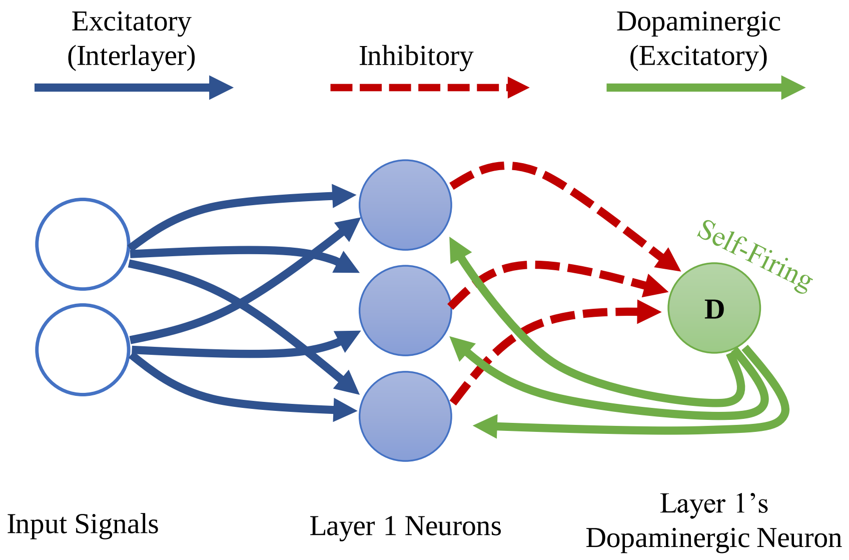


Figure 4.9. Single-layer CFN architecture. The dopaminergic neuron fires when the other neurons on its layer are *not* firing, often a sign of novel information. The firing of the dopaminergic neuron stimulates firing in the other neurons while temporarily enhancing plasticity. The stimulation signals from the dopaminergic neuron are weighted to provide heterogeneous, targeted stimulation. The other neurons within a layer each have additional laterally inhibitory connections for competition (not shown here). (©2020 Allred and Roy [12])

4.4.1 Biologically inspired dopaminergic plasticity modulation

Dopamine acts as a neuromodulator which gates synaptic plasticity. Dopamine signals are most commonly thought of as reward signals. In addition, though, dopamine releases are also associated with encountering novel data, which allows the brain to quickly adapt to new information [50]. We adopt this concept of novelty-induced plasticity modulation for our goal of local, rapid adaptation. We mimic a novelty-induced dopamine release by including a *dopaminergic neuron* at a given layer of a CFN (see Figure 4.9). We discuss how to identify novel information in an STDP-trained SNN, how the dopaminergic neuron is designed to fire under those conditions, and how the dopaminergic neuron modulates plasticity.

In STDP-trained SNNs, the weight vectors stabilize on the radial center of seen input clusters and are more likely to fire for inputs to which they are *angularly closer*–meaning inputs where the angle between the input vector and the weight vector are smaller for a given vector magnitude (see Sections 3.1.2 and 4.5.2). In other words, a sample from an unseen distribution will be less likely to induce firing than a sample from a learned distribution. Thus, when an input sample results in little-to-no firing activity at a given layer of neurons, we may assume that it contains information novel to that layer. Data in Section 4.6.3 validate this assumption, showing that a dopaminergic neuron designed to fire under these conditions is indeed triggered more frequently whenever the system switches to an unseen class and otherwise sees a reduction in triggered dopaminergic activity as the new class is learned over time.

We design the dopaminergic neuron with a resting potential higher than its firing potential, giving it a self-firing property. It is additionally suppressed via inhibitory connections from the other neurons in its layer so that it only spikes when they do not. This setup allows the dopaminergic neuron to fire only when novel information is detected.

When it fires, the dopaminergic neuron enhances plasticity by temporarily boosting the learning rate of the other neurons in its layer all the way to one while simultaneously stimulating firing in those other neurons via excitatory synaptic connections that we are calling *dopaminergic weights*. Because of the lateral inhibition discussed previously, once one of the stimulated neurons fires, it prevents or reduces the probability of the other neighboring neurons from firing. A neuron with a boosted learning rate then resets its learning rate the next time it fires or receives an inhibitory signal from a neighboring neuron, indicating that one of its neighbors has fired. Thus, while the dopamine signal is sent to many neurons, only the first neuron(s) to fire undergo the enhanced plasticity, creating heterogeneous plasticity and allowing the dopamine signal to perform an isolated targeting for local, rapid adaptation rather than global interference. Temporarily modulating the learning rate to the full value allows the first neuron that responds during a dopamine release to undergo a one-shot rapid learning of the current, novel sample and be “reassigned” without corruption from its old weight values. Then the learning rate is reset, allowing the representation to generalize with traditional, gradual weight changes. Figure 4.10 presents an example of the dopaminergic neuron in operation. The dopaminergic neuron fires for novel representations and does not fire if an input is similar to one already seen.

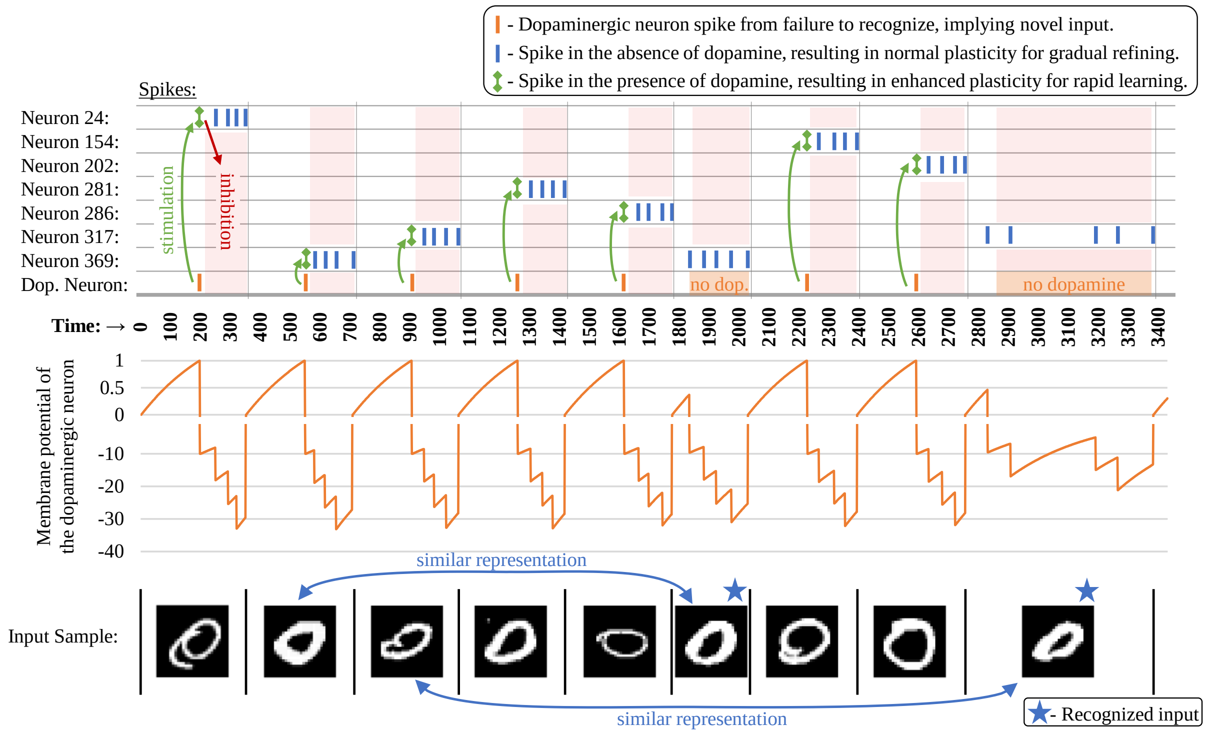


Figure 4.10. Example spiking activity (top) showing the interaction between the dopaminergic neuron and the other neurons during training for the first several samples (bottom) for a CFN of 400 neurons. Only neurons that fired during this small time interval are shown. The membrane potential of the dopaminergic neuron is shown (middle), demonstrating its self-firing property unless inhibited, with a firing threshold at 1. (A break is shown in the graph at zero indicating a different scale for the positive and negative values.) (©2020 Allred and Roy [12])

4.4.2 Targeted stimulation for controlled forgetting

We have addressed how to make the forgetting process rapid and local in order to reduce interference between old and new information. However, we must also control the specific locality of the forgetting so as to maintain high accuracy for previous tasks. A uniform stimulation would cause the neuron that is angularly closest to the input to fire first and adapt to the novel information, independent of how useful that neuron is for previous tasks. When the network is refining representations that have already been seen, adjusting the closest weight vector is appropriate to promote generalization. However, when novel data in presented in high-dimensional space, the closest neuron is more likely to be one that has already learned a distribution from a previous class rather than an unused neuron that is completely uncorrelated. Thus the stimulation must be controlled to avoid overwriting the most essential information from previous tasks. We provide this control by heterogeneously stimulating the other neurons to fire during the release of dopamine via the excitatory dopaminergic weights. Training these weights allows specific neurons to be targeted to undergo forgetting and re-learning.

To minimize accuracy degradation caused by forgetting, we would ideally like to forget outlier or stale information rather than commonly-used or recent information that may be essential for returning to previous tasks, applying knowledge from old tasks to new tasks, or generalizing the rapidly learned novel information. As a proxy for this categorization, we target neurons with low overall firing frequency (outliers) or less recent firing activity (stale). Considering firing age over firing frequency is a tunable parameter that controls how much if any preference should be given to more recent tasks. For the experiments in this paper, we consider all tasks as equally important no matter how recently seen, so we target neurons with low firing frequency.

For these purposes, we enact a simple local learning rule: a dopaminergic weight depresses each time its post-synaptic neuron fires. This rule causes a dopaminergic weight to be smaller when the post-synaptic neuron it is targeting has a higher firing rate, and vice versa. To maintain positive values, the depressions are proportional to the current value, causing an exponential decay. Otherwise, the dopaminergic weights experience a gradual potentiation.

Potentiation must occur to prevent the weights from tending toward zero with differences between weights too small to distinguish on implementations with finite precision. The rate of potentiation is irrelevant in our setup as long as it is the same for all dopaminergic weights in the layer, maintaining their relative values, because the dopaminergic neuron continues to send the dopaminergic signal until one of the other neurons in the layer fires. For the experiments in this work, we effect this potentiation by L^2-normalizing the fan-out vector of dopaminergic weights after a depression.

4.5 Dopaminergic Learning: Experimental Methodology

To evaluate the effectiveness of our proposed lifelong learning approach, we simulated CFNs on the MNIST dataset [27] on network sizes of 400, 900, 1600, 2500. 3600, 4900, and 6400 excitatory neurons, each for five different seeds. We compare the CFNs that have dopaminergic neurons to the same setups without dopaminergic neurons, both with and without homeostasis from adaptive thresholding. Also, as a control for the label assignment and evaluation process, we also test equivalently-sized networks with randomized weights, also averaged over five seeds, to compare with the accuracy achievable solely by the label assignment process.

4.5.1 Event-driven computation

As opposed to the time-stepped Brian simulator used previously in the forced firing method and also next the the adaptive synaptic plasticity method, for this method we employed pure event-driven computation in Matlab. Using exponential kernels, we treat spikes as inducing instantaneous voltage potentiations in the respective post-synaptic neuron membranes with exponential decay. As such, neurons only fire upon receiving an incoming spike and will not fire between incoming spikes, with the exception of the dopaminergic neurons which are handled separately. This allows us to emulate the networks using purely event-driven computation rather than breaking time into discrete time steps and updating neurons states at each time step. Because we encoded input spike trains as Poisson point processes, the time between spikes is an exponential random variable with $\lambda_\text{i} = \text{i}nput_\text{i}$.

Therefore, rather than incrementing time in fixed intervals, we calculate the time until the next input spike arrival and decay all the traces and membrane potentials according to that time interval before processing that input spike.

The dopaminergic neurons are an exception, as they fire in the *absence* of input spikes. Therefore, before processing an input spike, we first check to see if the dopaminergic neuron would have fired earlier, in which case, it is processed at its respective time interval first.

4.5.2 Modulating STDP

Due to the rapid learning that occurs in the presence of dopamine and the lack of traditional homeostatic threshold dynamics, we modify the STDP learning rule for improved stability, discussed previously in Section 3.2. Dopaminergic modulation of plasticity is implemented by dynamically changing the learning rate α. During normal operation, α is set to 0.01 for gradual generalizing refinement of the synaptic weights. When the dopaminergic neuron fires, α is temporarily set to one for the reasons discussed in Section 4.4.1.

Normalization

The MNIST dataset is a magnitude insensitive dataset, meaning that increasing or decreasing the intensity of a sample does not alter its class and that angular distance is more important than Euclidean distance. As given in (3.16), the mean pre-firing potential of a spiking neuron is proportional to the L^2-norm of its weight vector and also to the L^2-norm of the input rate vector. Although a larger mean pre-firing potential does not always correspond to a larger firing rate due to differing variances caused by the Hadamard square of the weight vector as shown in (3.18), the correlation between $E[V]$ and the firing rate sufficiently holds for datasets like MNIST with inputs of large enough dimensions and fairly comparable input sparsity between samples.

As such, for a given input and assuming equal weight vector magnitudes, the neuron that is angularly closest to the input will be more likely to fire, allowing for unsupervised Hebbian learning by training neurons on correlated inputs. Therefore, we L^2-normalize each neuron's weight vector, and for the same reason the input rate vectors are also L^2-normalized. Weight normalization has recently been shown to occur in biology [51] and may still be considered a localized function, as the processing can occur at the post-synaptic neuron to which all the weights in a given weight vector are directly connected.

4.5.3 Timing and time constants

As our evaluations and simulations are purely event-driven, the concept of discrete computational time steps is not applicable. Timing parameters are thus purely relative. Therefore, without loss of generality, the L^2-normalized input rate vectors were defined as having an L^2 rate magnitude of one spike per time unit, and all other timing values are relative to that. This subsection discusses the timing values used in the simulations.

Membrane decay time constant

According to Equation (3.16), the expected value of the membrane potential saturates in time according to $(1 - \mathrm{e}^{-t/\tau})$. A smaller τ results in a faster convergence to the steady state, or, equivalently, fewer input spikes to converge. E.g., in five time constants, the expected potential reaches over 99% of is steady-state value. However, using (3.18), the steady state standard deviation of the potential in proportion to the mean decreases as the decay rate increases:

$$\frac{\sqrt{Var(V)}}{E[V]} \propto \frac{1}{\sqrt{\tau}} \tag{4.2}$$

Thus, a larger membrane decay constant is better for proper discrimination between two differing inputs, but increases the number of computations. For the L^2-normalized MNIST dataset with 784 input dimensions, the angular distances between samples of differing classes are close enough to require at least 10 to 15 normalized time units for τ_{mem} in order to successfully establish a firing threshold that can discriminate between classes, and so τ_{mem} was set to 15 time units.

Time to recognize

A τ_{mem} of 15 still produces enough variance according to (3.18) that two to three time constants (between 30 and 45 time units) is on average sufficient time for the potential to rise above its steady-state mean. As mentioned earlier, we identify successful recognition of an input sample after registering five output spikes. Therefore, a total of 150 to 225 time units was generally sufficient to produce five sequential firing events in a reference vector neuron with a center close to the input.

In our simulations, we found little accuracy change by adjusting this hyperparameter within this range as long as the threshold voltage was appropriately tuned, so we fixed the time to recognize at 200 normalized time units for each simulation. We tuned the dopaminergic neuron to fire after those 200 time units unless it has been otherwise inhibited as discussed in Section 4.4. Specifically, with v_{reset} set to zero as a reference voltage, the dopaminergic neuron's firing threshold v_{th} was set to one with a resting voltage set higher at two, causing the membrane potential to rise until it fires. Setting its rising time constant to $\frac{200}{ln(2/1)}$ then meets this objective. Figure 4.10 shows the membrane potential of the dopaminergic neuron during simulation for the first several samples as an example of its operation over time.

We also set τ_{pre} to the same timing value of 200 time units to capture as much of the input train as possible because of the rapid one-shot dopaminergic learning of novel samples.

4.5.4 Determining v_{th} without adaptive thresholding

As discussed in 4.1.2, adaptive thresholding for homeostasis can interfere with lifelong learning on changing input distributions by temporally and spatially distributing the firing activity. Long-term adaptive thresholding may still be used with controlled forgetting if properly tuned, but our proposed method of enhanced plasticity and stimulated firing of infrequently-firing neurons is itself a form of deliberate, controlled homeostasis. Therefore, for a more accurate evaluation of the CFNs, we do not have the CFNs employ any adaptive thresholding–having static thresholds instead. With normalized weight vectors and input vectors, the larger the ratio $v_{th} : E[V(t)]$ the closer the input rate vector must be angularly to the weight vector to produce a given firing probability. Determining the proper v_{th}

without dynamic adaptation, therefore, depends on the tightness of the clustering in the dataset. With this context, we included v_{th} in our hyper-parameter search, discussed next.

4.5.5 Hyper-parameter sweep

SNNs are known to be highly sensitive to hyper-parameters, especially during unsupervised learning without error signals to provide dynamic corrections. We perform a small search in the hyper-parameter space, adjusting v_{th} and the number of training epochs. Results from this search are shown in Table 4.1, with hyperparameters resulting in the best accuracy highlighted for each size. Good machine learning practice requires that we choose the system parameters based only on the training set, so only training set accuracy results are shown here. Testing accuracy results are discussed later in the Results section. A similar hyper-parameter sweep was performed for the non-dopaminergic SNNs that also do not have homeostatic adaptive thresholding, as well as for the SNNs with randomized weight vectors.

Neuron firing thresholds, v_{th}

Based on the discussion above, v_{th} should be close to but slightly less than τ_{mem} in voltage units, which is set to 15 time units. For MNIST, we initially found that if v_{th} is much less than 13.5, a neuron may too likely fire for samples from other classes, while if v_{th} is much higher than 14.25, a neuron may not fire for very close samples, even different stochastic instances of the same sample. We therefore tested each setup with four different threshold values in this range: 13.5, 13.75, 14.0, and 14.25. Smaller networks require each individual neuron to capture a larger subset of input samples, generally requiring slightly lower thresholds than those in larger networks.

Number of training epochs

Larger networks can capture representations that are less common but still useful. As such, for larger networks more epochs within a class are required before proceeding to subsequent tasks in order to refine the less common representations. For smaller networks, on the other hand, more epochs may reinforce less useful outliers, making it more difficult to make room for subsequent tasks.

Comparison of $E[V(t)]$ at v_{th} with k-means clustering angular error.

We can compare the v_{th} values selected in the hyper-parameter search with the mean angular distance to a neuron's weight vector that would on average result in a membrane potential equal to that threshold. Performing a simple k-means clustering on the L^2-normalized

Table 4.1. Training accuracy results of hyper-parameter sweep for each network size across both v_{th} and number of training epochs per task. Highlighted cells are best configuration for each size. (©2020 Allred and Roy [12])

		# of Training Epochs per Task			
Neurons	v_{th}	1	5	10	20
400	13.50	87.63%	83.82%	79.23%	74.15%
	13.75	87.63%	84.07%	78.34%	75.18%
	14.00	86.64%	82.49%	77.11%	75.20%
	14.25	85.50%	83.59%	75.75%	75.06%
900	13.50	89.78%	91.07%	89.36%	85.31%
	13.75	89.11%	90.91%	89.81%	86.24%
	14.00	87.83%	91.47%	89.71%	84.75%
	14.25	86.82%	91.46%	89.94%	84.14%
1600	13.50	91.54%	92.42%	92.21%	91.35%
	13.75	91.24%	92.87%	92.48%	91.40%
	14.00	90.08%	93.34%	92.20%	91.30%
	14.25	88.48%	93.06%	92.85%	91.74%
2500	13.50	93.15%	93.62%	93.46%	93.13%
	13.75	92.82%	93.65%	94.06%	93.20%
	14.00	91.80%	93.37%	94.28%	93.53%
	14.25	90.06%	93.18%	94.04%	93.49%
3600	13.50	93.88%	94.09%	94.04%	93.80%
	13.75	93.90%	94.12%	94.48%	94.52%
	14.00	93.31%	94.02%	94.53%	94.52%
	14.25	92.33%	93.27%	94.40%	94.27%
4900	13.50	94.51%	94.91%	94.67%	94.77%
	13.75	95.00%	94.92%	94.82%	95.09%
	14.00	94.61%	94.85%	94.97%	95.21%
	14.25	93.59%	93.82%	94.54%	95.29%
6400	13.50	95.39%	95.25%	95.28%	95.25%
	13.75	95.42%	95.55%	95.39%	95.68%
	14.00	95.33%	95.59%	95.42%	95.79%
	14.25	94.79%	94.99%	95.21%	95.88%

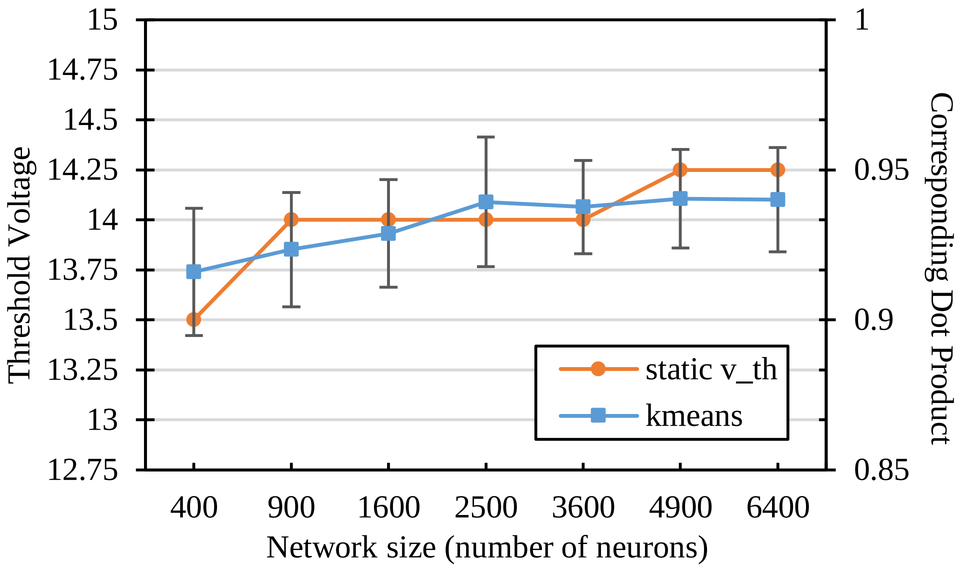


Figure 4.11. Comparison of the static v_{th} selected in the hyperparameter sweep with the corresponding dot product of the nearest training error in a kmeans network of the same size. The kmeans error bars represent two standard deviations over 100 trials each. (©2020 Allred and Roy [12])

MNIST dataset yields information on the relative desired scope of each reference vector, depending on the number of reference vector neurons. Figure 4.11 shows the dot product associated with the angular distance of the closest training sample / reference vector pair from differing classes for each network size after k-means clustering. The figure also shows the average membrane potential of a spiking neuron corresponding to these angles. For SNNs, neurons that are able to fire for samples that are further away than these angles are thus more likely to fire for samples of the wrong class. As the number of reference vector neurons increases, the portion of the input space per neuron decreases, improving accuracy by allowing each individual neuron to be more restrictive in its angular scope, which is relatively similar to those associated with the v_{th} values selected in the hyper-parameter sweep.

4.6 Dopaminergic Learning: Experimental Results

In this section, we present the results of simulating the CFNs and the non-dopamine comparison networks for the various sizes in both the interleaved classes scenario and the

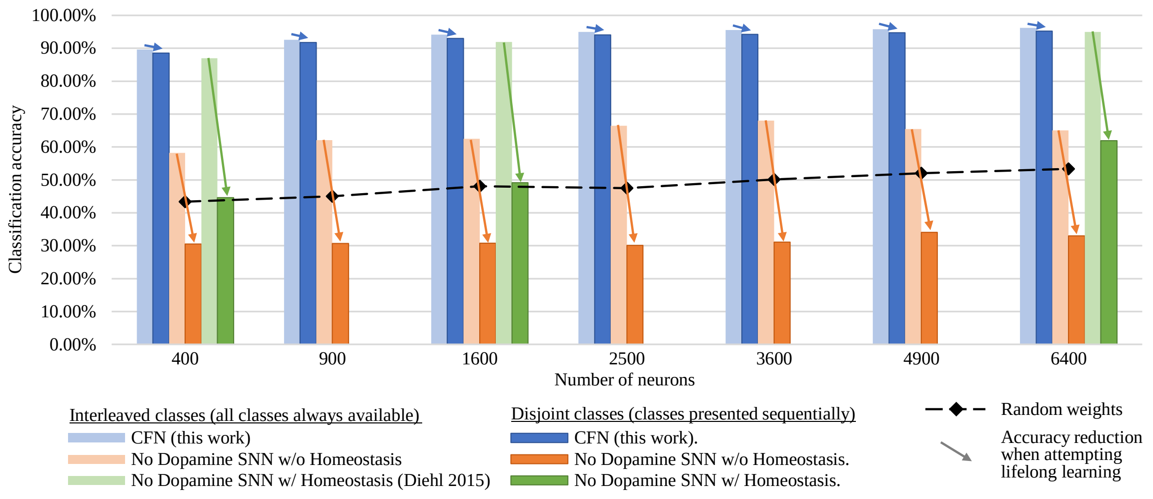


Figure 4.12. Final classification accuracy at various sizes of the CFNs compared to SNNs without dopamine. Accuracy is shown for both the interleaved class scenario and the disjoint class scenario, showing the resulting accuracy reduction by sequentializing the classes. CFNs show average over five seeds. (©2020 Allred and Roy [12])

fully disjoint classes scenario. We present both the combined accuracy and the per digit accuracy, with final results and (in the disjoint scenario) results throughout the attempted lifelong learning process.

4.6.1 Combined, across-task accuracy results

Figure 4.12 shows the final combined, across-task classification accuracy of the CFNs and comparison networks for both the interleaved scenario and the disjoint scenario for all network sizes. The comparison with [22] is provided for the network sizes for which results were published (400, 1600, and 6400). In the fully disjoint scenario, the 6400 CFN achieves on average **95.24%** classification accuracy across all digits, compared to 32.97% for a non-dopamine SNN without homeostasis, 61.95% accuracy for a non-dopamine SNN with homeostasis, and 53.30% accuracy for an SNN with random weights.

Figure 4.13 shows the combined, across-task accuracy over time for the CFNs in contrast to the comparison networks for network sizes of 1600 and 6400 neurons. CFN results for the other sizes are shown in Figure 4.14. The combined, across-task accuracy over time is

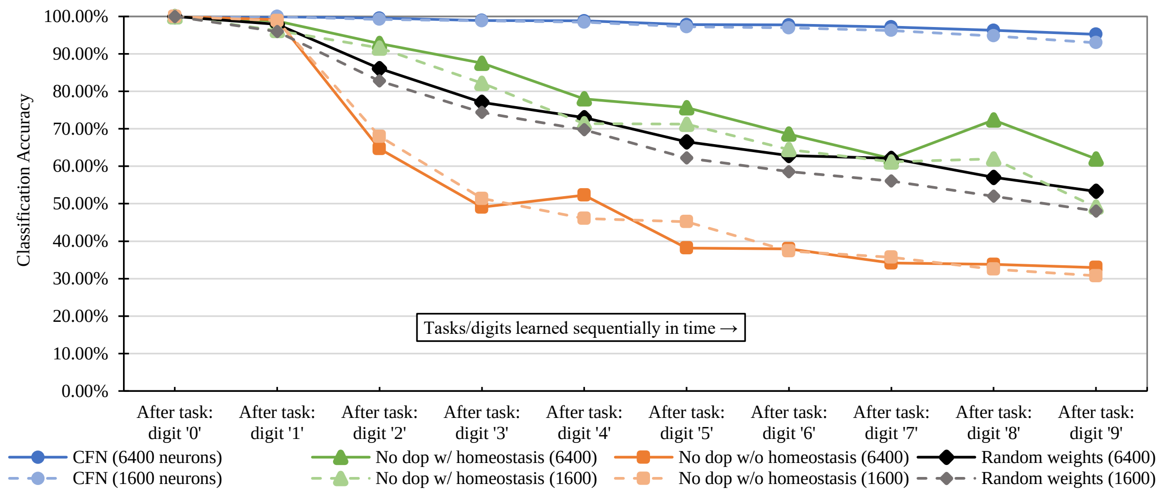


Figure 4.13. Classification accuracy over time at each stage of the learning process (i.e. after each new task/digit) in the disjoint scenario, comparing the proposed CFNs to SNNs without dopamine and to the randomized weight control. Accuracy is for all previous tasks, up to and including the current task. CFN results are averaged over five seeds. (©2020 Allred and Roy [12])

defined as classification accuracy on the portion of the testing set consisting of all previously-seen classes, up to and including the current task. For the 6400 size, the CFN incurs its largest accuracy drop at the last stage, adding digit '9,' dropping 1.06 percentage points. In comparison, at that size the non-dopamine SNN without homeostasis incurs a 34.41 percentage point drop when adding digit '2,' the non-dopamine SNN with homeostasis incurs a 10.41 percentage point drop adding digit '9,' and the SNN with random weights incurs an 11.82 percentage point drop adding digit '2.'

4.6.2 Per-digit accuracy results

Figure 4.15 shows the final accuracy of each individual task/digit by the end of the training process for 6400 neurons, comparing the distribution of accuracy across tasks for the CFNs in both the interleaved and disjoint scenarios, as well as with both the non-dopamine SNNs in the disjoint scenario and the randomized weights. In the disjoint scenario, the CFN's final worst performing class is digit '9' at 91.18% accuracy, which is also the worst performing class in the interleaved scenario at 93.60% accuracy. In comparison, for the

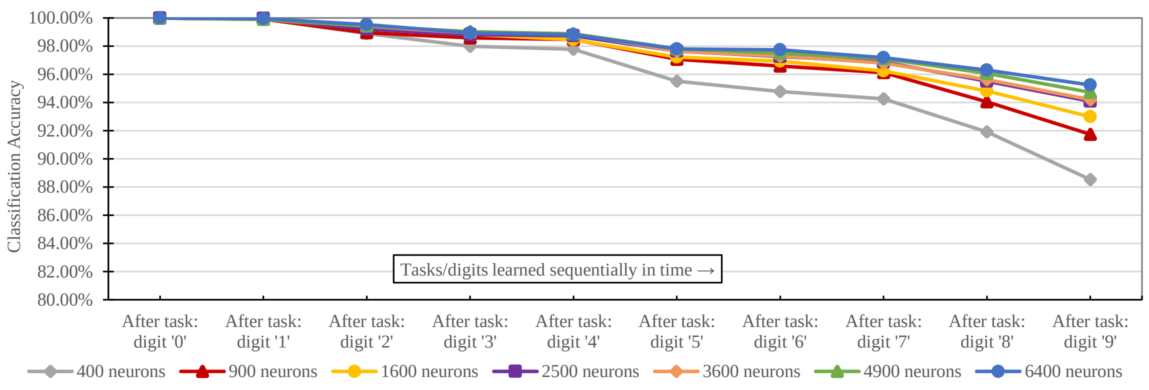


Figure 4.14. CFN classification accuracy over time as the number of tasks increases. Accuracy shown at each stage of the learning process (i.e. after each new task/digit) for CFNs of each size. (Five seeds.) (©2020 Allred and Roy [12])

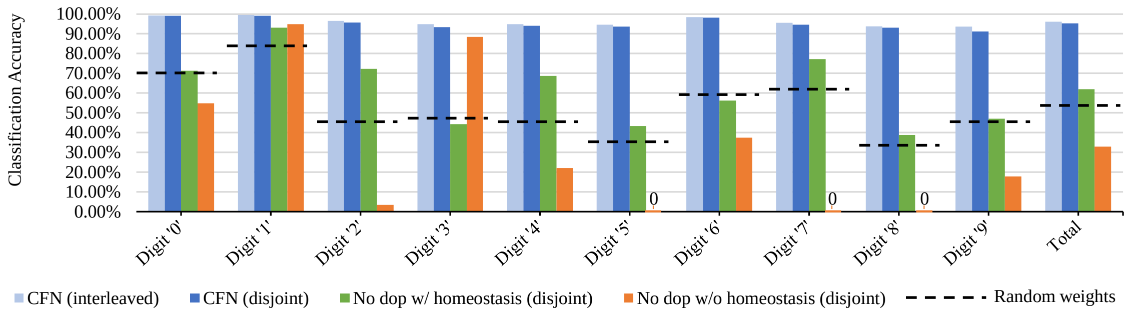


Figure 4.15. Final per-digit accuracy (size 6400), comparing interleaved CFN accuracy to the disjoint CFN accuracy. Also showing failure for individual digits in the disjoint scneario for SNNs without dopamine. (©2020 Allred and Roy [12])

other networks in the disjoint scenario, the final worst performing class is digit ‘8’ at 38.81% accuracy for the non-dopamine SNN with homeostasis; digits ‘5,’ ‘7,’ and ‘8’ tied at 0.00% accuracy for the non-dopamine SNN without homeostasis; and digit ‘8’ at 33.37% accuracy for the SNN with random weights.

Figure 4.16 shows the per-digit accuracy over time for each network of 6400 neurons. Per-digit false positives over time are provided in Figure 4.17. The CFN incurred its largest per-digit accuracy drop for digit ‘4’ after adding digit ‘9,’ decreasing 3.89 percentage points

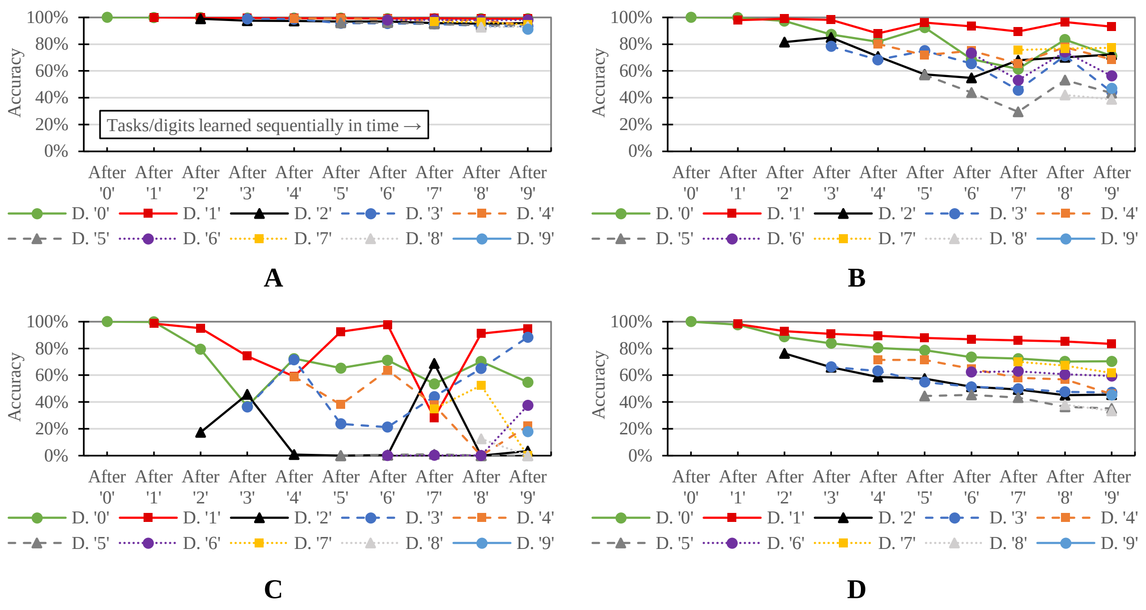


Figure 4.16. Per-task/digit classification accuracy as new tasks/digits are added over time for the following networks, all of size 6400: (a) the proposed CFN, (b) no dopamine SNN with homeostasis, (c) no dopamine SNN without homeostasis, and (d) SNN with random weights. (©2020 Allred and Roy [12])

for digit ‘4’ during that task change. In comparison, the non-dopamine SNN with homeostasis incurred a 23.07 percentage point drop for digit ‘4’ at that same transition; the non-dopamine SNN without homeostasis incurred a 69.52 percentage point drop for digit ‘1‘ after adding digit ‘7;’ and the SNN with random weights incurred a 10.98 percentage point drop in accuracy for digit ‘4’ when adding digit ‘9.’

4.6.3 CFN dopaminergic spiking activity during training

Figure 4.18 shows the dopaminergic spiking activity during the training process using CFNs on the disjoint MNIST dataset. Note how dopaminergic activity suddenly increases each time a task change occurs and novel data is presented, followed by a gradual decrease in dopaminergic activity as the CFN learns the new task. These activity statistics validate the assumption that low spiking activity in the non-dopaminergic neurons (which triggers the dopaminergic neuron) is a meaningful measure of novelty.

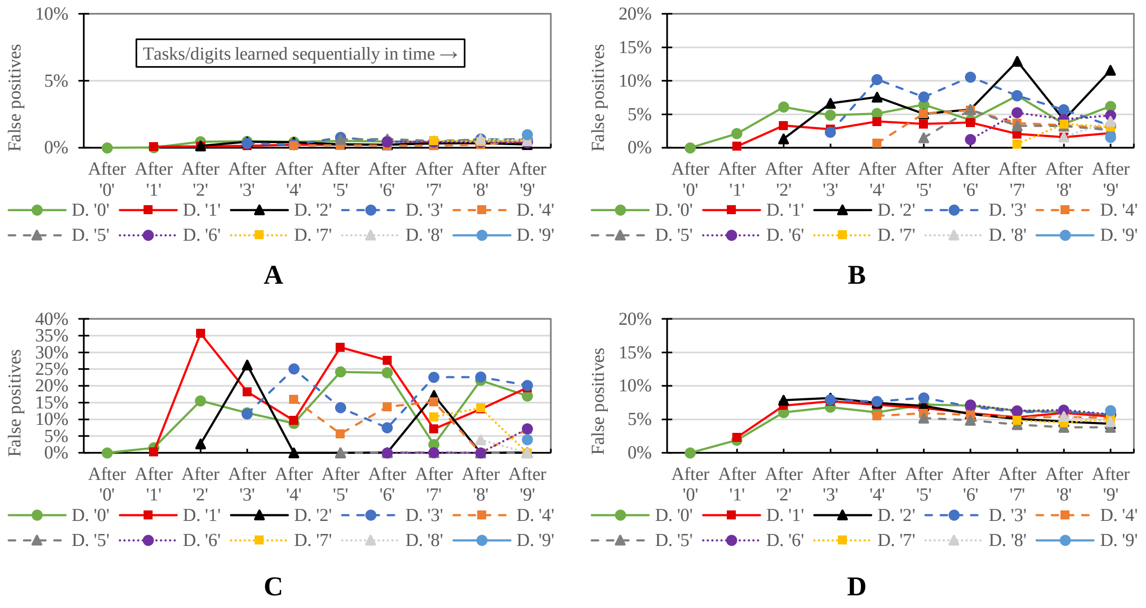


Figure 4.17. Per task/digit false positives as new tasks/digits are added over time for the following networks of size 6400 neurons: (a) the proposed CFN, (b) a no dopamine SNN with homeostasis (c) a no dopamine SNN without homeostasis, and (d) an SNN with random weights. (Note: vertical scales differ between charts because of the wide variation; grid lines remain consistent at 5% intervals.) (©2020 Allred and Roy [12])

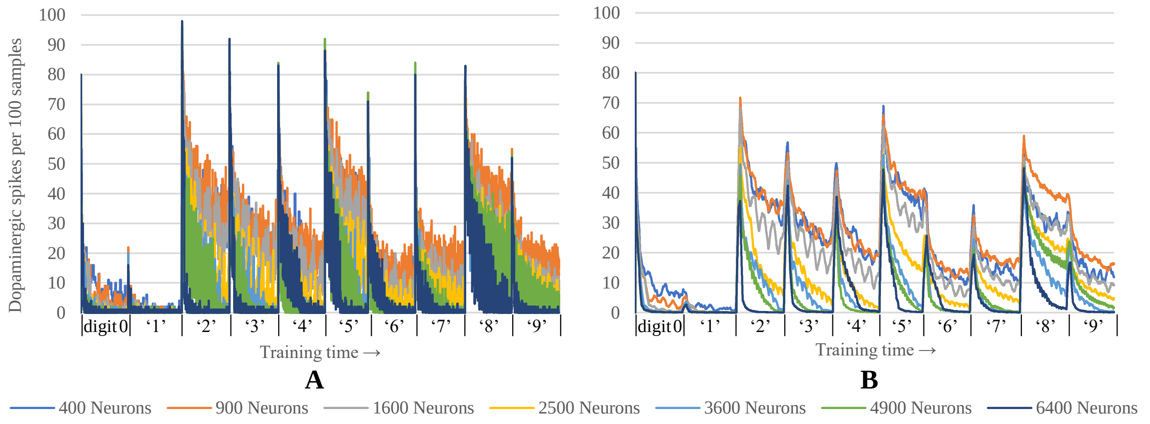


Figure 4.18. Dopaminergic spiking activity during training on the disjoint MNIST with CFNs. Values represent (a) dopaminergic spikes per 100 samples and (b) the same data smoothed using a running average over 100 bins. (©2020 Allred and Roy [12])

This expected variation in dopaminergic activity is most pronounced in the larger networks that have capacity to learn many different digit representations. For example, in the CFN with 6400 neurons the dopaminergic activity reaches near-zero levels by the end of learning each task. In the very small networks, the decay of dopaminergic activity is slower because a smaller network capacity means that even within a class there are more representations than capacity and thus a subset of less common but previously-seen representations continue to be viewed as "novel" since the network did not have capacity to ever permanently learn them.

Also note how different classes require different dopaminergic activity than others. For example with digit '1' there is very little required dopaminergic activity because there are fewer significantly different representations of that digit within the class. Once the CFN has learned these few representations, the remaining presentations are easily recognized, reducing the need for dopaminergic assistance.

4.6.4 CFN excitatory neuron activity during testing

Making a shallow network wider has only marginal improvements in accuracy. However, these neurons are still used, even if infrequently. Figure 4.19 shows the spiking activity distribution across neurons for the CFNs during testing of all digits at the end of learning in the disjoint scenario. For each size, all or almost all of the neurons experience firing activity during testing, with only 0.7% of the neurons in the large 6400 network having zero spikes. However, the "win" counts–the number of samples that each neuron was the highest-spiking neuron–indicate that only just over half of the 6400 neurons in the large CFN were ever a decisive neuron during testing.

There are indeed a few overfit representations, as indicated by the distorted tails in the spiking distributions. These are the result of rapidly learned novel data in the last class (digit '9') that didn't experience generalization but were not subsequently overwritten because there was no following task. However, since most of the non-decisive neurons do experience some spiking activity, we can assume that a majority of them are likely not overfit representations and may therefore be useful for a different testing set selection. Additionally,

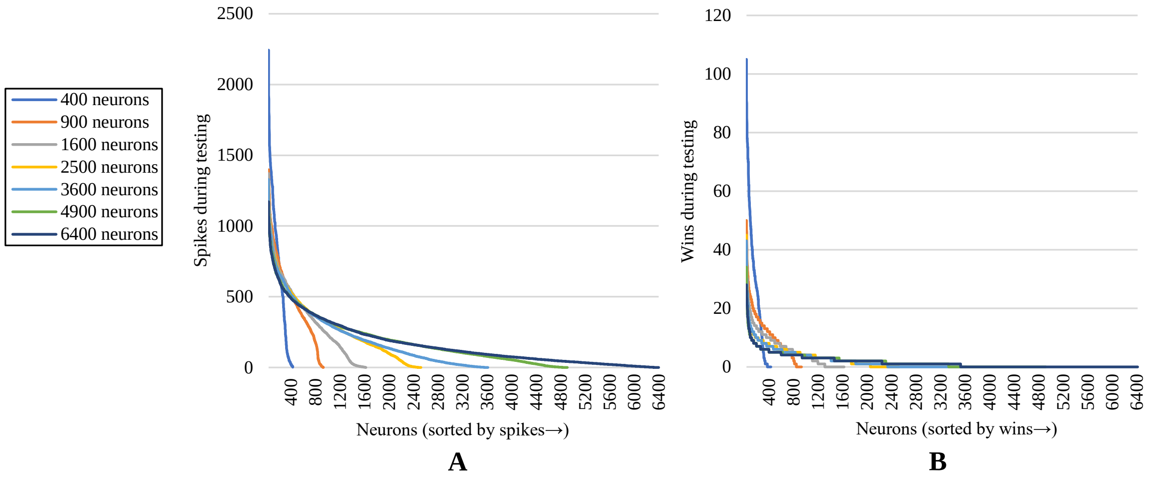


Figure 4.19. Activity distribution of neurons in the CFNs during testing on the disjoint MNIST. Values represent (a) spikes for each neuron during testing and (b) wins (number of samples for which each neuron was the highest spiking neuron). (©2020 Allred and Roy [12])

this additional space may be viewed as a “scratchpad” or working memory for temporarily storing novel inputs until it is determined if they will be kept.

4.6.5 Discussion

In this section, using a qualitative analysis we discuss reasons why the non-dopamine SNNs failed at lifelong learning in the disjoint scenario and how the CFNs avoided those failures. We also discuss the expected sequential penalty and graceful degradation of accuracy.

A qualitative analysis

In these fully-connected one-layer SNNs, each neuron’s weight vector can be viewed as a reference vector that captures a specific input representation, ideally successfully generalized. As such, we may qualitatively observe the success of dopaminergic learning over time by viewing these representations. For a better visual demonstration of the disjoint scenario, we show the weights of the networks for the first four digits ‘0’ through ‘3’ in Figure 4.20, with 100 neurons arranged in a 10x10 grid.

Note that in the CFN case (Figure 4.20(a)) there are two very distinct categories of representations. The digit representations that appear to have a more consistent pixel intensity and a more consistent line width and curvature are generalized representations refined by many similar samples in a cluster. On the other hand, the digit representations that appear less defined and with more irregularity in pixel intensity are outlier representations from only one or a few samples. Notice that the digit representations that are preserved from one task to another are the useful generalizations rather than the outliers, which on the other hand are the first to be overwritten when space for a new task is required. In addition, the representations that are preserved from previous tasks experience very little and infrequent corruption during later learning stages. The dopamine signals are able to successfully replace old information with new information without interference and while maintaining accuracy because of the targeted localization.

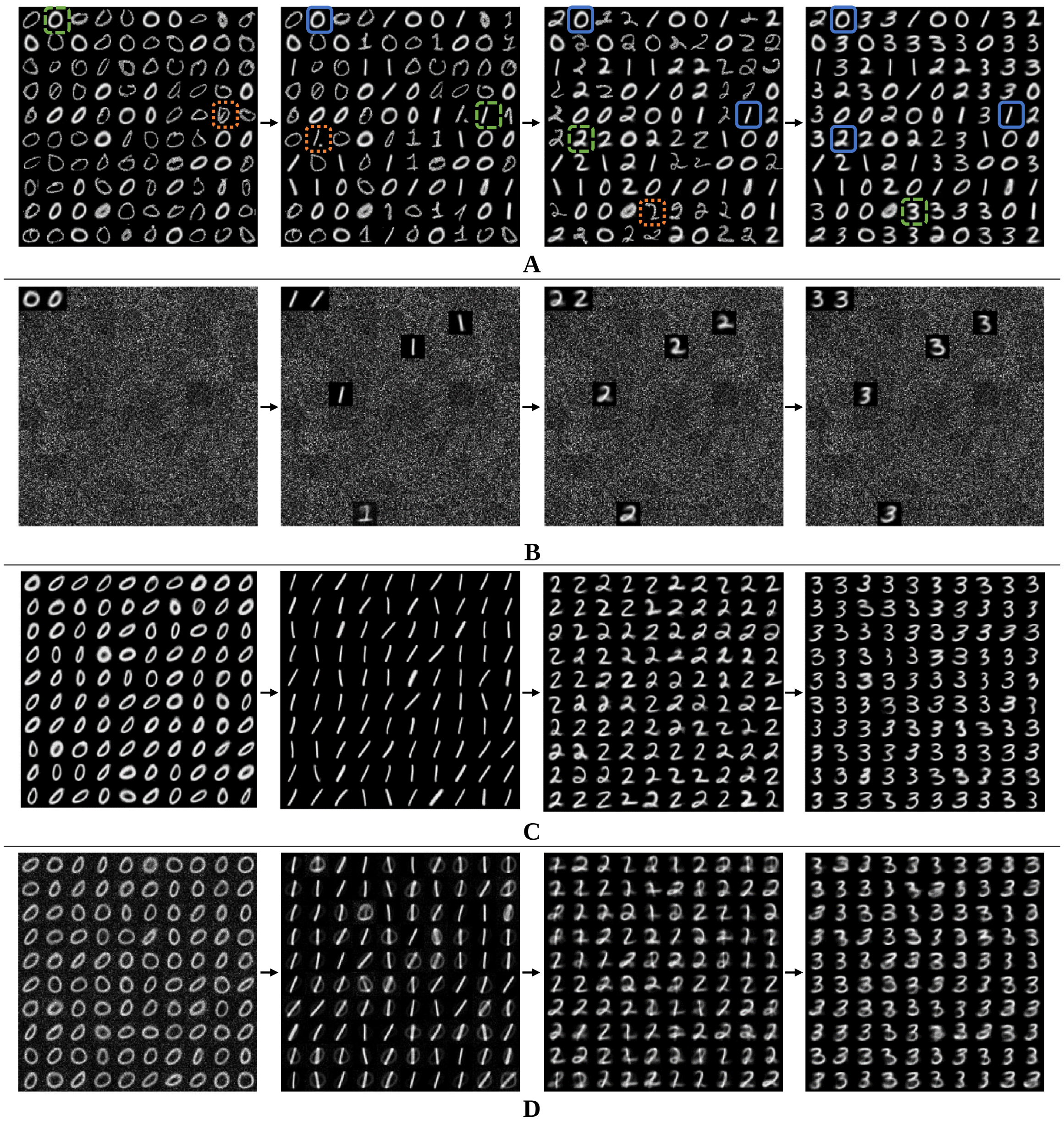


Figure 4.20. Grid view of the weight vectors over time, showing the first four digits, learning ‘0’ through ‘3’ for (a) the proposed CFN, (b) a non-dopamine SNN without homeostasis, and (c) a non-dopamine SNN with homeostasis, each with 400 neurons, although only the 100 top-firing neurons are shown for space. Digits highlighted in dashed green are examples of successfully learned generalized representations. Digits highlighted in dotted orange are examples of outlier representations. Digits highlighted in solid blue are examples of representations preserved from previous tasks. Also shown is (d) another non-dopamine SNN with homeostasis, but with reduced plasticity, showing catastrophic interference between classes causing corruption. (©2020 Allred and Roy [12])

In contrast, we can visually see the failure of the non-dopamine SNNs in the disjoint scenario. In the network without homeostasis (Figure 4.20(b)) we see that only a few neurons experienced any learning. Without homeostasis the neurons that fired first migrated closer to the input distributions and dominated the firing activity. Even when the input distribution changed between tasks, the already used neurons were closer to the new distributions than the unused neurons with random weight vectors. Continuing the reuse the same neurons caused the SNN to overwrite and forget previous tasks.

Next, in the network with homeostatic adaptive thresholding (Figure 4.20(c)), we see a better use of network resources from the distributed firing activity. But without targeted dopaminergic modulation homeostasis distributes the learning for a new task over all the neurons previously used in earlier tasks. Even when the learning per-digit is reduced (Figure 4.20(d)), the activity for the new tasks are still globally distributed by the adaptive thresholding, causing corruption between tasks.

The CFNs with dopaminergic learning avoid globally distributing firing activity during a single task by not having traditional homeostatic adaptive thresholding. In addition, the CFNs avoid continuing to reuse the same neurons by proactively identifying novel data and targeting specific neurons to learn the novel data, preserving essential information from previous tasks.

We note that for the failed networks where older classes are entirely overwritten by new classes, the networks still report some, albeit poor, accuracy for the forgotten tasks. This is because the varied intra-class distributions can still be somewhat useful at differentiating inter-class distributions. For this purpose, the accuracy comparisons to the SNNs with random weights are essential at identifying catastrophic forgetting, indicating that around 40-50% is a failure baseline for unsupervised learning using SNNs of these sizes on the MNIST dataset.

The expected "sequential penalty"

We see that the CFNs in the disjoint scenario perform on par with the interleaved scenario, averaging only a 1.04% accuracy reduction across all sizes. This penalty is expected due to sequentializing the tasks. In fact, such a penalty may be impossible to completely avoid, as the interleaved scenario provides more information to the network throughout training by providing all distributions up front, whereas the disjoint scenario never provides an opportunity to temporally overlap learning of different distributions. Even so, the sequential penalty for the CFNs is minimal, and may be acceptable given the system's avoidance of catastrophic failure in the disjoint scenario. In fact, even with this penalty, the 6400 neuron CFN achieves a respectable 95.24% test accuracy after lifelong learning, which we believe is the best unsupervised accuracy ever achieved by a fixed-size, single-layer SNN on a completely disjoint MNIST dataset. The CFNs in the disjoint scenario even outperform [22] in all cases for which they provide results, even though that work is in the interleaved scenario.

Graceful degradation instead of catastrophic forgetting

Controlled forgetting allows the network to gracefully degrade its accuracy in exchange for the ability to learn new tasks with limited resources, rather than failing. The true success of a lifelong learning system is shown not just by the final accuracy, but also by its performance throughout the training process and across training tasks. Notice how in Figure 4.15 while the system expectedly performs better for some tasks rather than others, there is no single task for which the system fails; i.e., the sequential penalty is spread between tasks. In fact, the lifelong system performs best at the same tasks (digits '0,' '1,' and '6') and worst at the same tasks (digits '8' and '9') that the offline/non-lifelong system does.

We believe that this type of approach with modulated plasticity and targeted stimulation can be useful for allowing online systems to gracefully adapt to changing environments rather than failing to adapt or requiring frequent offline retraining.

4.7 Additional Work: Adaptive Synaptic Plasticity

The previous proposed approach took inspiration from the biological brain in stimulating spiking activity and enhancing plasticity to *sensitize* the network to novel information. In that process, information was only forgotten when the resources were needed for other information. Complimentary to that approach is the Adaptive Synaptic Plasticity (ASP) approach presented in this section, extracted from my contributions to [15] (©2018 IEEE[4]), which takes inspiration from the gradual, long-term adaptation and stabilization of synaptic plasticity in the biological brain to *habituate* the network to important information, letting less-important information be purposely and *preemptively* forgotten to allow for adaptation without corruption.

Simply speaking, *habituation* is a decreasing response to a repeated stimulus and has been observed as a learning mechanism in even the simplest life forms. In a dynamic environment, habituation can stabilize long-term memory of frequently encountered information and enable a short-term response to novelty. In the material field, this form of habituation-based plasticity has been recently demonstrated in a perovskite quantum system via dynamic modulation of electron localization [52].

In mimicking this habituation mechanism, there are two goals: (1) increase the decay or “forgetting” of synaptic weights that are associated with less important information, i.e. stale or outlier information (Section 4.7.1); and (2) reduce the relative plasticity for synaptic weights that are associated with important information, i.e. information encountered recently and/or frequently (Section 4.7.2).

[4]↑In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Purdue’s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

4.7.1 Useful forgetting

In this work, forgetting is enabled via a continuous, gradual weight decay mechanism. Two forms are explored, an exponential decay and a linear decay:

$$\frac{dw}{dt} = -\frac{w}{\tau_{leak}} \cdot c_{exp} \tag{4.3}$$

$$\frac{dw}{dt} = -\frac{1}{\tau_{leak}} \cdot c_{lin} \tag{4.4}$$

where c_{exp} and c_{lin} are constants for each form and together with τ_{leak} create the decay rate at which the weight value "leaks" toward zero. Weights are allowed to be positive or negative and zero is the median between w_{max} and w_{min}. In a high-dimensional space, such as with MNIST [27], visual information is encoded in patterns that express themselves as input vectors with a Manhattan distance further from the mean than a randomly chosen vector in that space, as dictated by the curse of dimensionality. As such, decaying toward the median weight value represents an erasure or forgetting of the associated information.

These decay mechanisms act similar to an extension of the $offset$ term discussed in Sections 2.4.2 and 3.2.1 for the linear and exponential forms, respectively, and act to allow a weight vector to remove traces of its previous positions as it learns a new position. Without any such decaying, catastrophic interference occurs as information encoded in the weight vectors becomes corrupted with outliers or multiple distributions.

This work modifies this weight decay of the learning process with habituation to provide more useful forgetting. To accomplish this, the τ_{leak} value is adapted dynamically to the firing characteristics of the post-synaptic neuron, tying the value exponentially to the homeostatically-adapted threshold (see Section 2.4.1) and linearly to an exponential trace of the firing activity of its post-synaptic neuron. In effect, τ_{leak} is larger for synaptic weights whose post-synaptic neuron has been firing more recently or more frequently, slowing down the decay for that potentially more important information.

When a neuron is not firing, this adaptive weight decay is the only process affecting its fan-in weights. These time windows are referred to hereafter as the *decay phase*.

4.7.2 Adaptive plasticity

The other component of this work occurs in the *recovery phase* in which weight values are potentiated or depressed away from the resting weight mean with Hebbian learning using the one-sided STDP learning rule to encode generalized information in the weight vectors. Two adjustments to this STDP rule apply habituation-based mechanisms to adapt the plasticity for a dynamic environment.

First, the learning rate, α, is inversely scaled by the trace of the post-synaptic firing activity. As such, parameters that are used more frequently and/or more recently become more rigid, keeping and protecting useful information even when the environment changes. Conversely, parameters that have not been used very often or after an extended amount of time become more plastic, providing parametric resources for new information.

Second, the $offset$ term in the STDP rule is increased with an additional term which is exponentially decayed to the traces of the pre-synaptic firing activity, allowing for larger depressions for synaptic weights from outlier or uncorrelated pre-synaptic activity.

4.7.3 Experimental results

Using the Brian spiking simulator [19] and the architectural and unsupervised learning setup of [22] found at [20] on the MNIST dataset [27], this ASP method was tested for its resilience against catastrophic forgetting in a changing environment. First, a test case is presented to small networks with a single class addition, followed by a more extensive test on a fully disjoint dataset with larger networks.

Offline pre-training with subsequent scenario change

The exponential form of ASP is compared to an SNN with traditional one-sided STDP, each with 100 excitatory neurons. Each network is initially pre-trained to the same state using digits '0' through '8' with uniformly-distributed access to all these classes throughout pre-training. Then, to represent an environmental change, digit '9' is presented for online learning without providing any of the previous classes for data reinforcement.

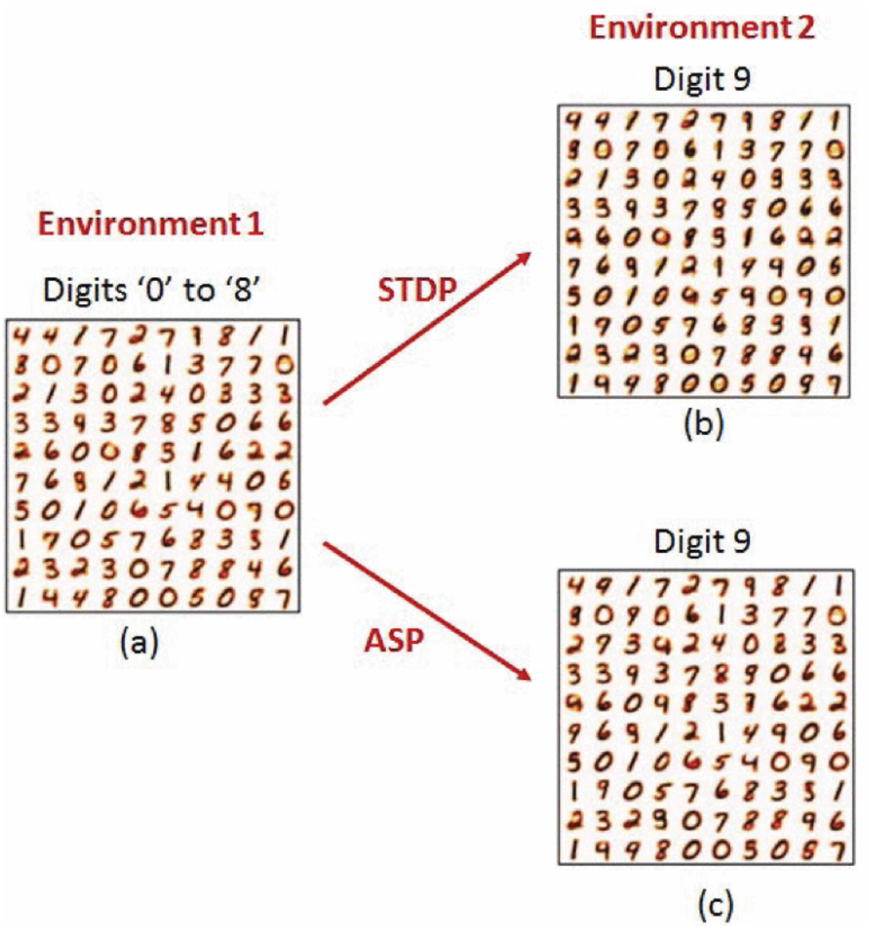


Figure 4.21. SNN weight vectors starting with (a) a pre-trained network using digits '0' through '8', and then subsequently trained only with digit '9' for both (b) traditional STDP and (c) exponential ASP. (©2018 IEEE [15])

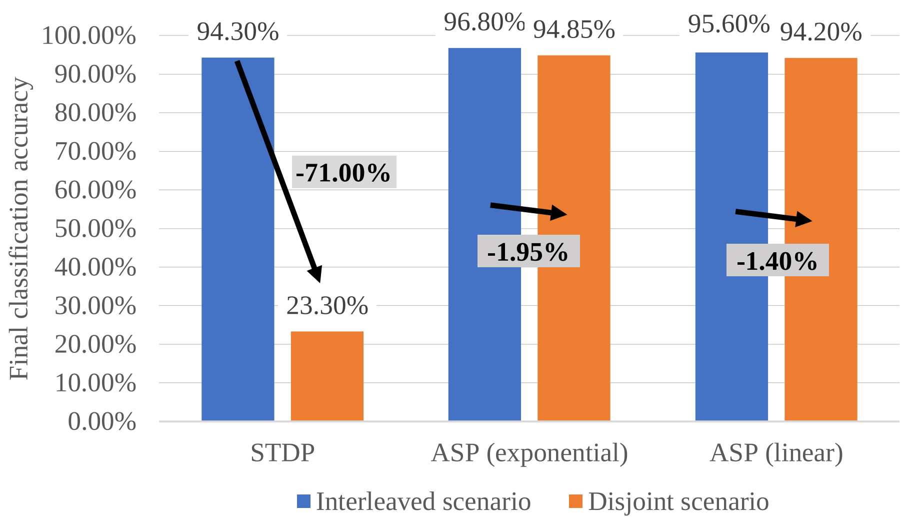


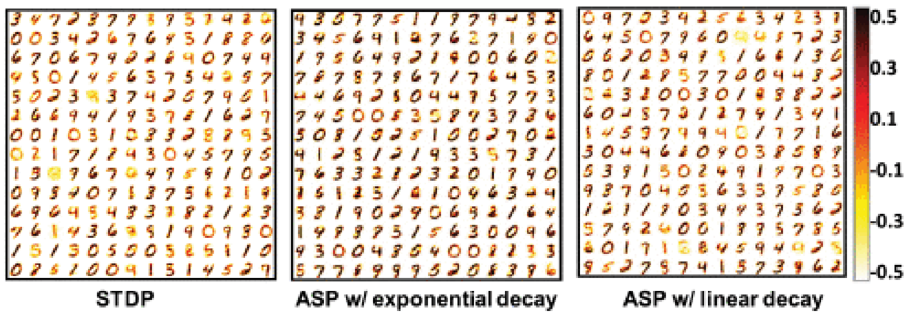
Figure 4.22. Final accuracy results of ASP versus traditional STDP after learning all 10 digits, showing both the interleaved scenario and the disjoint scenario, highlighting the accuracy drop from sequentializing.

Figure 4.21 shows the resulting weight changes, with the added class significantly corrupting most reference vectors in the traditional STDP setup. On the other hand, with ASP the adaptations to the new class are isolated and significant corruption is avoided. Final classification accuracy across all ten classes fell down to 62.8% accuracy for STDP, compared to 73.4% for ASP. Next, network sizes were increased for better baseline accuracy and the dynamic environment was extended.

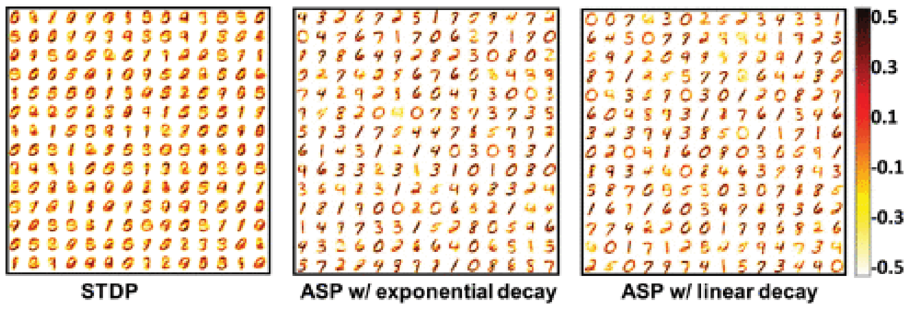
Lifelong learning (sequential classes)

A more extensive test of lifelong learning was performed by bypassing any pre-training and instead presenting all digit classes sequentially (the *disjoint* scenario). The final results of this scenario are compared to the traditional *interleaved* scenario with all classes distributed throughout training.

Using networks of 6400 excitatory neurons three SNNs were tested, one with traditional STDP, one with exponential ASP, and one with linear ASP. Final classification accuracy results are shown in Figure 4.22. While the STDP SNN catastrophically fails, the ASP



(a) Interleaved scenario.



(b) Disjoint scenario.

Figure 4.23. The final synaptic weights of ASP versus traditional STDP after learning all 10 digits, either in (a) the interleaved scenario, or (b) the disjoint scenario. (©2018 IEEE [15])

SNNs maintain high accuracy even in the disjoint scenario, demonstrating successful lifelong learning. In fact, the ASP SNNs exhibit a less than 2% drop in accuracy by sequentializing the classes, with the exponential ASP SNN achieving 94.85% accuracy in the harder disjoint scenario. Additionally, even in the interleaved scenario the ASP SNNs outperform traditional STDP. While the environment remains unchanging in the interleaved scenario, the proposed process of learning to forget helps remove outliers to improve generalizations. This can be seen in the final synaptic weights shown in Figure 4.23.

Similar to how this method of habituation was correlated with possible emerging devices demonstrating such properties, proposed future work for this lifelong learning body of work include demonstrating lifelong learning with a new set of potential device characteristics that combine both habituation-based learning and the previously discussed sensitization-based dopaminergic learning.

4.8 Potential Relationship with Observed NiO Behavior

The dopaminergic learning discussed and employed in Sections 4.4, 4.5, and 4.6 includes a stimulated rapid increase in synaptic plasticity. This functionality is sometimes called *sensitization*, in which the system is rapidly more sensitive to a new stimulus. Sensitization is complimentary to *habituation*, as discussed in Section 4.7, which is a reduced response to stimuli that the system has become accustomed to. While habituation allows the system to be stable, sensitization allows for adaptation. This combination, which has been observed in biological species, allows for non-associative learning and can help mitigate catastrophic forgetting.

4.8.1 Habituation and sensitization behavior of NiO

In [16], both habituation and the ability to trigger sensitization is observed in the Mott insulator NiO with certain stimuli. When exposed to H_2, the electrical resistance of NiO is increased and when the stimulus is removed, the resistance returns to its previous state. However, under repeated exposures, that response habituates, and the change in resistivity is reduced for the later exposures.

Then, when the stimulus O_3 is presented for a short time, the next subsequent response to H_2 is sensitized, regaining its large increase in resistivity as if it had not previously been habituation to that stimulus. Further exposures to H_2 then again re-habituate the material to that stimulus.

4.8.2 Potential neuromorphic application

The observed behaviour of the NiO material is still relatively far from device development and has several obstacles, including the stimuli as a gas and the time scale of the observed behavior (with exposure increments measured in minutes). However, the connection between the behavior and the mentioned learning rules is worth exploring. The resistivity change of the material could be used to indicate the plasticity of an associated learning parameter. The habituating exposures could occur with each online use of the associated parameter, reducing its plasticity the more it is used. Then, when novel information in encountered, the sensitizing stimulus could be presented to a targeted set of parameters which contain the oldest or least-useful information, resetting their plasticity and response to the novel stimulus, similar to the dopaminergic learning presented previously.

To help make this connection in [16] (from which the remainder of this section is taken[5]), I provided the following example simulations that illustrate the benefit of employing both habituation and sensitization in the local learning rules of neurons in a dynamic environment (see Fig. 4.24 and Fig. 4.25).

4.8.3 Task description

The task we explore is an unsupervised clustering task similar to k-means clustering [53] but performed online, sample by sample. In a layer of reference vector neurons in an Artificial Neural Network (ANN), each neuron produces a higher activation value for inputs that are angularly closer to that neuron's fan-in weight vector (i.e. using cosine distance). With competition between neurons, often implemented via lateral inhibition, only the angularly closest neuron responds sufficiently to a given input, assuming L^2-normalization of neuron weight vectors. An individual neuron, then, can dominate a Voronoi partition of an $(n-1)$-sphere in the n-dimensional input space and may be representative of an input distribution in that partition, generally centered on the weight vector. Increasing or decreasing the L^2-norm of a given neuron's weight vector with respect to the other neurons will expand or shrink its Voronoi partition, respectively.

Hebbian learning can successfully perform such unsupervised clustering tasks. For a given input sample, this correlation-based learning occurs by adjusting the weight vector of the "winning" neuron which is closest to the current input vector toward the direction of that input vector, gradually centering on a particular input distribution. For example, Spike Timing Dependent Plasticity (STDP) [54] has been shown to perform such Hebbian learning in shallow Spiking Neural Networks (SNNs) [22]. In deeper SNNs, this type of learning is

[5]↑At the time this dissertation was submitted, this work was accepted but not yet published. My portion of this work is included here with permission. Further changes may be made during the publication process, and this version should not be considered as the final version.

useful for layer-wise training in a convolutional neural network [55] and may also be useful for online learning of the final readout layer of a pre-trained deep network [48].

To successfully perform this learning task, Hebbian learning is generally combined with a form of homeostasis [56], as exemplified in Fig 4.24(b) and reported in [22]. Because of the curse of dimensionality [57], differing input distributions are likely to be closer to each other than they are to a randomly chosen initial weight vector. Thus, homeostasis is implemented to distribute activity between competing neurons in order to prevent a single neuron or a few neurons from dominating the receptive field and blocking out other neurons from learning. For example, homeostasis is often implemented in SNNs via adaptive thresholding [58]. In an SNN, increasing or decreasing a neuron's firing threshold is functionally equivalent to decreasing or increasing the magnitude of its weight vector, respectively, and thus also the scope of its Voronoi partition. By setting the firing threshold higher for a neuron that fires more frequently, it becomes restricted in scope, allowing other neurons to compete for other nearby distributions.

Distributing neuronal activity throughout training works well when the input distributions are temporally homogeneous. However, traditional homeostasis presents a difficulty with a temporally changing dataset, as shown in Supplementary Fig. 4.25(a). Once a layer of neurons in an ANN has learned a set of input distributions, if novel input distributions are then presented, homeostasis can often make information from the later distributions replace important information learned from previous distributions, a problem which has been called *catastrophic interference*. This problem is in part due to the *stability-plasticity dilemma* [31]: if weight parameters are too rigid, the network will be stable but won't easily learn new information; if they are too plastic, old information will be lost; and trying to find an in-between global plasticity level can cause corruption between old and new information.

4.8.4 Habituation and sensitization-inspired plasticity modulation

One solution to the stability-plasticity dilemma and catastrophic forgetting has been local modulation of plasticity, allowing some weight parameters to be more plastic than others at different times. Local plasticity modulation can isolate network changes in response to information that is new while preserving information in the other portions of the network that may still be useful. The plasticity modulation of this learning approach is similar to the approach reported in [12] based on novelty-induced dopaminergic modulation of STDP [50] and demonstrates the usefulness of habituation and sensitization characteristics similar to those of the NiO device and Aplysia.

Habituation for stability

When training an ANN, a smaller learning rate provides more stability but requires a larger number of input samples to incrementally converge on the solution state. On the other hand, a larger learning rate allows for more rapid learning but leaves neuron weight vectors unstable, stochastically jittering around the learned distributions. To achieve both reduced training time and solution stability, the learning rate is often designed to start large and gradually reduce over time, as employed by the Habituation model shown in Fig. 4.24(c). For this clustering task, we gradually reduce the learning rate of a given synaptic weight each time the post-synaptic neuron fires, which is similar to the habituated response of NiO devices under repeated exposures to H_2.

Having only a reduction in the learning rate over time is fine for offline learning but fails to allow for adaptation during online learning in a dynamic environment. Therefore, in addition we have designed the learning rate of a synaptic weight to gradually increase over time if there has been a sufficient amount of time during which the associated post-synaptic neuron is no longer being used [12]. This property is comparable to how the resistance response of the NiO device gradually returns to the "forgotten" state when there is sufficient time without an H_2 stimulus. Thus, neurons that have learned a distribution of samples that have been seen more recently and/or more frequently will have more stable reference vectors while neurons that have learned a less frequent or less recent distribution will have more plastic parameters available for adaptation to new information.

Sensitization for novelty

In addition to dynamics that require the habituated gradual modulation of plasticity there may also be sudden changes in the environment, such as the introduction of novel information (Fig. 4.25(c) and 4.25(d)), which require a sudden and temporarily large increase in plasticity to avoid corruption between old information and new information [12]. In biology, this plasticity modulation (such as the sensitization behavior observed in Aplysia) is in part aided by the neuromodulator dopamine, which can be released when novel information is encountered [50]. When dopamine is present, the effect of STDP learning is significantly larger, allowing the network to quickly learn novel information. The sensitization response of the NiO device in the presence of O_3 acts similarly to this biological trigger, temporarily setting the learning rate very high. For this unsupervised clustering task, novel inputs may be identified by low activation values at the reference vector neurons, providing an indication of when the sensitization should be triggered.

In the brain, dopamine is released locally by dopaminergic neurons. Similarly, the described sensitized response should be local, targeted at specific reference vector neuron(s) that can be "reassigned" to the novel information with minimal loss of older information to prevent catastrophic forgetting (Fig. 4.25(d)). With the habituated plasticity response described above, a high current learning rate of a neuron's weight vector is a good identifier of a neuron associated with less-useful information, which may be an eligible target for sensitization.

4.8.5 Illustrative examples

Using a simple, illustrative example in a two-dimensional space representative of input and weight vectors normalized onto the surface of a sphere in the input space, we present samples from various input distributions to three competing reference vector neurons. The presented simulation setup is designed to demonstrate the key differences between the discussed learning rules and highlight the advantage of employing both habituation and sensitization in lifelong learning with sequentially presented distributions, as opposed to traditional homeostasis which is designed for a temporally homogeneous interleaving of the various input distributions throughout the training.

Interleaved classes scenario

For the first scenario shown in Fig. 4.24, three input distributions are presented in an interleaved ordering such that samples from each distribution are seen throughout training in a temporally homogeneous fashion. When there is no homeostasis, i.e. no balancing of firing activity between neurons, a single neuron dominates because it is always the closest neuron (Fig. 4.24(a)), which results in non-usable Voronoi partitions (dotted lines). When homeostasis is added to distribute neuronal firing activity (Fig. 4.24(b)), all neurons get used with each moving separately toward a different cluster because of the competition. However, with a static plasticity, implemented with a constant learning rate, the neuron weight vector locations continue to bounce around due to outliers and fail to stabilize, leading to less accurate Voronoi partitions. To resolve this, the learning rate can be habituated, i.e. starting out large and reducing exponentially each time the neuron is activated—similar to the habituation behaviour of NiO under cyclic H_2 exposure. With such habituation, the neurons successfully stabilize on the centers of the input clusters, which attributes to more accurate Voronoi partitions (Fig. 4.24(c)).

Sequential classes scenario

Fig. 4.25 shows a more challenging unsupervised clustering task when the input ordering is not temporally homogeneous but is instead a completely sequential ordering of input distributions. In this scenario, using traditional homeostasis plus habituation to temporally distribute activity between neurons causes old information to be lost (Fig. 4.25(a)). However, we can apply a sensitization behaviour, similar to the the NiO response after O_3 exposure, by resetting the learning rate α_i of a neuron to its initial value (Table 4.2) in the presence of novel information. When we keep the habituation but replace the traditional homeostasis with this controlled and targeted sensitization response (Fig. 4.25(b)), the network can learn novel distributions by utilizing unused or less-used neuron instead of corrupting valuable older information in other portions of the network. This method of habituation plus sensitization can also successfully learn the previous interleaved example with temporally homogeneous input ordering (Fig. 4.24(d)) and is thus useful for both scenarios.

Memory-constrained scenario

To highlight how controlled sensitization can reduce catastrophic interference or corruption between old and new information, we extend this illustrative example to be memory-constrained. As shown in Figs. 4.25(c) and 4.25(d), we present a fourth input distribution after the previous three distributions have already been learned, but we keep only three reference vector neurons to demonstrate the response when some forgetting must necessarily occur. With the traditional homeostasis plus habituation rule (Fig. 4.25(c)), all three neurons attempt to learn the new distribution, ending somewhere between their old and their new positions. The result is catastrophic interference, failing to completely learn the new information and corrupting the old information. However, with controlled sensitization added to habituation (Fig. 4.25(d)), the response to new information can be targeted to replace the least useful information, in this case the oldest information. As shown in Fig. 4.25(d), sensitized plasticity causes a rapid, isolated response that learns the new information without corruption and while keeping as much of the old information as possible. Therefore, this online unsupervised clustering example demonstrates the usefulness of having weight parameters in an ANN that undergo local, dynamic modification to their plasticity in patterns similar to the habituation and sensitization responses observed in NiO.

4.8.6 Simulation methodology

During this unsupervised training task, samples are presented one-by-one. For the current sample j, a single neuron wins and undergoes Hebbian learning. Table 4.2 includes the variations in the learning and evaluation rules for each learning scenario. For the current sample, each neuron i is evaluated. The winning neuron is the neuron that has the shortest distance from that neuron to the current sample, scaled by the distance scaling factor for each neuron, d_{i}. The weight vector of the winning neuron ($\vec{w}_{\mathrm{i}}$) is adjusted according to

Table 4.2. Learning rules for each learning scenario.

Learning Rules	**Distance Scaling**	**Initial Plasticity**	**Habituated Plasticity Rules**	**Sensitized Plasticity Rules**
No Homeostasis	$d_i = 1$ (no scaling)	$\alpha_i = 0.5$ (constant)	N/A	N/A
With Homeostasis	$d_i = \theta_i + \theta_{base}$ $\Delta\theta_{winner} = 1E2$ $\frac{d\theta_i}{dt} = -0.1\theta_i$ $\theta_{base} = 1$	$\alpha_i = 0.5$ (constant)	N/A	N/A
Homeostasis + Habituation	(same as above)	$\alpha_i(0) = 1$	$\frac{d\alpha_i}{dt} = 0.0001(1 - \alpha_i)$ $\Delta\alpha_{winner} = -0.1\alpha_{winner}$	N/A
Habituation + Sensitization	$d_i = 1$ (no scaling)	$\alpha_i(0) = 1$	(same as above)	$\alpha_{reset} = 1$ $d_{threshold} = 0.2$

$\Delta\vec{w}_i = \alpha_i(\vec{s_j} - \vec{w}_i)$, where $\vec{s_j}$ is the input vector for sample j, and α_i is the learning rate or plasticity level for neuron i.

Homeostasis

If homeostasis is used, the distance scaling factor d_i consists in part of a dynamic θ_i which increases for the winning neuron according to $\Delta\theta_{winner}$, and decays exponentially for all neurons through time (the time scale being normalized to 1 unit time per sample) according to $\frac{d\theta_i}{dt}$.

Habituation

When habituation is not used, the learning rate α_i remains constant. With habituation, on the other hand, the initial values $\alpha_i(0)$ start higher, but are reduced exponentially each time a neuron wins according to $\Delta\alpha_{winner}$ (applied after the weight change) and otherwise slowly saturate back up to the original values according to $\frac{d\alpha_i}{dt}$ when it is not winning.

Sensitization

When sensitization is used, if the distance to the winning neuron is less than $d_{threshold}$, then instead the neuron with the highest α_i is chosen as the winner and its learning rate is reset to α_{reset} before undergoing a weight change. Sensitization is used in conjunction with habituation, and the habituation rules that exponentially adjust α_i also apply.

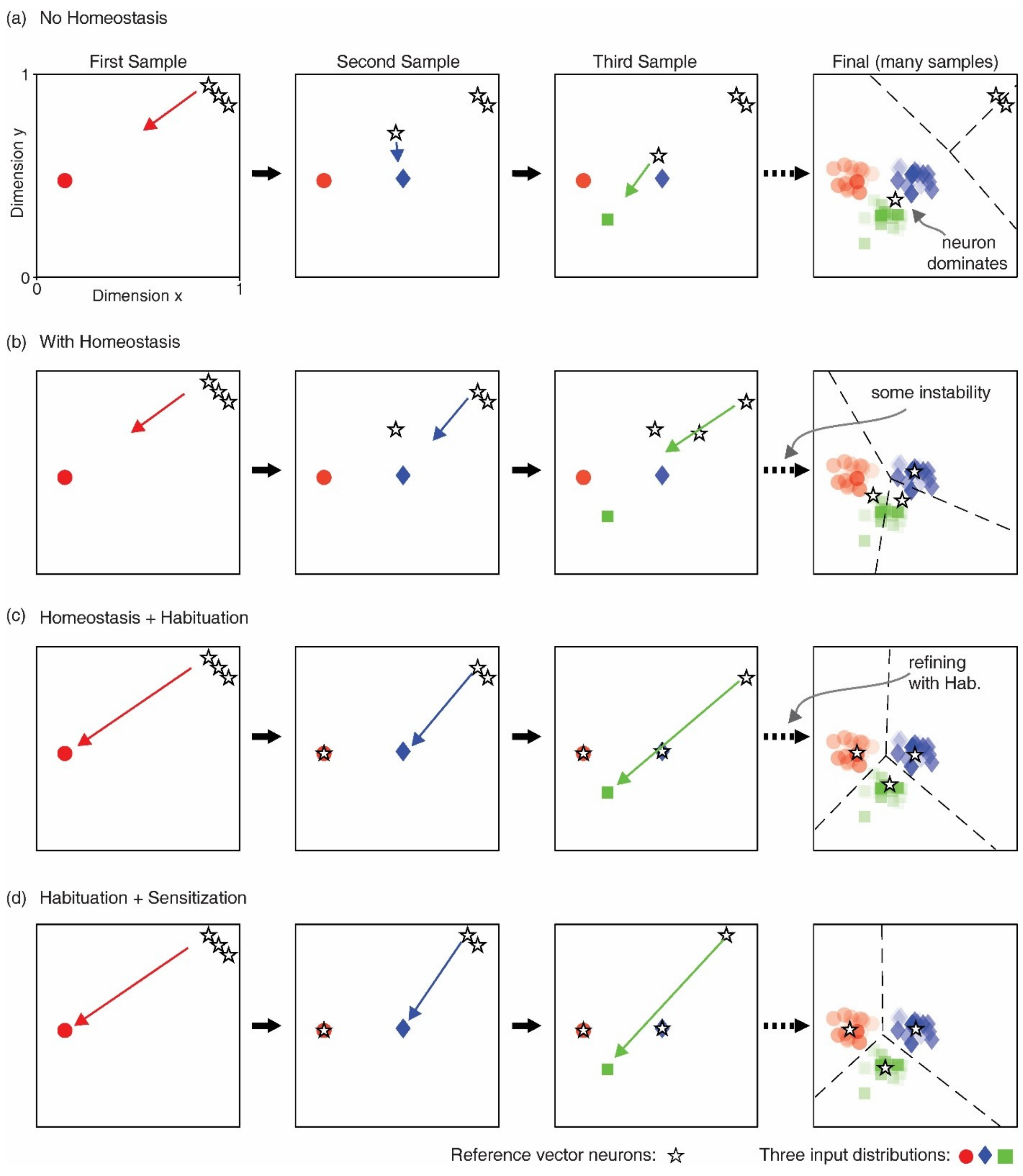


Figure 4.24. Illustration of *interleaved* unsupervised online clustering task in a layer of reference vector neurons: (a) no homeostasis; (b) with homeostasis; (c) homeostasis with habituation; (d) homeostasis with sensitization.

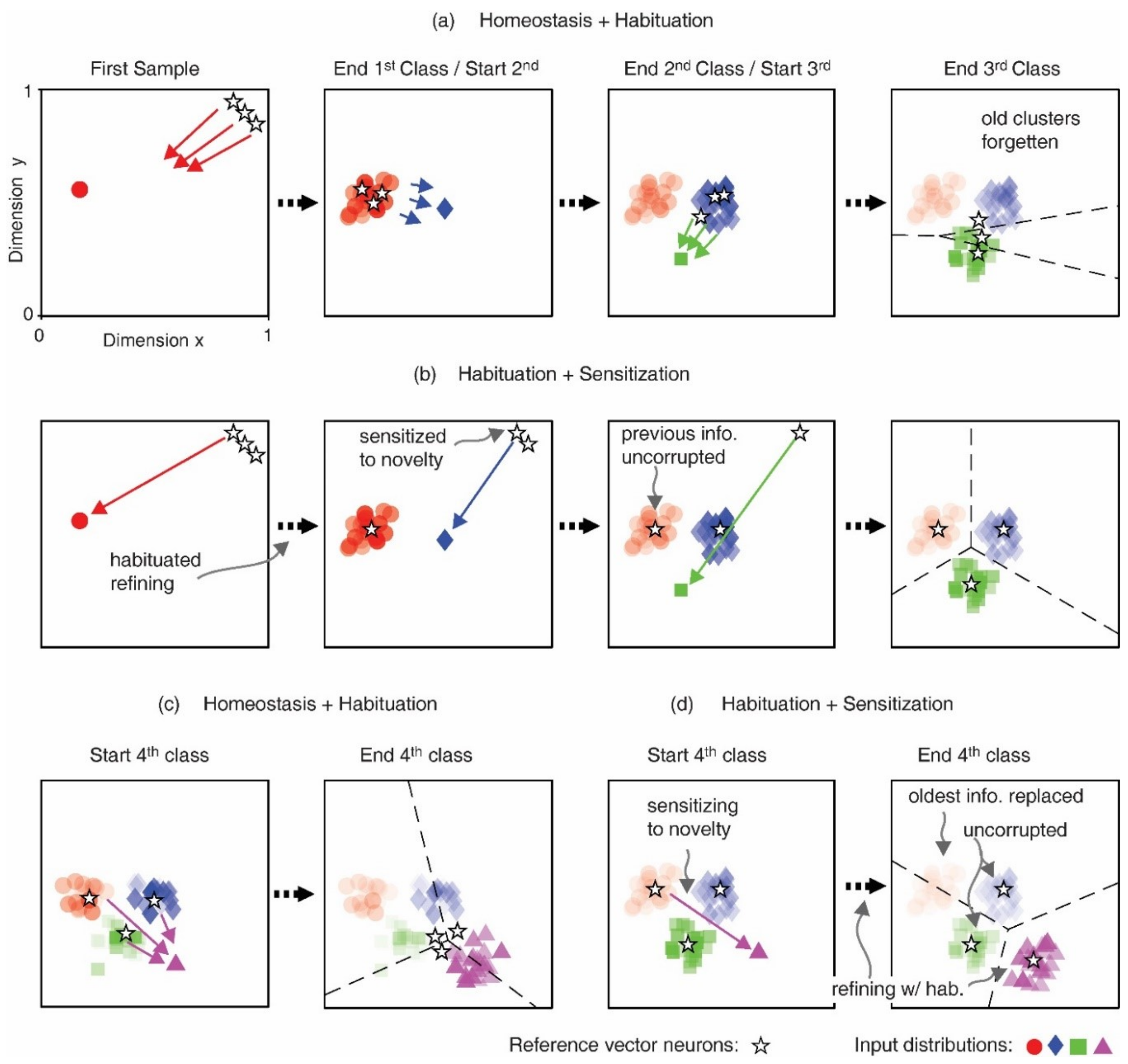

(a) Homeostasis + Habituation
First Sample
End 1st Class / Start 2nd
End 2nd Class / Start 3rd
End 3rd Class
old clusters forgetten
Dimension y
Dimension x
1
0
0
1
(b) Habituation + Sensitization
sensitized to novelty
previous info. uncorrupted
habituated refining
(c) Homeostasis + Habituation
(d) Habituation + Sensitization
Start 4th class
End 4th class
Start 4th class
End 4th class
sensitizing to novelty
oldest info. replaced
uncorrupted
refining w/ hab.
Reference vector neurons: ☆
Input distributions: ● ◆ ■ ▲

Figure 4.25. Illustration of *sequential* unsupervised online clustering task in a layer of three reference vector neurons: (a) homeostasis with habituation, 3 sequential classes; (b) habituation with sensitization, 3 sequential classes; (c) homeostasis with habituation, adding a fourth class to pre-trained network; (d) habituation with sensitization, adding a fourth class.

5. WEIGHT-LOCALIZED CONVOLUTIONS ON DIRECT-MAPPED HARDWARE

Hardware implementations of neural networks with the intrinsic capability of real-time computing on analog, asynchronous signals are a promising avenue for the future of machine learning. As introduced in Section 1.1.1, the success of machine learning has been significantly advanced by neural networks employing convolutions, which promote learning generalizations through subsampling and parametric reduction.

However, for neuromorphic hardware implementations with dedicated neural units, sharing kernel weights between the convolutional windows can be disadvantageous because it generally requires replicating the shared kernel components to implement the entire feature map as well as the global distribution of weight updates for the replicated components. On the other hand, hardware implementations comprising generic functional units require the storage and regeneration of internal signals and synaptic weights, which is difficult for analog, asynchronous designs.

This chapter, taken from my work in [17] (©2017 IEEE[1]), presents a method of performing convolutions over time in a way that avoids both kernel replication and the storage and regeneration of internal signals, thus allowing for local weight updates, direct node-to-node synaptic connections, and reduced network area. Convolving over time is accomplished via recurrent connections that allow for sequentially sharing kernel weights for each convolutional window while still propagating the computed signals.

We evaluate the proposed approach with a proof of concept, showing only a slight accuracy reduction as a tradeoff for the previously mentioned connectivity, area, and storage benefits.

[1]↑In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Purdue's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

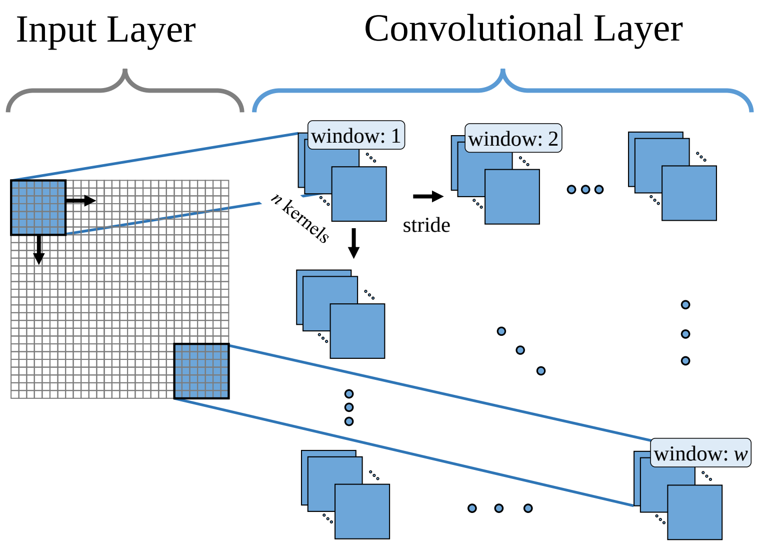


Figure 5.1. A traditional convolutional layer. (©2017 IEEE [17])

5.1 The Challenge of Weight Sharing in Neuromorphic Hardware

CNNs enact convolutions by sharing kernel weights across the input windows. Figure 5.1 shows a single convolutional layer where all n different kernels are shared over each w windows of convolution. In this section, we first discuss traditional methods of CNN weight sharing in hardware. Then, we identify the disadvantages of these approaches with regard to biological plausibility and neuromorphic design.

5.1.1 Traditional CNN weight sharing in hardware

There are generally two methods of enabling the necessary weight sharing between kernel instances for implementing convolutional neural networks in hardware. The first method replicates dedicated hardware components. The second reuses generic components.

Weight sharing via dedicated component replication

Very fast neural network performance can be achieved when the network's weighted synaptic connections, integration units, and activation functions are synthesized as dedicated components in the hardware for each corresponding instance of the various logical components. Specifically, each kernel is replicated for each window over which it convolves, and the weights of a kernel are identical between the various instances of that kernel. Dedicated components for each instance provide maximum parallelism within the network and allow analog and/or asynchronous signals to propagate between layers via direct node-to-node synaptic connections.

Weight sharing via generic component reuse and shared memory

When area and power budgets limit the number of available hardware components, generic functional units can execute on latched activation values and stored weights that are loaded from shared memory when needed. With this method, the logical neural units are not tied to specific hardware units. Therefore, area and power goals can be balanced with desired parallelism goals, according to hardware constraints.

5.1.2 Biological plausibility and neuromorphic CNNs

Neuromorphic designs are inherently more apt to compute on biologically plausible asynchronous, analog signals (e.g. [59]). However, the weight-sharing requirement of convolutional architectures is an obstacle for implementing hardware convolutions using analog or asynchronous signals.

Both of the methods described in the previous subsection are lacking in biological plausibility. Although biological plausibility is not a necessity for neuromorphic design, the specific areas in which these two methods differ from biology are the cause of significant disadvantages to neuromorphic design. As a review from Section 1.2.2, four possible attributes of neuromorphic systems include: (1) local learning, (2) fixed configuration, (3) dedicated units and hardwired connections, and (4) asynchronous or analog computation.

The disadvantages of dedicated component replication

While the brain prominently exhibits significant redundancy, the discussed method of kernel replication is at odds with biology's localized learning. When kernel instances are replicated in hardware, updates to the weight values must be calculated for each instance, then summed or averaged, and then redistributed back to all the instances. This coordination between instances requires additional infrastructure and synchronization overhead. Such weight coordination is often difficult in neuromorphic designs of asynchronous analog circuitry.

Further, the parallelism benefit of kernel replication comes with a substantial area and power cost. Quantified, a single convolutional layer with n kernels, each of size m, requires $n \cdot m$ incoming synaptic weights for a single window. If the kernels are convolved over w windows, then the n kernels would be replicated w times, with each instance receiving a different selection of inputs indexed appropriately from the preceding layer. In total, the $n \cdot w$ kernel instances would require $n \cdot m \cdot w$ incoming synaptic weights, just at that layer. For deep CNNs, such hardware dedication is expensive.

The disadvantages of generic component reuse

The use of generic components means that the operands on which it is computing are not intrinsically local or directly connected to the functional unit, unlike neural networks in the brain. For hardware designs, this non-locality presents data hazards when a unit is stalled waiting for the appropriate kernel weights to be loaded or for the incoming values from the previous layer to be made available. Beyond these data hazards are the structural hazards that arise when there are insufficient hardware resources. In other words, without dedicated components for each logical unit in the network, the number of available functional units limits the possible parallelism. Both types of hazards limit the speed of such a network. In addition, storing and regenerating analog or time-based signals in neuromorphic designs is challenging.

Ideally, neuromorphic designs invite weight-sharing techniques that avoid the replication of weighted synaptic connections and eliminate both the need to load weight values from shared memory and the need to redirect computed values to available components. These stipulations lead us toward a neural network that employs recurrent connections to convolve in time, rather than space.

5.2 Convolving Over Time with Recurrent Convolutional Neural Networks

In addition to the discussed feedforward networks, networks with feedback connections can compute temporally. These Recurrent Neural Networks (RNNs) operate over several time steps, allowing it to compute based on both the current inputs and the internal network state.

In this section, we present a neural network that uses such recurrent connections to convolve over time and enable sequential weight sharing. For simplicity, we will refer to these networks as Recurrent Convolutional Neural Networks (RCNNs). By sharing weights in time, these RCNNs can perform convolutions with dedicated hardware components, while avoiding replication, data storage, and data redirection.

5.2.1 Related works

As our work combines the benefits of CNNs with the benefits of RNNs, we briefly discuss previous works that also merge the two network models. Donahue et al. [60] provide methods in visual recognition and description tasks that utilize recurrent connections to pass information between input frames in convolutional networks or to provide feedback from previous outputs when producing sequential outputs. Pinheiro and Collobert [61] add feedback connections to a CNN to converge iteratively on scene predictions. Similarly, Liang and Hu [62] utilize recurrent connections within a single frame to modulate convolutional units by their neighboring units, creating an effectively deeper network with fewer parameters. These methods all use recurrent connections to improve or augment the CNNs. However, in this work, we employ recurrent connections to implement the convolutions themselves, allowing for kernel weight sharing in time, which we describe next.

5.2.2 Weight sharing in time instead of space

Without replication of dedicated components or reuse of generic components, *time* is the remaining dimension for sharing kernel weights over the windows of convolution. Figure 5.2 conceptually presents this convolutional method. Sufficient functional units can be

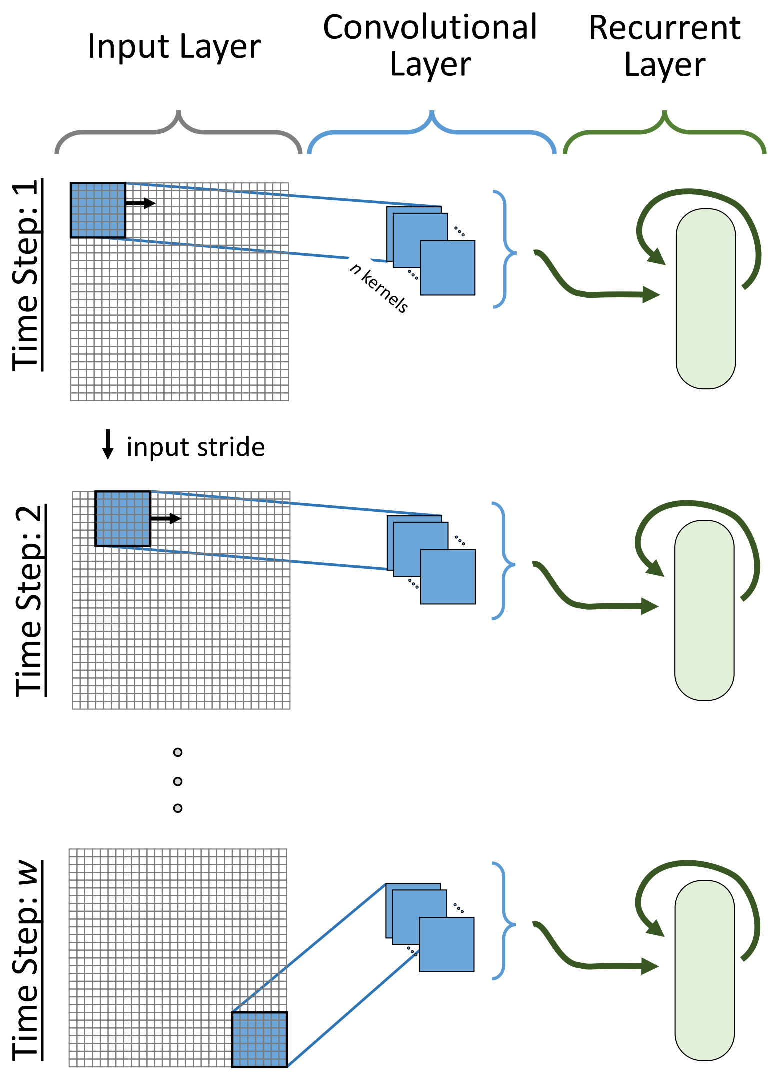


Figure 5.2. Convolutions over time with the same kernels at each time step. (©2017 IEEE [17])

synthesized to provide dedicated components for each kernel, which are then reused in time for each window instance. Of course, simply performing the convolutions for each window sequentially is not in itself novel or that beneficial. It simply serializes the previously parallel process. The true benefit is the elimination of data storage and data redirection for (1) the incoming signals, (2) the internal synaptic weights, and (3) the outgoing signals.

Incoming signals

Rather than each kernel convolving over the windows of a fixed input, we can now view each kernel as the fixed component, while the preceding input layer presents the values from each window at different time steps using the same synaptic connections. Thus, convolving over time on a dedicated hardware unit removes the data hazards and data traffic issues associated with redirecting the incoming values toward whichever (if any) unit is available.

Internal synaptic weights

The dedicated kernel components also eliminate the data hazards and data traffic for synaptic weight values because the weight values do not need to be stored and loaded—they are embedded in the dedicated kernel hardware. Further the weight updates can occur locally, as there is only one instance of each weight value. Quantified, this method requires only n kernel instances (just one instance for each n kernels) and $n \cdot m$ incoming synaptic weights at the given layer. Thus, convolving over time reduces the kernel area by a factor of w when compared to the replication method.

Outgoing signals

Whether convolving over time or space, the same number of convolutions occur, and the given layer still produces n values for each w windows. If simply stored until the other time steps complete, these values would have a storage requirement of $O(n \cdot w)$. However, because of the recurrent connections discussed next, the outgoing signals undergo state compression as they are retained over the remaining time steps in a recurrent layer, further reducing area and eliminating data redirection on the outgoing side of the convolutional layer.

5.2.3 A recurrent state compression/retention layer

Information from previous states can be effectively retained and compressed by appropriately utilizing recurrent connections [63], [64]. In analog neuromorphic designs, latching computed values and recalling them from storage is not as straightforward as it is for digital designs. This is also true asynchronous or time-based signals, such as spike trains. Because the network reuses the dedicated kernel hardware components in the subsequent time step, the produced values must be transmitted to the next layer as they are generated. We use the recurrent layer to receive these signals.

At each time step, the outgoing signals of each kernel propagate into the recurrent layer. This layer has sparse recurrent connections creating built-in, interleaving cycles of systematic lengths to retain information over the remaining time steps. For the RCNNs in the following experiments, the neurons in the recurrent layer are arranged in a 2-D grid where each (i, j)th node (zero-indexed) sends its signals to every node in the $(\mathrm{i}+\mathrm{j}+1)$st column, modulo the grid dimensions. Training these recurrent connections allows the network to retain only the most relevant information from the preceding time steps as the data signals merge and interfere with each other in the recurrent layer. The resulting compression allows the recurrent layer to be much smaller than if each signal were to be preserved in its entirety throughout the remaining time steps.

Figure 5.3 illustrates this complete architectural setup with an example RCNN. Overall, the key to this approach is that rather than replicating all the kernels for each window, the same kernel components are used for all the windows—one window per time step, and rather than storing and regenerating internal signals, the signals are received, retained, and compressed via recurrent connections.

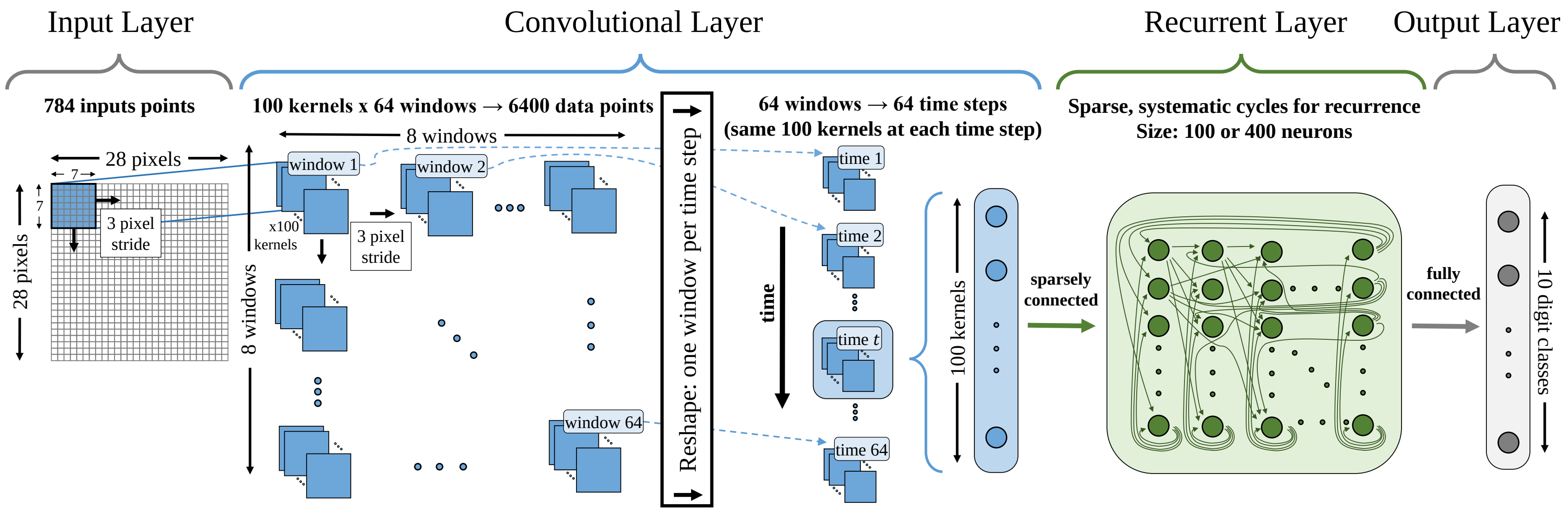


Figure 5.3. Example architecture of a neural network with convolutions implemented via recurrency. (©2017 IEEE [17])

5.3 Experimental Proof of Concept

Here we validate this approach of convolving over time while still propagating internal signals with a simple proof of concept experiment.

5.3.1 Methodology

To validate the effectiveness of convolving over time, we set up a comparison between the proposed RCNN architecture and a traditional CNN architecture. This section presents the details of our experiment, the results of which we provide in the following section.

Network training

The networks were trained in MATLAB with standard gradient decent, using backpropagation for the traditional CNNs and backpropagation through time (BPTT) for the RCNNs. To perform BPTT, the RCNNs were logically unrolled for a number of time steps equal to the number of convolution windows. Although other recurrent training techniques may be used, BPTT was sufficient for validating the architecture.

The networks were trained and tested using the MNIST dataset [27], which is a collection of grayscale images representing handwritten digits 0-9. There are 60,000 images in the training set and 10,000 images in the testing set. We presented the training set to the network for 30 passes, with a different random permutation of the set for each pass. After training, the weight values were fixed, and the accuracy values for both the training and testing sets were calculated.

Architecture and hyperparameters

The hyper-parameters in our setup were arbitrarily selected to allow for a more fair comparison between the RCNNs and the traditional CNNs, which exist in different optimization spaces. For the input layer, the MNIST values were scaled to [-1, 1], corresponding to the output range (-1, 1) of the selected non-linear hyperbolic tangent activation function used throughout the network. Neuron outputs were capped within that range and output targets were set within that cap to prevent migration towards the extremes where the gradient approaches zero and restricts weight updates.

The convolution kernels were size 7x7 with a stride of three. As the images in the MNIST dataset are of size 28x28 pixels, the kernel size and stride result in 8x8 (64 total) convolutional windows. The number of kernels was set to 100 for each network design. The network sizes were designed with the goal of comparability given the competing architectures. For example, in place of the recurrent layer in the RCNNs, the CNNs had an equivalently sized hidden layer with comparable connection density. For both the CNNs and the RCNNs, two different sizes for this layer were evaluated: 100 neurons and 400 neurons.

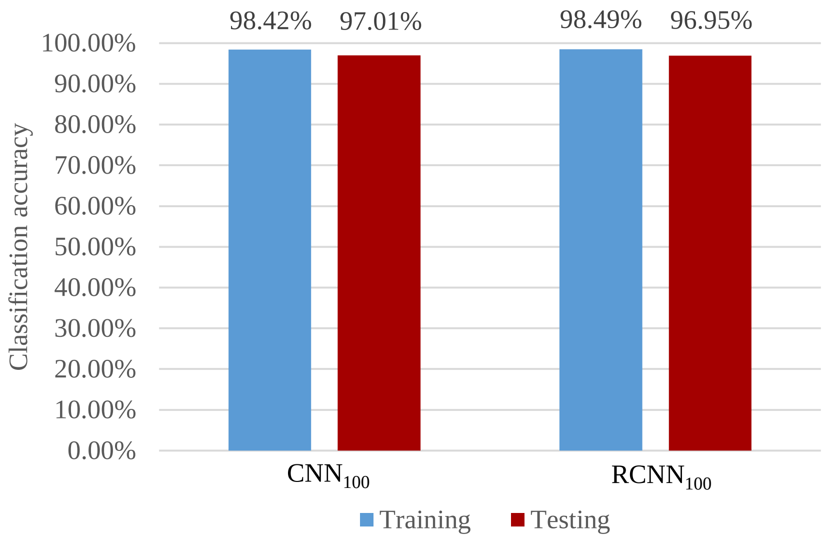
5.3.2 Results and discussion

We compare a traditional CNN that has 100 post-convolution hidden layer neurons (CNN_{100}), a CNN with 400 hidden layer neurons (CNN_{400}), an RCNN with 100 recurrent layer neurons (RCNN_{100}), and an RCNN with 400 recurrent layer neurons (RCNN_{400}). After presenting the classification accuracy, we discuss the network size and latency implications of these four networks.

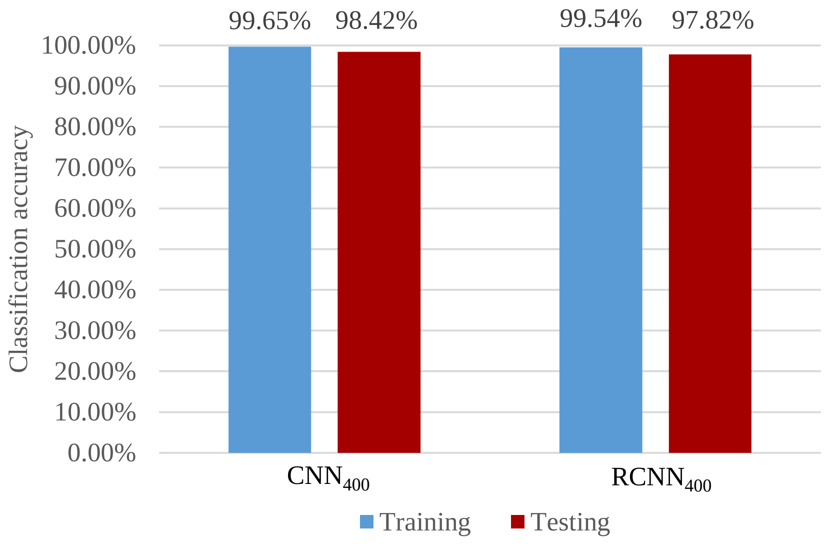
Network accuracy

Figure 5.4a compares the classification accuracy of CNN_{100} and RCNN_{100}. As expected, the testing accuracy degrades from CNN_{100} to RCNN_{100} because the convolutional signals in an RCNN are not preserved in their entirety. However, the overall accuracy levels are quite comparable, with only a 0.06% reduction in testing accuracy.

Similarly, Figure 5.4b presents the classification results when we increase the size of the hidden/recurrent layer to 400 neurons. With more neurons the accuracy rates increase compared to the smaller networks, as is typical. In these larger networks, the testing accuracy reduction from CNN_{400} to RCNN_{400} is only 0.60%. These values are encouraging, especially when considering the RCNN benefits discussed in Section 5.2 and the fact that hyper-parameter optimization and deeper networks (see Section 5.4) are natural avenues for improving the classification accuracy beyond this initial comparison.



(a) Size 100



(b) Size 400

Figure 5.4. Accuracy results of a traditional CNN compared to the proposed RCNN with equivalent-sized fully connected hidden layer (CNN) and recurrent layer (RCNN) neurons with (a) 100 or (b) 400 hidden layer / recurrent layer neurons. (©2017 IEEE [17])

Table 5.1. Number of trainable parameters (synaptic weights). (©2017 IEEE [17])

Architecture		Post-input Network Layers			
Type	**Weight Sharing**	**Conv.**	**Hidden / Recurrent**	**Output**	**Total**
CNN_{100}	Kernel replication	4900	1612	1010	7522
CNN_{100}	Component reuse	4900	1612	1010	7522
$RCNN_{100}$	Convolving in Time	4900	1505	1010	7415
CNN_{400}	Kernel replication	4900	9404	4010	18314
CNN_{400}	Component reuse	4900	9404	4010	18314
$RCNN_{400}$	Convolving in Time	4900	8975	4010	17885

Discussion on network size

One benefit we note in the proposed RCNNs is the substantial network size reduction when compared to CNNs that replicate components for kernel weight sharing. Although the CNNs were designed with a comparable number of training parameters (see Table 5.1), the CNNs that replicate kernel components for weight sharing in hardware have 64 times the number of neuron components in the convolution layer when compared to the RCNNs (see Table 5.2). This ratio is equal to the number of convolution windows, as the RCNN requires only one instance of each kernel, which it uses for all windows.

As discussed in Section 5.1, a CNN can reduce its parallelism (and thus increase its latency) to avoid the larger hardware resource requirement by reusing generic, or even kernel-specific components for multiple windows. However, since the subsequent layer cannot proceed until the convolutional layer is complete, a traditional CNN that reuses components must still store and then regenerate all of the produced neuron outputs. Such storage or regeneration requirements complicate any possible neuromorphic implementations of analog or time-based neural networks. On the other hand, the proposed RCNN can pass produced values or time-based signals into the recurrent layer as they are generated, further reducing area costs by avoiding this restricting storage requirement when compared to a CNN with generic components.

Table 5.2. Neuron counts. (©2017 IEEE [17])

Architecture		**Post-input Network Layers**			
Type	**Weight Sharing**	**Conv.**	**Hidden / Recurrent**	**Output**	**Total**
CNN_{100}	Kernel replication	6400	100	10	6510
CNN_{100}	Component reuse*	6400	100	10	6510
$RCNN_{100}$	Convolving in Time	100	100	10	210
CNN_{400}	Kernel replication	6400	400	10	6810
CNN_{400}	Component reuse*	6400	400	10	6810
$RCNN_{400}$	Convolving in Time	100	400	10	510

* Note: For CNNs with generic component reuse, this chart represents the number of logical neurons. The number of hardware components is dependent on the specific implementation.

Discussion on network latency

While RCNNs come with the additional latency cost of serialization when compared to a completely parallelized CNN, there are a few latency benefits associated with RCNNs when compared to CNNs with the same availability of hardware resources. Unlike CNNs that reuse generic components, an RCNN does not have the latencies that result from data redirection, data hazards, structural hazards, reading/writing weight values, storing and regenerating internal signals, or control/synchronization penalties.

Further, for larger images our proposed RCNNs may be combined with attention-based RNNs, such as that proposed by Mnih, et al. [65], in order to redirect the windows of the convolutional network towards the most information-rich areas of the image. By so doing, the RCNN may be able to improve latency by only convolving over the most relevant portions of the image. Attention-based RCNNs are the subject of future work, as are deeper networks and spiking neural networks as discussed next.

5.4 Potential Future Work

Future work prescribes the analysis of recurrent convolutions on SNNs with spike train inputs. Further, we have hereto only validated the proposed method of convolving in time for networks with a single convolutional layer. This method may be expanded to deeper networks. The kernels of subsequent layers may be formed via small competing recurrent clusters or via a larger recurrent cluster with competing outgoing weights. These recurrent clusters in the later stages would receive appropriately timed input from the outputs of the kernels of preceding layers.

Consider the example in Figure 5.5. If convolutional layer B has windows that encompass a 2x2 area from the outputs of convolutional layer A, then the time steps for the recurrent cluster(s) for layer B will encompass 4 time steps of the recurrent clusters in layer A (e.g. time step B1 covers time steps A1, A2, A3, and A4). Thus, the order of input windows cannot simply be left to right, top to bottom. The inputs should be presented in a zigzag pattern that respects the overall window order required by later convolutional layers. Overlap between windows in later stages may then require input signals to be repeated. Careful stride patters, such as spiraled strides, may reduce the required repetition.

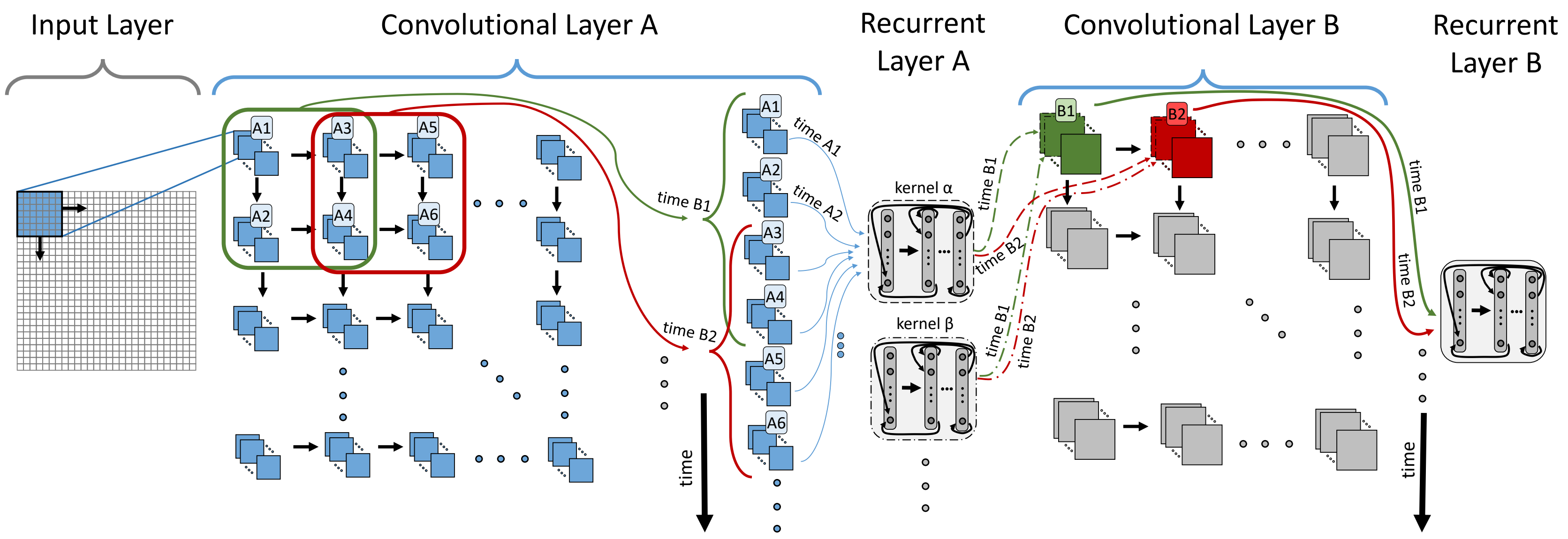


Figure 5.5. Example architecture of an RCNN with two convolutional layers implemented via recurrency. (©2017 IEEE [17])

6. EXPLICITLY TRAINED SPIKING SPARSITY (ETSS)

SNNs are being explored in machine learning in part for their potential energy efficiency benefits due to the inherent computational sparsity that comes from event-driven computation [9]. The computational energy consumed in a spiking network during inference is highly correlated with the number of spikes that occur because each spike at a given neuron induces accumulation computations in each of that neuron's fan-out neurons as well as a membrane reset computation. Thus, reducing spiking activity is an important part of improving energy efficiency in an SNN.

Until recently, training large-scale SNNs for complicated datasets was a difficult task because discontinuous neuron activations are non-differentiable, preventing direct backpropagation. One of the first workarounds to this problem was converting a pre-trained, non-spiking ANN to an SNN [29]. This approach allowed for competitive inference on an SNN for complicated tasks like ImageNet, but it failed to capture the energy efficiency benefits of sparsity. This failure was because the networks were trained in a highly precise, deterministic environment, and switching to the stochastic environment of an SNN reduces resolution at small time scales, requiring a larger inference time to accurately distinguish between close activation values. This larger inference time results in a significant number of spiking operations, limiting energy efficiency benefits.

However, more recent works have demonstrated effective methods at backpropagating directly in a spiking environment, e.g. [66], [67]. These methods approximate the gradients over the discontinuous spiking activations, allowing for backpropagation through a deep SNN. Lee et. al [67] have shown that this method of spiking backpropagation significantly reduces the inference time required, and, with that, the total number of spikes and computations that occur per inference, further improving the energy efficiency.

A beneficial side effect of these approximate spiking gradient techniques is that spikes themselves may now be included in these surrogate backpropagation learning algorithms. Since spiking sparsity is an energy goal of SNNs, this chapter presents the proposal of including spiking activity directly in the loss function (Section 6.1), explicitly training the SNNs to be more sparse in a multi-objective optimization process, taken from my work in

[18][1]. We will refer to these networks as SNNs with Explicitly Trained Spiking Sparsity (ETSS).

We additionally explore a simulated annealing-inspired loss function, providing backpropagation with a dynamic weighting of the two optimization goals (accuracy and sparsity), which can help avoid early local minima or solution states that catastrophically sacrifice accuracy (Section 6.2). We compare these preliminary results (Section 6.3) to a modification of Multi-Objective Particle Swarm Optimization (Section 6.4). When then expand into larger networks by beginning with a pre-trained network (Section 6.5).

We note similar work in the SNN "activity regularization" portion of [68] that was developed concurrently with our work[2]. Our simulated annealing-inspired loss function which dynamically adjusts the balance between the multiple objectives during training is a unique contribution of our work, allowing the sparsity objective to be even stricter over time without catastrophically reducing accuracy.

6.1 Balancing Accuracy and Sparsity with Surrogate Gradients

In SNNs, a spiking activation is often modeled as inducing an instantaneous weighted potentiation in the membrane potentials of fan-out neurons. Emre et. al [66] have analyzed the effectiveness of various "surrogate" or approximate gradients over spiking activations, including fast-sigmoid, linear, and exponential surrogates, and have developed open-source code for easy backpropagation in PyTorch using these surrogate gradients, called SpyTorch [21]. These approximate gradients allow us to choose any criterion for the loss function, $L()$, based on final classification error, e.g. mean squared error, cross entropy, etc., and let the automatic software tools perform backpropagation.

$$L_{classification} = criterion(output, target) \tag{6.1}$$

[1]↑This reference is a pre-print which, at the time of submitting this dissertation, currently contains only preliminary results. The expanded results in this chapter are expected to be added to the pre-print soon. This material may or may not undergo a future copyright transfer, but will retain permission to be included here. This version is not necessarily endorsed by any potential future publisher in its current form.

[2]↑Preliminary results from our work were first published internally to sponsors on 5 Oct 2019 and then publicly at [18] on 2 Mar 2020. The work in [68] was first published on 3 Nov 2019. All updated results presented here were produced before we were made aware of [68].

Poggio et. al [69] have shown that the global minima for over-parameterized networks often reside in flat valleys or basins within the optimization space. This means that many neighboring solutions have near equivalent accuracy. These flat regions provide flexibility. Our goal is to let the optimizer find solutions within those flat basins that have less spiking activity without compromising accuracy too much.

Being able to differentiate over spiking events enables gradient descent based on a loss that includes a measure of those spikes. The total loss, then, can be a combination of both the classification accuracy goal *and* the spiking sparsity goal:

$$L_{total} = L_{classification} + L_{sparsity} \tag{6.2}$$

$$L_{sparsity} = \sigma(spike\mathrm{e}Count) \tag{6.3}$$

where $\sigma()$ is a weighting function that scales the spike count loss to provide appropriate balancing between the two loss components in (6.2).

6.2 Simulated Annealing-inspired Optimization Balancing

The most trivial approach for $\sigma()$ would be to use a constant scalar:

$$\sigma_{constant}(spike\mathrm{e}Count) = \sigma_0 * spike\mathrm{e}Count \tag{6.4}$$

We consider a potential problem from adding in a sparsity loss function. When we change the optimization topography, if the optimizer is constrained for sparsity too much, too early in the training process, the gradients in the new landscape may not allow the system to reach solution states that also reside in the basins of the classification landscape, causing a significant reduction or complete failure of the classification accuracy. So for the constant sparsity loss scaling function, σ_0 must be small enough to allow the classification loss to dominate the total gradient direction if classification accuracy is to be maintained. However, letting the classification loss dominate too much, even after reaching the basins, may fail to achieve the best sparsity.

We explore a potential solution to this problem, inspired by simulated annealing–allow the optimizer to disregard sparsity early in the training process and then slowly increase the constraints for spiking sparsity as training continues. This approach makes $\sigma()$ a function of training time, or more simply, the current training epoch. In addition to the constant sparsity loss function, we evaluate five annealing-inspired sparsity loss functions that increase the sparsity constraint over time. These include a linear increase (6.5) and a quadratic increase (6.6). The next two loss functions alternate between excluding and including the sparsity loss function throughout the training process, where the portion of epochs in which the sparsity loss is included increases linearly during training from zero-inclusion during the first epoch to always-included in the last epoch. The first of these alternating loss functions uses a constant σ_0 when it the sparsity loss is included (6.7), and the other uses a linearly increasing σ_0 when it is included (6.8). The final sparsity loss function switches every 5 epochs between including or excluding the sparsity loss (6.9).

$$\sigma_{linear}(spike\text{e}Count, n_{\text{epoch}}) = \left(\sigma_0 \cdot \frac{n_{\text{epoch}}}{N_{\text{epochs}}}\right) \cdot spike\text{e}Count \tag{6.5}$$

$$\sigma_{quadratic}(spike\text{e}Count, n_{\text{epoch}}) = \left(\sigma_0 \cdot \frac{(n_{\text{epoch}})^2}{N_{\text{epochs}}}\right) \cdot spike\text{e}Count \tag{6.6}$$

$$\sigma_{alternating}(spike\text{e}Count, n_{\text{epoch}}) = \Big(A(n_{\text{epoch}}) \cdot \sigma_0\Big) \cdot spike\text{e}Count \tag{6.7}$$

$$\sigma_{alternating\_linear}(spike\text{e}Count, n_{\text{epoch}}) = \left(A(n_{\text{epoch}}) \cdot \sigma_0 \cdot \frac{n_{\text{epoch}}}{N_{\text{epochs}}}\right) \cdot spike\text{e}Count \tag{6.8}$$

$$\sigma_{on\_off}(spike\text{e}Count, n_{\text{epoch}}) = \left(\sigma_0 \cdot \left(\left\lfloor \frac{n_{\text{epoch}}}{5} \right\rfloor \bmod 2\right)\right) \cdot spike\text{e}Count \tag{6.9}$$

where $A()$ is a binary value following the the alternating function discussed above.

Table 6.1. Best preliminary sparsity results for each method when allowing for an up to *0.5% drop* in validation accuracy for CIFAR-10 on a VGG-5 trained from scratch.

	Best Validation Hyperparameters		Validation Results		Testing Results	
Loss Function	σ_0	Training Epochs	Accuracy	Average Spikes	Accuracy	Average Spikes
CrossEntropy (baseline)	0	124	81.49%	698,579	82.17%	663,577
C.E. (baseline) w/ drop	0	88	81.06%	679,384	81.61%	648,981
C.E. + $\sigma_{constant}$	1.0e-05	87	81.12%	644,459	82.46%	359,542
C.E. + σ_{linear}	1.0e-07	125	81.17%	489,694	82.22%	455,087
C.E. + $\sigma_{quadratic}$	1.0e-09	125	81.02%	530,408	82.11%	488,893
C.E. + $\sigma_{alternate}$	0.002	93	81.24%	94,996	81.69%	82,113
C.E. + $\sigma_{alternate\_linear}$	1.0e-05	124	81.28%	121,142	81.24%	105,012
C.E. + σ_{on_off}	0.002	120	81.02%	77,149	81.26%	73,305

6.3 Preliminary Results

Preliminary experiments were conducted in PyTorch with SpyTorch using backpropagation with piece-wise linear surrogate gradients on the Cifar-10 dataset with the VGG-5 architecture for 125 epochs. Cross entropy was chosen as the classification loss function. For each of the sparsity loss functions discussed above, we performed a hyperparameter search to discover the largest σ_0 that still provides acceptable classification accuracy based on the validation set. We set aside 20% of the training set as a validation set and trained with the remaining 80%. Both the sparsity loss scaling constant, σ_0, and the number of training epochs were determined based on the validation results.

Using the hyperparameters for each method that gave the best spiking sparsity with an up to *0.5%* allowed drop in validation accuracy, we report the testing results in Table 6.1 and Figure 6.1(a). Results when allowing an up to *1%* drop in validation accuracy are shown in Table 6.2 and Figure 6.1(b), and results for an up to *5%* drop are shown in Table 6.3 and Figure 6.1(c).

Note that baseline accuracy values are low because these preliminary results are on VGG-5 and for the reduced training set (with the validation set removed). Moving to larger networks and re-including the removed validation set into training (after hyperparameter selection) will significantly improve the baseline accuracy (Section 6.5).

Table 6.2. Best preliminary sparsity results for each method when allowing for an up to *1% drop* in validation accuracy for CIFAR-10 on a VGG-5 trained from scratch.

	Best Validation Hyperparameters		Validation Results		Testing Results	
Loss Function	σ_0	Training Epochs	Accuracy	Average Spikes	Accuracy	Average Spikes
CrossEntropy (baseline)	0	124	81.49%	698,579	82.17%	663,577
C.E. (baseline) w/ drop	0	76	80.82%	661,041	81.15%	635,610
C.E. + $\sigma_{constant}$	1.0e-05	78	80.58%	637,949	81.83%	366,718
C.E. + σ_{linear}	1.0e-06	125	80.58%	195,387	82.28%	177,737
C.E. + $\sigma_{quadratic}$	1.0e-08	125	80.50%	245,884	81.78%	224,648
C.E. + $\sigma_{alternate}$	0.002	124	80.52%	86,060	81.32%	76,932
C.E. + $\sigma_{alternate\_linear}$	5.0e-05	86	80.69%	85,753	80.57%	79,508
C.E. + σ_{on_off}	0.0027	121	80.67%	62,010	81.07%	63,369

Table 6.3. Best preliminary sparsity results for each method when allowing for an up to *5% drop* in validation accuracy for CIFAR-10 on a VGG-5 trained from scratch.

	Best Validation Hyperparameters		Validation Results		Testing Results	
Loss Function	σ_0	Training Epochs	Accuracy	Average Spikes	Accuracy	Average Spikes
CrossEntropy (baseline)	0	124	81.49%	698,579	82.17%	663,577
C.E. (baseline) w/ drop	0	73	79.47%	658,835	80.32%	636,826
C.E. + $\sigma_{constant}$	1.65e-4	124	78.81%	141,011	80.41%	126,720
C.E. + σ_{linear}	1.0e-05	125	78.36%	91,881	78.66%	83,243
C.E. + $\sigma_{quadratic}$	4.0e-07	96	76.49%	74,813	76.25%	69,005
C.E. + $\sigma_{alternate}$	0.003	119	76.52%	62,967	77.21%	57,955
C.E. + $\sigma_{alternate\_linear}$	5.0e-05	124	79.32%	72,665	79.46%	69,570
C.E. + σ_{on_off}	0.0027	94	76.54%	54,642	77.98%	52,298

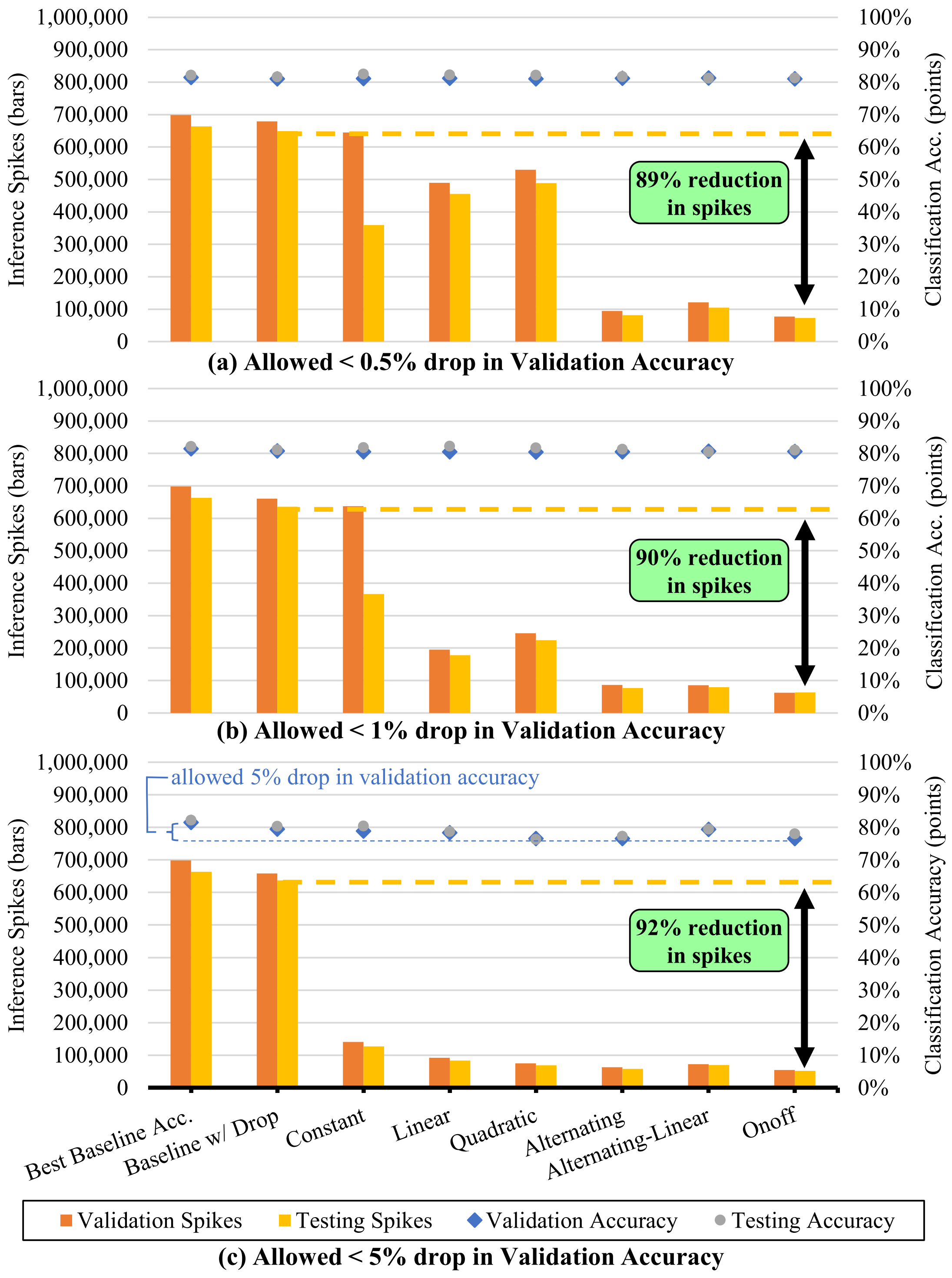


Figure 6.1. Preliminary ETSS trade-off results (CIFAR-10 on VGG-5, trained from scratch over 125 epochs). Results shown for the hyperparameter selection in each method that resulted in the fewest average validation spikes while maintaining the designated validation accuracy.

6.4 Comparison with Multi-Objective Particle Swarm Optimization

The above preliminary results were achieved with a very ad-hoc exploration of the hyperparameter space of the sparsity scalar. Before expanding to larger networks, we wanted to systematically explore how the sparsity scalar should change over time to observe any useful patterns that would reduce the amount of hyperparameter exploration needed in the costly temporal training of the larger networks. We performed this exploration with a modified Multi-Objective Particle Sward Optimization (MO-PSO).

PSO employs many workers or *particles* that simultaneously explore the search space. Each individual particle attempts to move toward a locally optimal solution, but is influenced in its direction toward a globally optimal solution by better solution states found by other particles. This is the *swarm* aspect of the optimization process.

6.4.1 MO-PSO methodology

Our process differs slightly from traditional PSO because of the temporal aspect of training. Not only does the specific hyperparameter(s) states that the particles have found make up their solution state, but also the amount of time or epochs that the particles have spent with the hyperparameter(s) at any previous value during previous epochs. Thus the solution state isn't just it's current hyperparameter choice, but also it's entire history.

We modify PSO to have Pareto-dominated particles be "killed off" and then clone a state that is branched off from another Pareto-optimal particle. From each Pareto-optimal point, we allowed for up to 6 different branches: (1) maintaining the current scalar value, (2) a proportionally slightly higher scalar, (3) a proportionally slightly smaller scalar, (4) maintaining the same momentum, i.e. letting the scalar continue to change in the direction it had changed previously, (5) maintaining momentum with a proportionally slightly higher scalar, and (6) maintaining momentum with a proportionally slightly smaller scalar. The allowed deviations were approximately 4%.

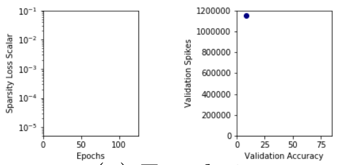
Our initial attempts with branching were very noisy, with many branches being killed by particles that had one statistically good epoch, but then were generally worse than the particles they killed off. We explored allowing branches from "grandfather" particles, averaging accuracy across branching generations, using training loss instead of validation accuracy, or requiring two sequential dominations to be killed off, but we ultimately found success with expanding the intervals to 5 epochs between evaluations/changes and keeping the same seed between sibling branches. It was also important for us to restrict our definition of Pareto-optimality to within a current generation so that child particles couldn't be dominated by the parent particles their branched from because of a single bad run.

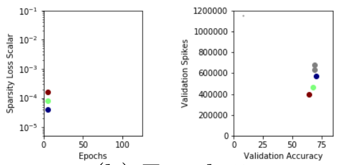
We started at six starting values (1E-5, 2E-5, 4E-5, 8E-5, 1.6E-4, and 3.2E-4), logarithmically in the range that we knew from the previous ad-hoc attempts wouldn't overpower accuracy in the first few epochs. We ran the entire process for 125 epochs, exploring every allowed non-dominated path along the way.

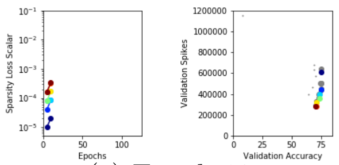
6.4.2 MO-PSO results and discussion

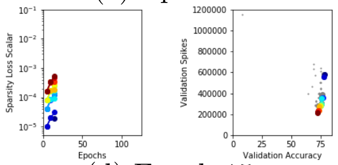
Figure 6.2 shows the MO-PSO results. There are three interesting observations that we see in the MO-PSO results. First, the final accuracy/sparsity trade-off was on par with the ad-hoc methods tried previously. Second, the sparsity scalar paths as expected started small and increased, but then after sparsity was maxed out, the more sparse nodes then began to branch off paths that once again decreased the scalar to focus on accuracy, and in so doing, surpassed and killed off the branches that had stayed lower. Third, more sparse paths seemed to follow the maximum increase allowed under the branching rules. While this implies that they were constrained by limited branching options, it would have been computationally infeasible to test the ability to jump to any arbitrary sparsity scalar at each branching point.

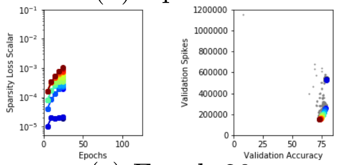
The fact, however, that the sparsity scalar could not start very high without failing and that the fact that the ad-hoc tests also failed with too rapid increases in the sparsity scalar both support the case for an initial period focused on training more for accuracy than sparsity. This is encouraging for training larger networks, because it means that we can bypass much of that training time by using pre-trained networks.

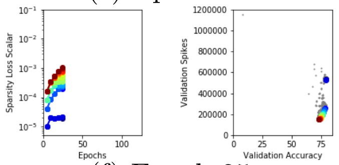

(a) Epoch 0

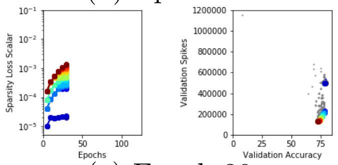

(b) Epoch 5

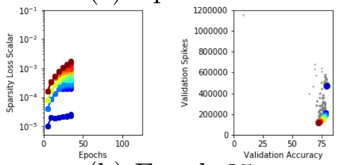

(c) Epoch 10

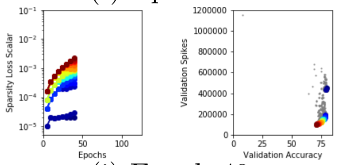

(d) Epoch 15

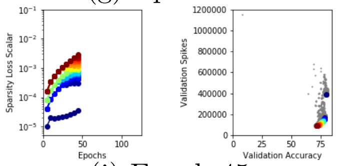

(e) Epoch 20

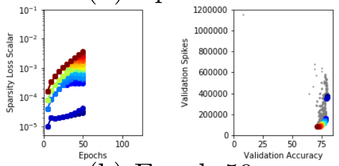

(f) Epoch 25

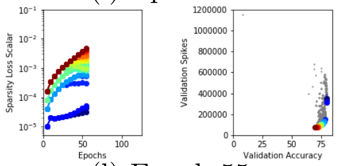

(g) Epoch 30

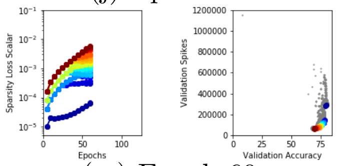

(h) Epoch 35

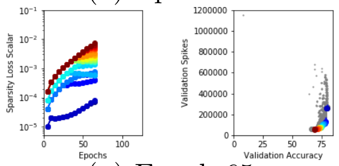

(i) Epoch 40

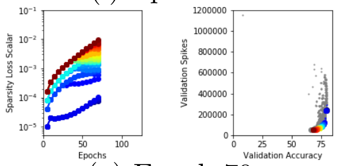

(j) Epoch 45

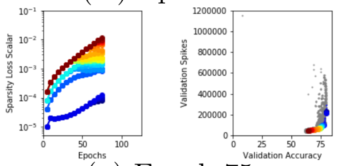

(k) Epoch 50

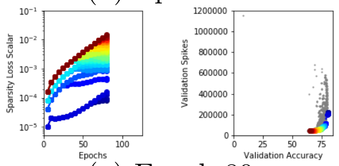

(l) Epoch 55

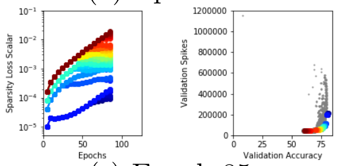

(m) Epoch 60

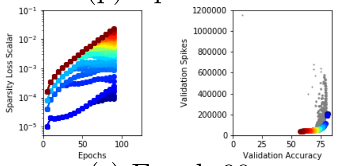

(n) Epoch 65

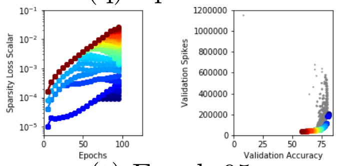

(o) Epoch 70

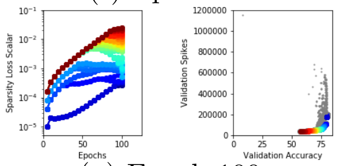

(p) Epoch 75

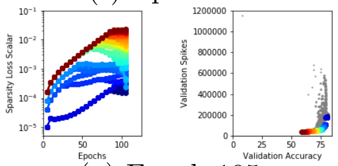

(q) Epoch 80

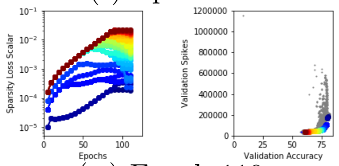

(r) Epoch 85

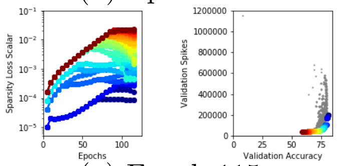

(s) Epoch 90

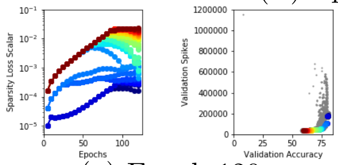

(t) Epoch 95


(u) Epoch 100


(v) Epoch 105


(w) Epoch 110


(x) Epoch 115


(y) Epoch 120

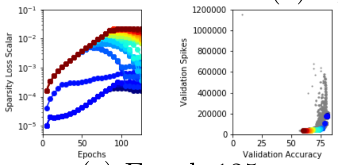

(z) Epoch 125

Figure 6.2. PSO. Left: path of surviving nodes. Right: corresponding trade-off.

6.5 Expanding to Larger Networks/Datasets

We expanded to RESNET-20 on CIFAR-10 and then VGG-11 on CIFAR-100. Because training in the time domain is very slow and memory-hungry, and because of the many versions we needed to train for each network, expanding to ImageNet was not within the scope of the computational resources we had available.

6.5.1 Starting with pre-trained networks

We expand to larger networks by beginning with pre-trained ANNs that are converted to SNNs. While the option of beginning with pre-trained SNNs was also a possibility, the addition of more spike-based training onto a network already trained in the spiking domain would cause overfitting to the training set. By transferring domains, there is still some spike-based training required, which training time we can use during the ETSS training before it overfits. This hybrid method of training (first a converted ANN followed by additional spike based training) is based on the methodology and code in [30], with the modifications of adding the spike count loss, as scaled by the sparsity scalar, to the classification loss used in the spike-based backpropagation. The domain transfer also allowed us to pre-train with the validation set included, and then pull it back out only during the spike-based training, increasing the baseline and overall accuracy.

We first tested this hybrid method on the previously explored VGG-5 on CIFAR-10 to verify that this shorted training process still allows for equivalent sparsity/accuracy trade-offs. Interestingly, we saw that even before any spike-based training, the baseline pre-trained and converted network operated with significantly lower spikes than the baseline that had only been trained in the spiking domain. In fact, the hybrid baseline was operating with sparsity on par with the best best ETSS systems trained previously.

An analysis of the weight magnitudes, input spike rates, and spike count processes narrowed down the discrepancy to the threshold balancing process during conversion. Because of the conversion process, the threshold to weight magnitude ratio for the first layer of the converted network was on the order of 10 times higher than the threshold to weight magnitude ratio of the SNNs trained only in the spiking domain. This difference resulted in an order of 1/10th the number of spikes propagated in the hybrid system. Thus, the conversion process included a form of weight regularization. This discovery helps us identify why we were able to previously achieve such a large reduction in spikes at near iso-accuracy. A majority of the pruned spikes were low-hanging fruit that could potentially be eliminated by that re-scaling alone. While this reset our baseline and reduces the amount of iso-accuracy sparsity improvement, we were still successful at further pushing the Pareto frontier with the subsequent ETSS training.

A benefit to the hybrid baseline's conversion-based sparsity is that the networks already undergo (1) a period of accuracy-only training, followed by (2) a significant activation pruning, which match the beginning and middle parts of the observed MO-PSO paths. With this as the new starting point for ETSS training, the main need for dynamically altering the sparsity scalar is reduced, and we can simply use a constant scalar to indicate the traded desired. Dynamically changing the scalar can still allow for a single training run to explore more area along the Pareto frontier, but a constant scalar can still push us to a corresponding trade-off points along the frontier.

6.5.2 Identifying Pareto-optimal points

The validation set is used to identify which sparsity scalars and epoch stopping points correspond with Pareto-optimal trade-offs. In other words, we do not use the testing data even to identify the Pareto-optimal points we wish to report. Only once those configurations are selected do we identify the testing results. This means that networks that were Pareto optimal in the validation set may not be Pareto optimal in the testing set, but in practice the match is very good. This, together with the inclusion of the validation set during pre-training, also means that an identified accuracy vs. sparsity trade-off in the validation set will not exactly correspond the an equivalent trade-off in the testing set, though in practice it is close, as shown next, and is sufficient to identify the better trade-offs.

6.5.3 ETSS pre-trained, expanded results

For Cifar-10, we train VGG-5 for 50 additional epochs and RESNET-20 for 20 additional epochs, and for Cifar-100 we train VGG-11 for 20 additional epochs. For each network, we trained for many constant sparsity scalars. Each of the results shown here are run and averaged over 3 seeds.

Cifar-10 on VGG-5

Figure 6.3 shows many iterations of training VGG-5 for different sparsity scalars, with the results pushing to the bottom right. Figure 6.4 shows the identification of the Pareto-optimal points according to the validation set and the corresponding testing set results. Figure 6.5 shows the spike count reduction for various levels of acceptable accuracy reduction, compared to the baseline network at equivalent levels of acceptable accuracy reduction.

Cifar-10 on RESNET-20

Figure 6.6 shows many iterations of training RESNET-20 for different sparsity scalars, with the results pushing to the bottom right. Figure 6.7 shows the identification of the Pareto-optimal points according to the validation set and the corresponding testing set results. Figure 6.8 shows the spike count reduction for various levels of acceptable validation reduction, compared to the baseline network at equivalent levels of acceptable accuracy reduction.

Cifar-100 on VGG-11

Figure 6.9 shows many iterations of training VGG-11 on CIFAR-100 for different sparsity scalars, with the results pushing to the bottom right. Figure 6.10 shows the identification of the Pareto-optimal points according to the validation set and the corresponding testing set results. Figure 6.11 shows the spike count reduction for various levels of acceptable accuracy reduction, compared to the baseline network at equivalent levels of acceptable accuracy reduction.

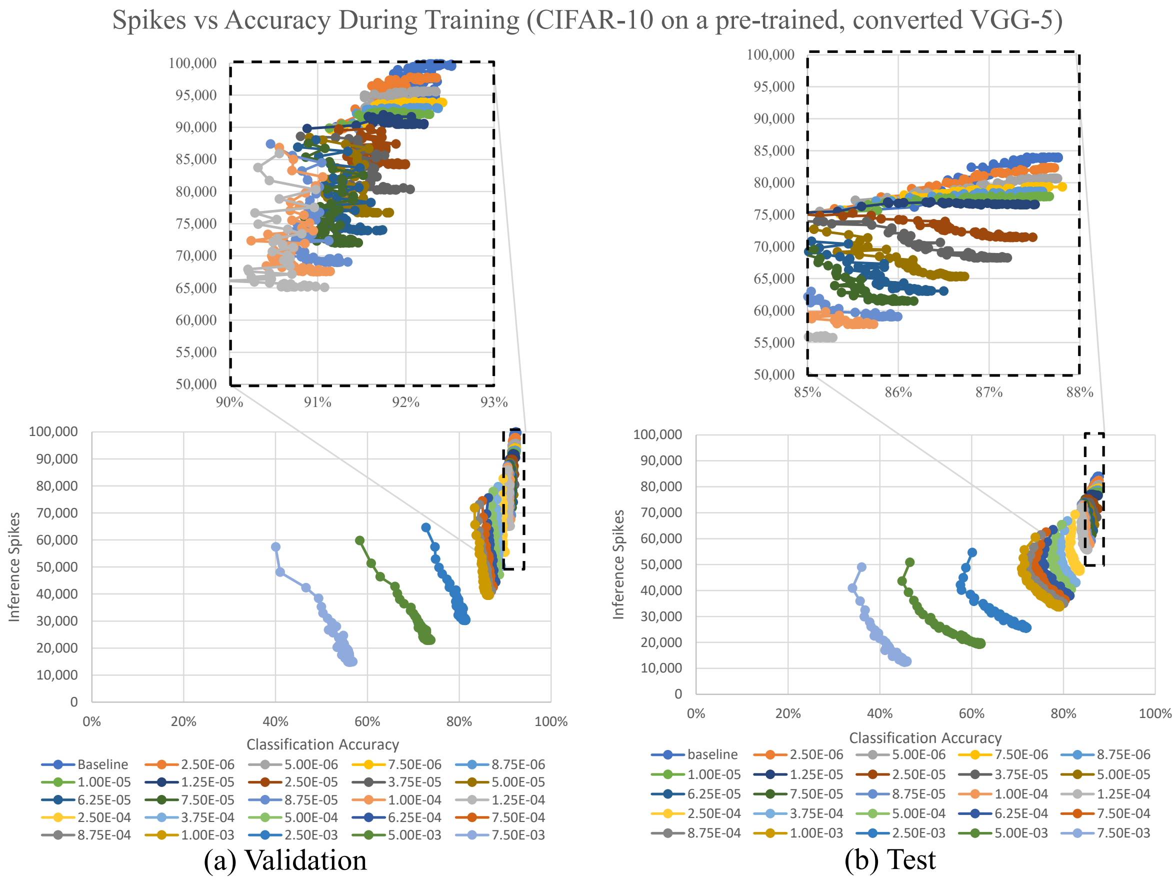


Figure 6.3. A converted VGG-5 network trained for an additional 50 epochs with spike-based training, comparing the baseline (no ETSS), to various constant ETSS methods with their sparsity scalars shown. To the bottom-right is better. Connected dots show the progress during training as they push to the right and down. (a) Validation. (b) Testing.

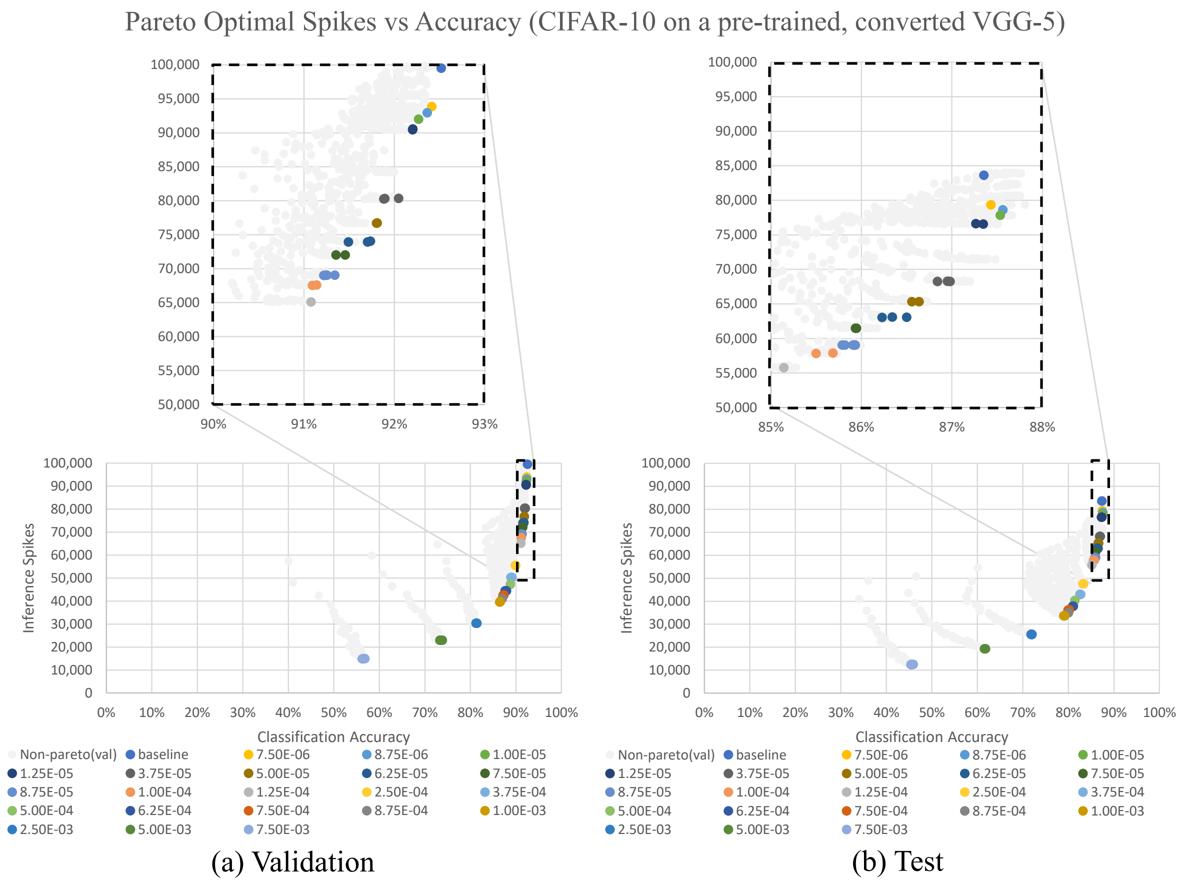
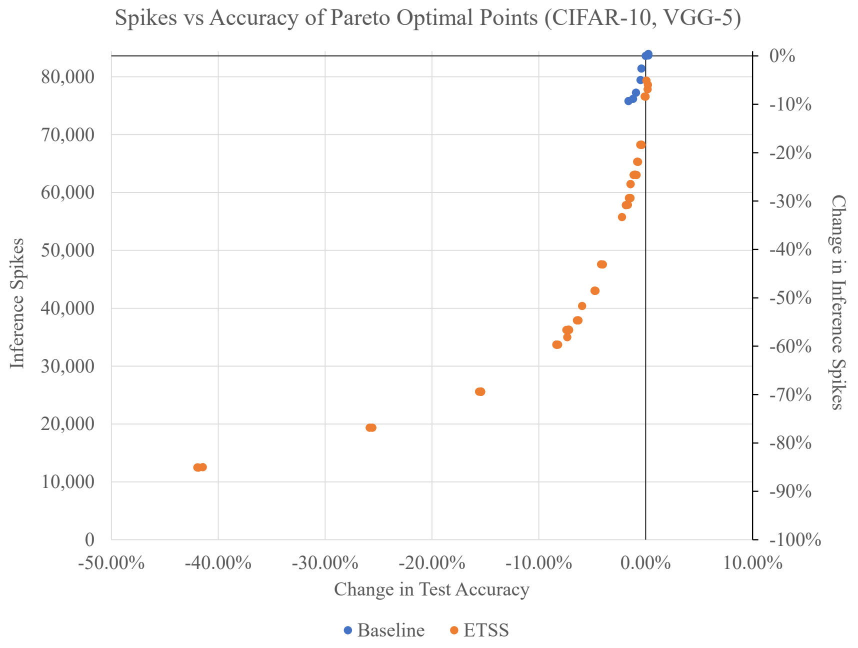
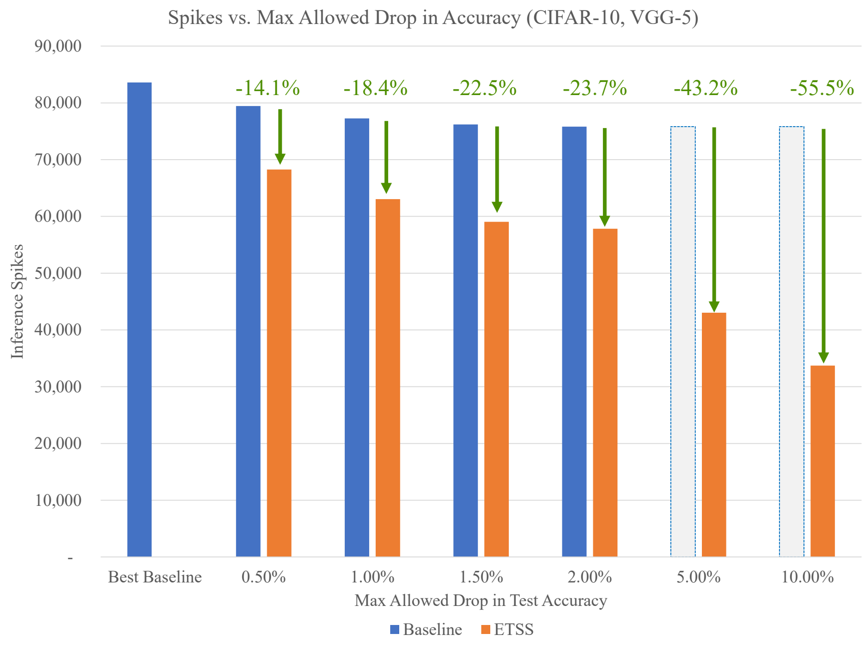


Figure 6.4. The same data as in Figure 6.3 with the Pareto-optimal points identified (according to the validation set). (a) Validation. (b) Testing.

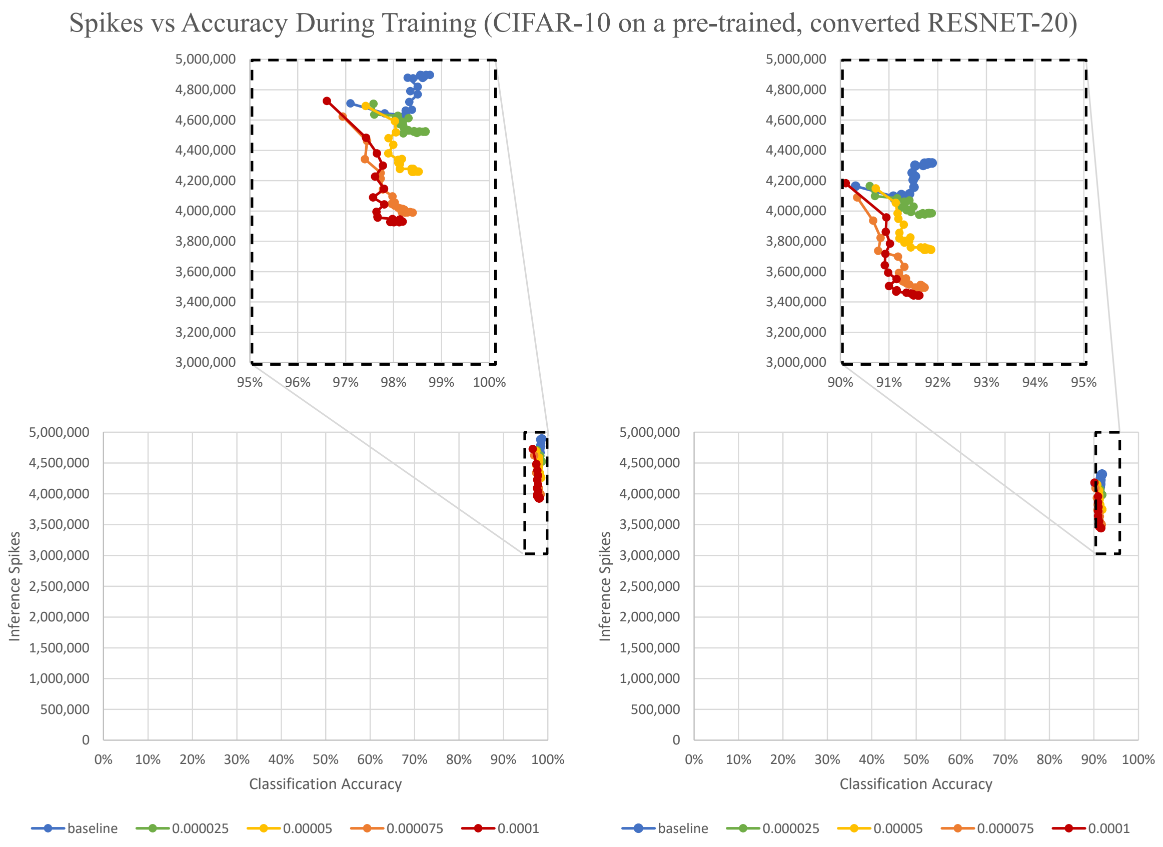


(a) All Pareto-optimal points.



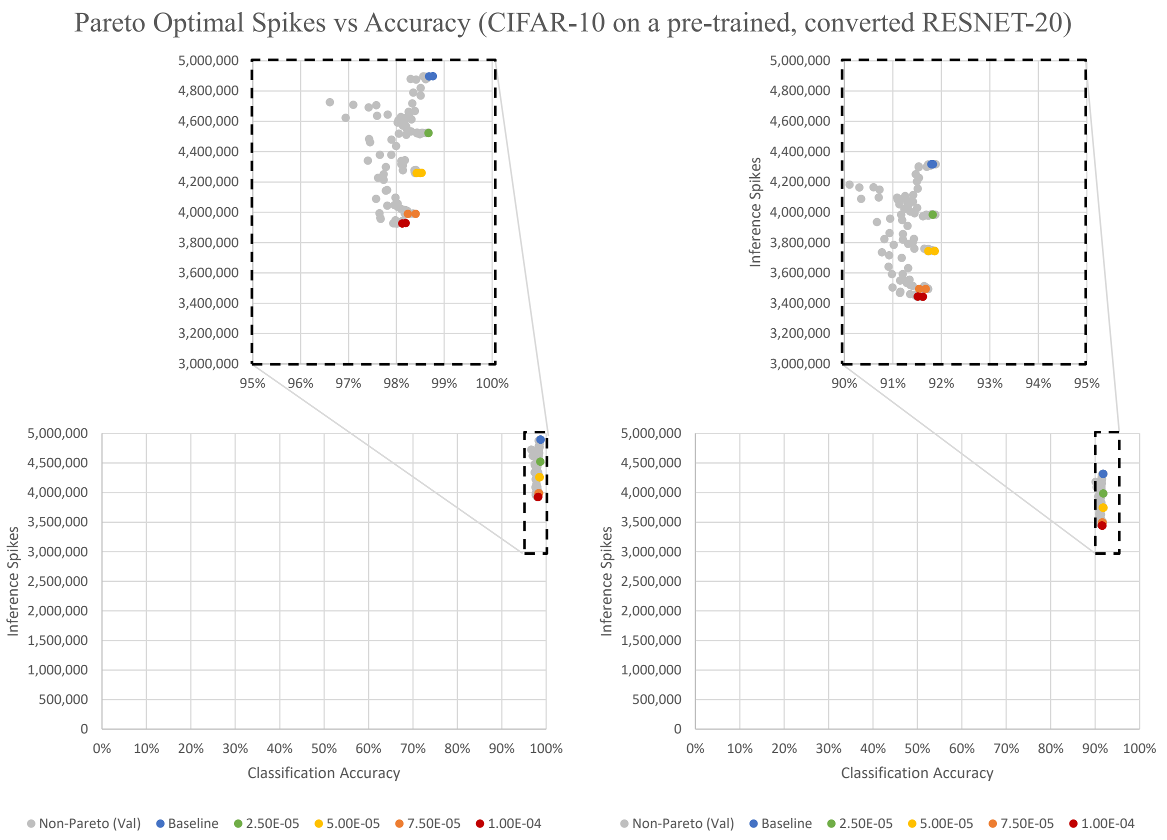
(b) Points with lowest spiking activity in a given window of allowed accuracy degradation.

Figure 6.5. Comparison of Pareto-optimal ETSS constant-scalar trade-offs with Pareto-optimal baseline trade-offs for CIFAR-10 on VGG-5. (Note that the pre-trained baseline never had accuracy less than 2% below its final best. Therefore, its lowest spike state is chosen as a comparison in the larger bins–represented with grayed out bars.)



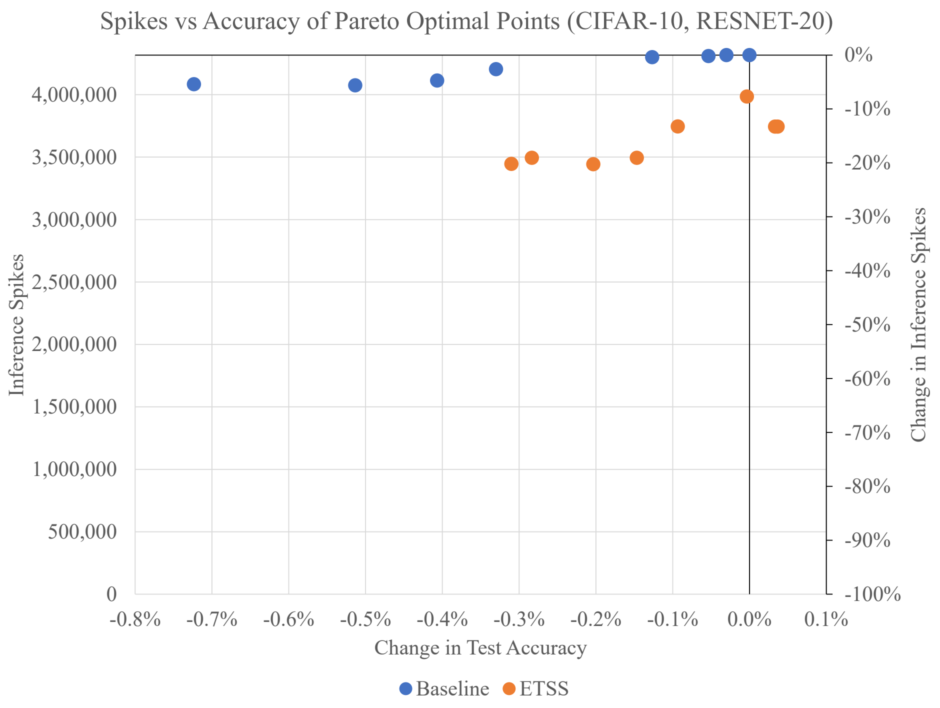
(a) Validation (b) Test

Figure 6.6. A converted RESNET-20 network trained for an additional 20 epochs with spike-based training, comparing the baseline (no ETSS), to various constant ETSS methods with their sparsity scalars shown. To the bottom-right is better. Connected dots show the progress during training as they push to the right and down. (a) Validation. (b) Testing.

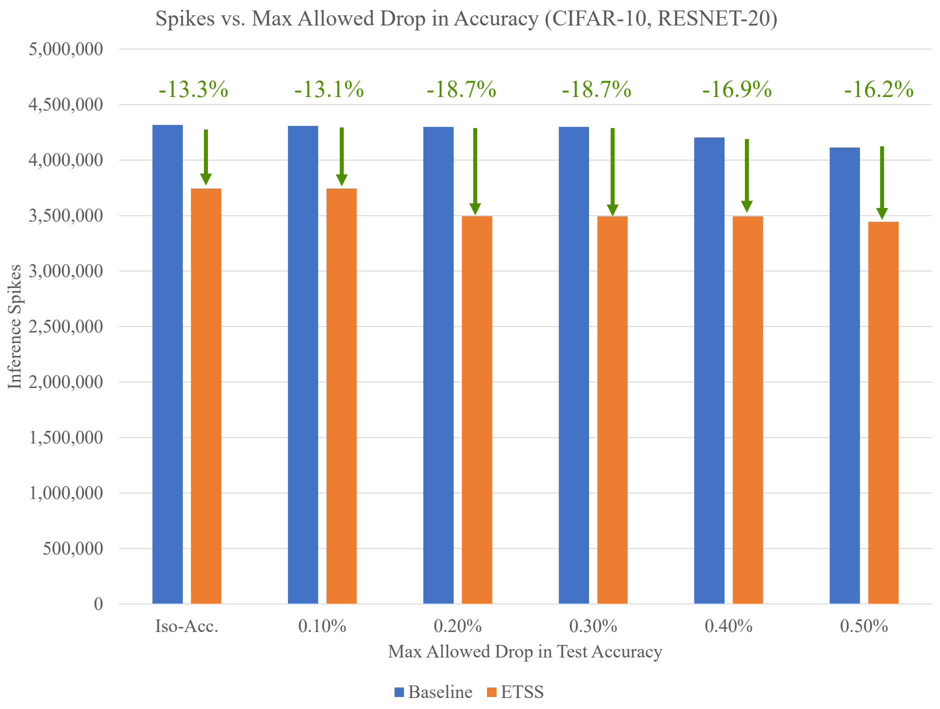


(a) Validation (b) Test

Figure 6.7. The same data as in Figure 6.6 with the Pareto-optimal points identified (according to the validation set). (a) Validation. (b) Testing.



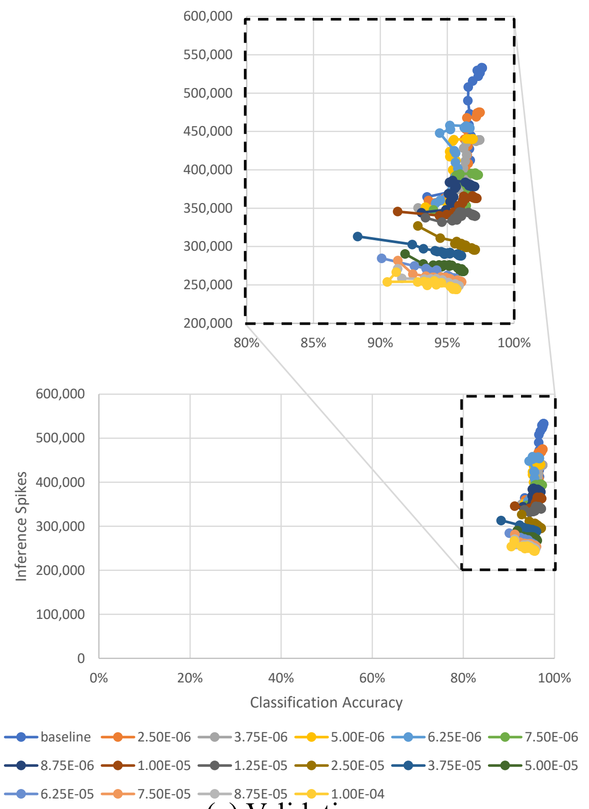
(a) All Pareto-optimal points.



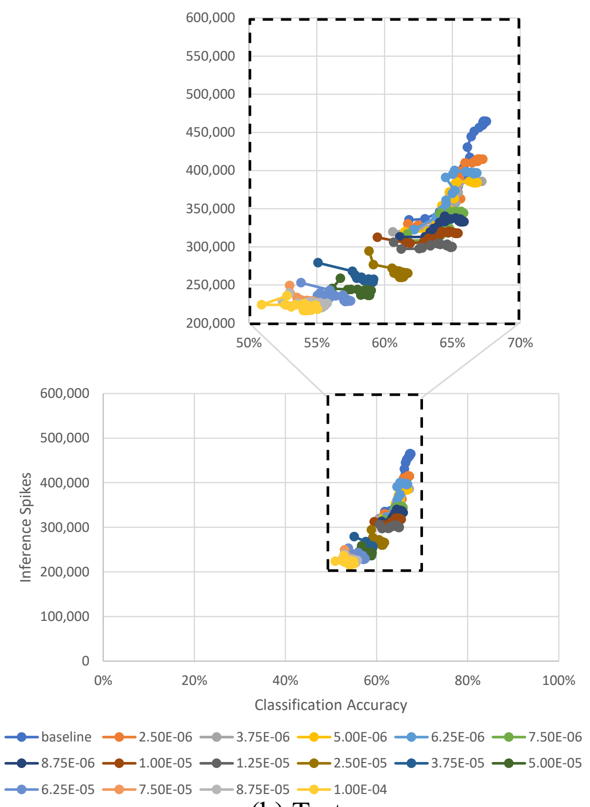
(b) Points with lowest spiking activity in a given window of allowed accuracy degradation.

Figure 6.8. Comparison of Pareto-optimal ETSS constant-scalar trade-offs with Pareto-optimal baseline trade-offs for CIFAR-10 on RESNET-20.

Spikes vs Accuracy During Training (CIFAR-100 on a pre-trained, converted VGG-11)



(a) Validation



(b) Test

Figure 6.9. A converted VGG-11 network trained for an additional 20 epochs with spike-based training, comparing the baseline (no ETSS), to various constant ETSS methods with their sparsity scalars shown. To the bottom-right is better. Connected dots show the progress during training as they push to the right and down. (a) Validation. (b) Testing.

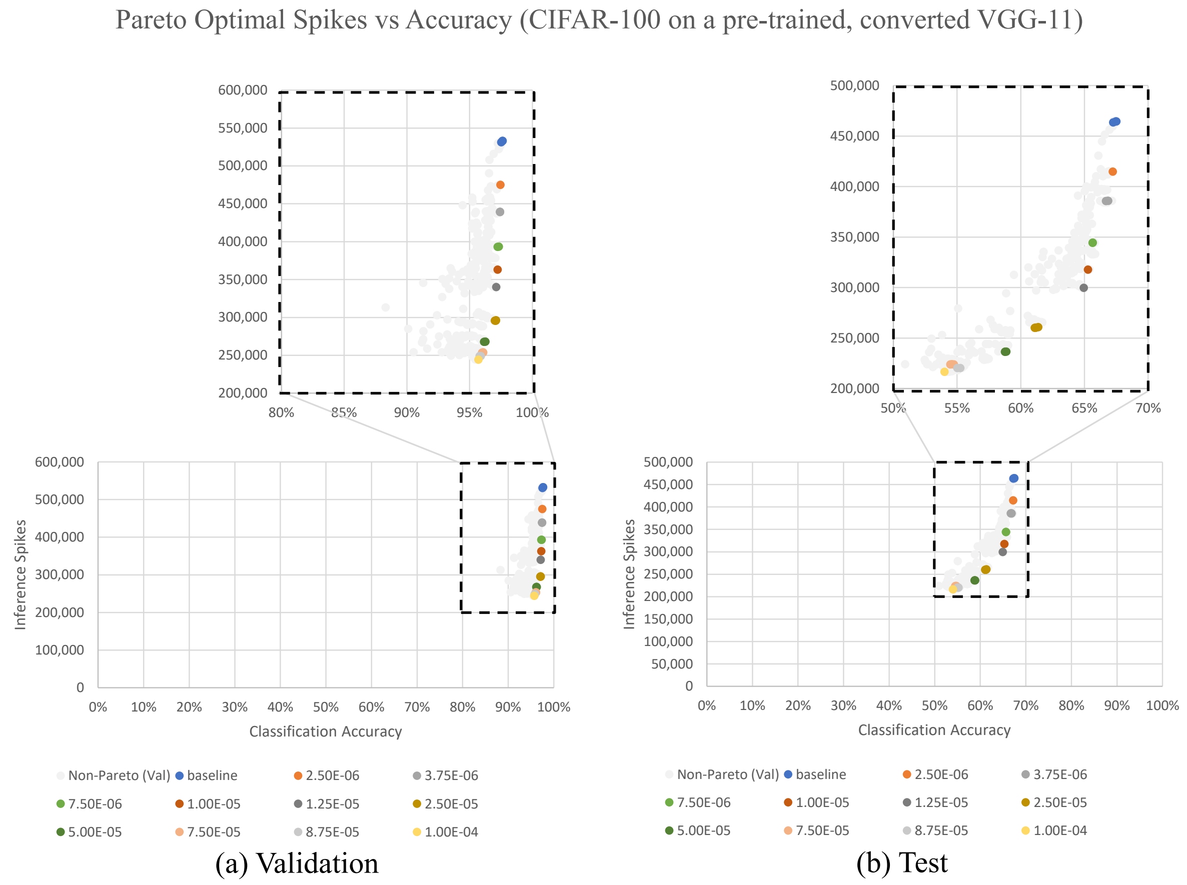
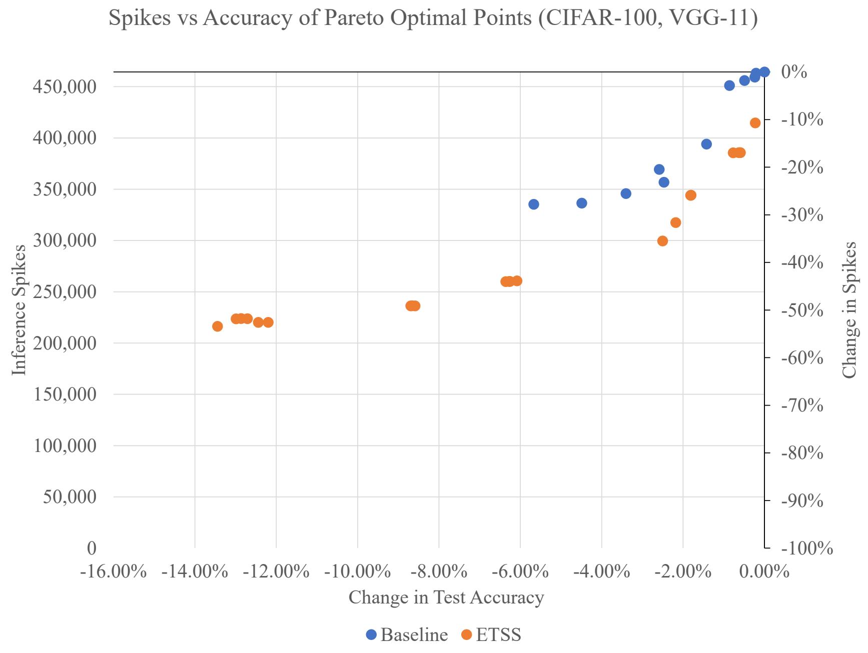
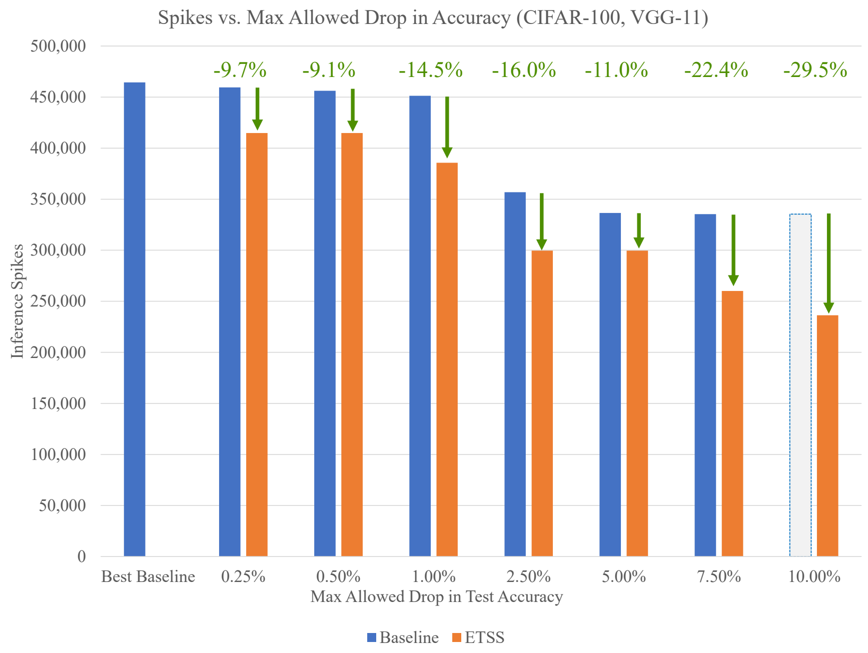


Figure 6.10. The same data as in Figure 6.9 with the Pareto-optimal points identified (according to the validation set). (a) Validation. (b) Testing.



(a) All Pareto-optimal points.



(b) Points with lowest spiking activity in a given window of allowed accuracy degradation.

Figure 6.11. Comparison of Pareto-optimal ETSS constant-scalar trade-offs with Pareto-optimal baseline trade-offs for CIFAR-100 on VGG-11. (Note that the pre-trained baseline never had accuracy less than 7.5% below its final best. Therefore, its lowest spike state is chosen as a comparison in the 10% drop bins–represented with a grayed out bar.)

6.5.4 Dynamic runs for comparison

Although we were able to get sufficient trade-off using constant sparsity scalars on larger, pre-trained networks, we still need to compare with dynamically adjusting the sparsity scalar during training. Because RESNET-20 and VGG-11 take a considerably larger amount of time and memory to train with temporal, spike-based backpropagation, not very many hyperparameters could be searched. We therefore replace the sparsity scalar functions tried previously in the preliminary results with a feedback function that only includes the sparsity loss in the current epoch *if* the current validation accuracy is within 1% of the max validation accuracy it has seen up until that point. Whenever the validation accuracy falls below that level, the dynamic network will switch to training only for classification error minimization to regain the lost accuracy.

We compare a few runs of this dynamic sparsity scalar method at different intensities against the Pareto-optimal points of the constant scalars. Figure 6.12 shows that trade-off comparison for CIFAR-10 on a pre-trained VGG-5, Figure 6.13 shows comparison for CIFAR-10 on a pre-trained RESNET-20, and Figure 6.14 shows the comparison for CIFAR-100 on a pre-trained VGG-11.

For the smaller VGG-5, the dynamic methods are able to push up to the Pareto-boundary, but do not significantly pass it with the testing set. However, for the larger RESNET-20, the less-intense dynamic function (orange) was able to drop the spiking activity significantly lower than the constant approaches. Stricter constant scalars in that area had all failed. Similarly, for VGG-11 the dynamic functions were able to push past the constants' Pareto frontier, in some places regaining 3-5% accuracy at a given spike count level.

For the larger networks, a constant scalar may be insufficient because the multi-objective loss landscape may continue to contain far more local minima with their higher parameter counts, even after pre-training. Further hyperparameter search in this space was, however, beyond the scope of this work due to computational constraints. This exploration merits further research.

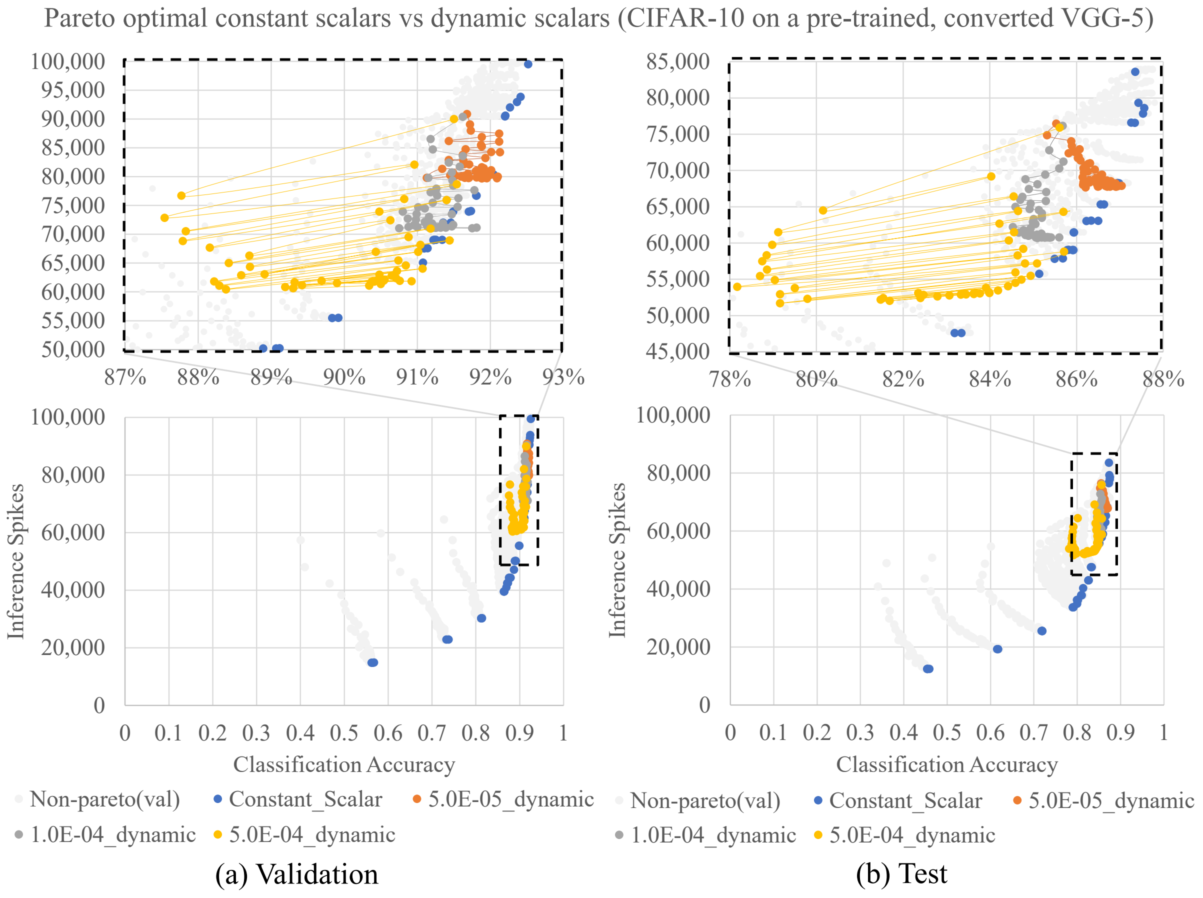


Figure 6.12. Trade-off results of a few dynamic scalar runs compared to the constant scalar pareto-optimal points for CIFAR-10 on a pre-trained VGG-5.

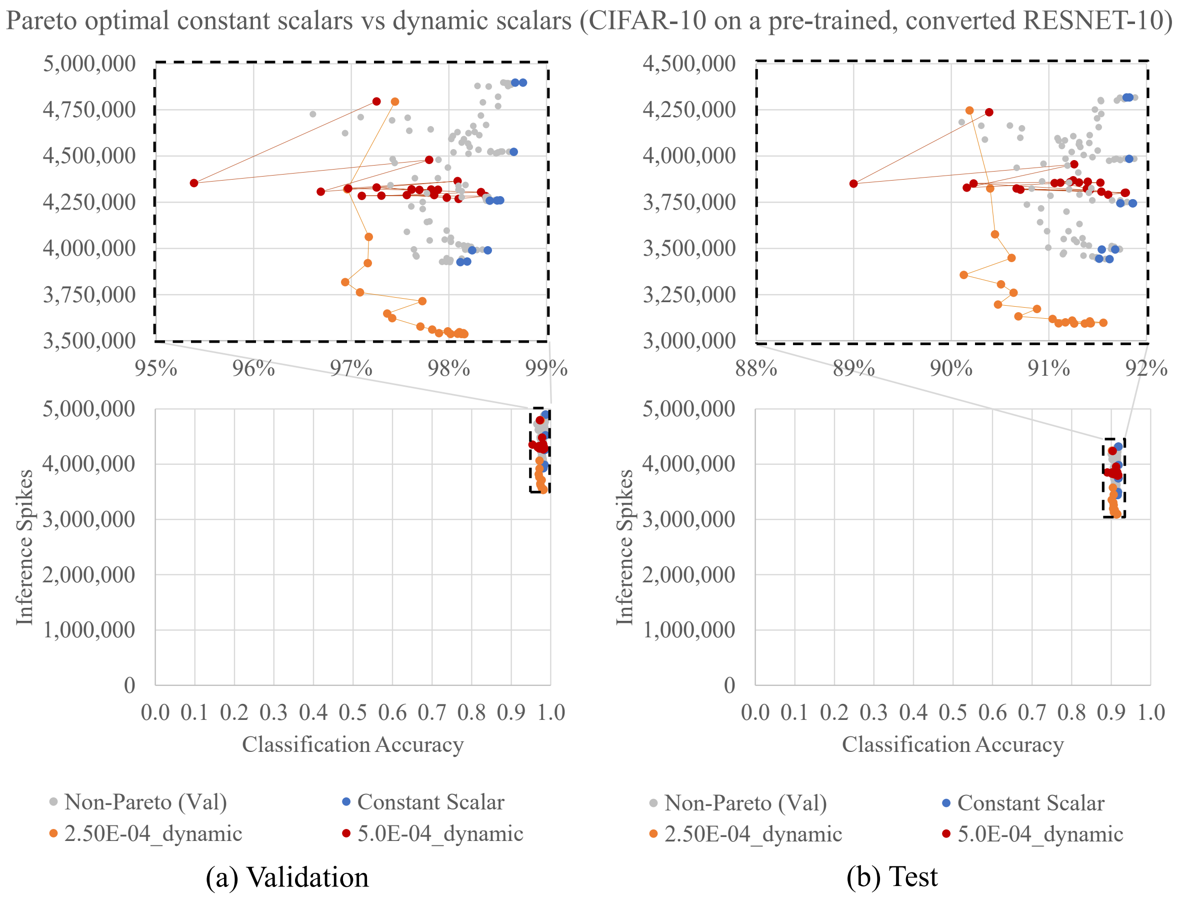


Figure 6.13. Trade-off results of a couple dynamic scalar runs compared to the constant scalar pareto-optimal points for CIFAR-10 on a pre-trained RESNET-20.

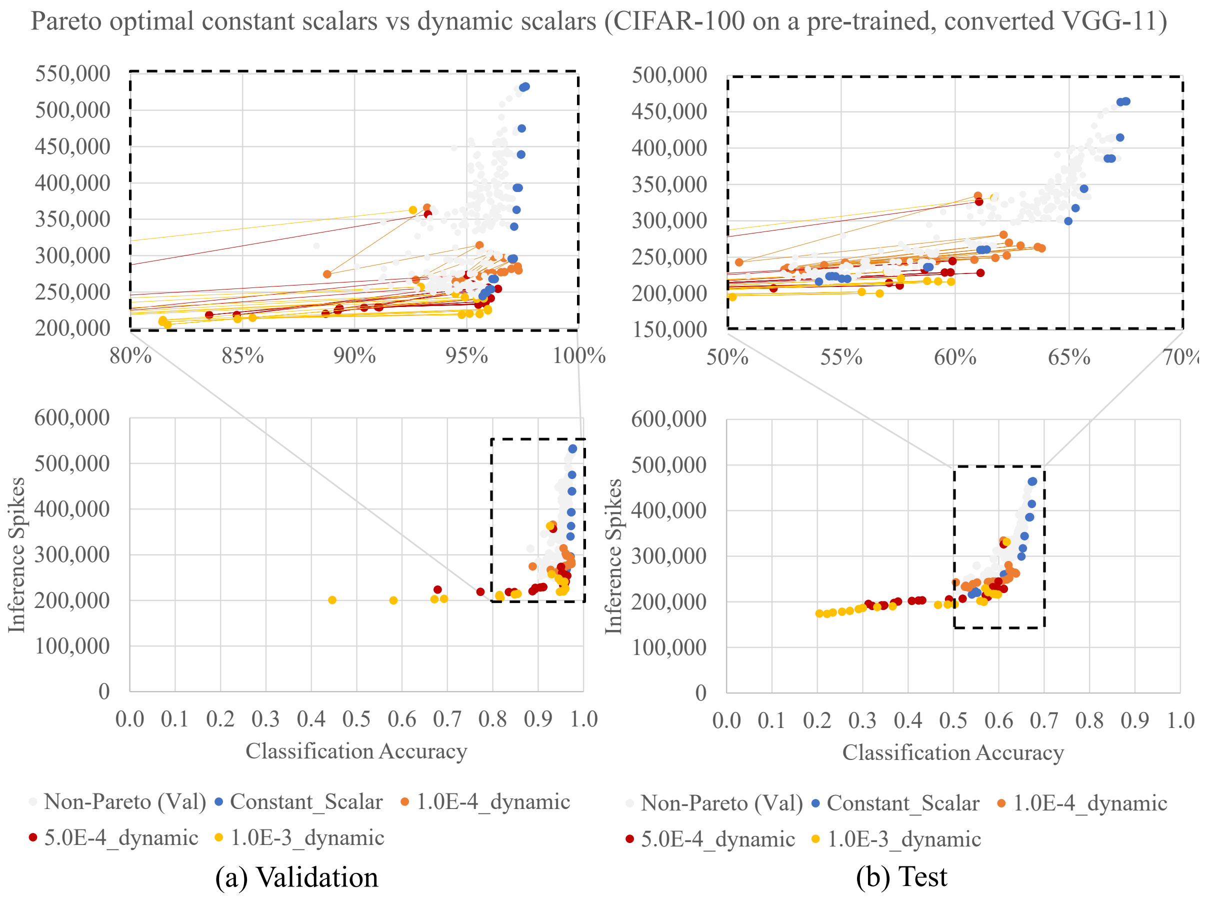


Figure 6.14. Trade-off results of a few dynamic scalar runs compared to the constant scalar pareto-optimal points for CIFAR-100 on a pre-trained VGG-11.

REFERENCES

[1] Y. Lecun, “Generalization and network design strategies,” *Connectionism in Perspective*, 1989.

[2] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[3] S. J. Nowlan and G. E. Hinton, “Simplifying neural networks by soft weight-sharing,” *Neural Computation*, vol. 4, no. 4, pp. 473–493, 1992. DOI: 10.1162/neco.1992.4.4.473. eprint: https://doi.org/10.1162/neco.1992.4.4.473. [Online]. Available: https://doi.org/10.1162/neco.1992.4.4.473.

[4] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, “Handwritten digit recognition with a back-propagation network,” in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed., Morgan-Kaufmann, 1990, pp. 396–404. [Online]. Available: http://papers.nips.cc/paper/293-handwritten-digit-recognition-with-a-back-propagation-network.pdf.

[5] Y. Lecun and Y. Bengio, “Convolutional networks for images, speech, and time series,” *The handbook of brain theory and neural networks*, 1995.

[6] D. Scherer, A. Müller, and S. Behnke, “Evaluation of pooling operations in convolutional architectures for object recognition,” in *Artificial Neural Networks – ICANN 2010*, K. Diamantaras, W. Duch, and L. S. Iliadis, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 92–101, ISBN: 978-3-642-15825-4.

[7] W. Maass, “Networks of spiking neurons: The third generation of neural network models,” *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997, ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(97)00011-7. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608097000117.

[8] S. Ghosh-Dastidar and H. Adeli, “Spiking neural networks,” *International Journal of Neural Systems*, vol. 19, no. 04, pp. 295–308, 2009, PMID: 19731402. DOI: 10.1142/S0129065709002002. eprint: https://doi.org/10.1142/S0129065709002002. [Online]. Available: https://doi.org/10.1142/S0129065709002002.

[9] B. Han, A. Ankit, A. Sengupta, and K. Roy, “Cross-layer design exploration for energy-quality tradeoffs in spiking and non-spiking deep artificial neural networks,” *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 4, pp. 613–623, Oct. 2018, ISSN: 2332-7766. DOI: 10.1109/TMSCS.2017.2737625.

[10] A. Sengupta, Z. Al Azim, X. Fong, and K. Roy, "Spin-orbit torque induced spike-timing dependent plasticity," *Applied Physics Letters*, vol. 106, no. 9, p. 093 704, 2015. DOI: 10.1063/1.4914111. eprint: https://doi.org/10.1063/1.4914111. [Online]. Available: https://doi.org/10.1063/1.4914111.

[11] M. Rahimi Azghadi, N. Iannella, S. Al-Sarawi, and D. Abbott, "Tunable low energy, compact and high performance neuromorphic circuit for spike-based synaptic plasticity," *PLOS ONE*, vol. 9, no. 2, pp. 1–14, Feb. 2014. DOI: 10.1371/journal.pone.0088326. [Online]. Available: https://doi.org/10.1371/journal.pone.0088326.

[12] J. M. Allred and K. Roy, "Controlled forgetting: Targeted stimulation and dopaminergic plasticity modulation for unsupervised lifelong learning in spiking neural networks," *Frontiers in Neuroscience*, vol. 14, p. 7, 2020, ISSN: 1662-453X. DOI: 10.3389/fnins.2020.00007. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2020.00007.

[13] J. M. Allred and K. Roy, "L4-Norm Weight Adjustments for Converted Spiking Neural Networks," *manuscript in preparation*, 2021.

[14] J. M. Allred and K. Roy, "Unsupervised incremental stdp learning using forced firing of dormant or idle neurons," in *2016 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2016, pp. 2492–2499. DOI: 10.1109/IJCNN.2016.7727509.

[15] P. Panda, J. M. Allred, S. Ramanathan, and K. Roy, "Asp: Learning to forget with adaptive synaptic plasticity in spiking neural networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 51–64, Mar. 2018, ISSN: 2156-3357. DOI: 10.1109/JETCAS.2017.2769684.

[16] Z. Zhang, S. Mondal, S. Mandal, J. M. Allred, N. A. Aghamiri, A. Fali, Z. Zhang, H. Zhou, H. Cao, F. Rodolakis, J. L. McChesney, Q. Wang, Y. Sun, Y. Abate, K. Roy, K. M. Rabe, and S. Ramanathan, "Neuromorphic learning with Mott insulator NiO," *Proceedings of the National Academy of Sciences*, to appear.

[17] J. M. Allred and K. Roy, "Convolving over time via recurrent connections for sequential weight sharing in neural networks," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 4444–4450.

[18] J. M. Allred, S. J. Spencer, G. Srinivasan, and K. Roy, "Explicitly Trained Spiking Sparsity in Spiking Neural Networks with Backpropagation," *arXiv e-prints, update to appear*, arXiv:2003.01250, arXiv:2003.01250, Mar. 2020. arXiv: 2003.01250 `[cs.NE]`.

[19] D. Goodman and R. Brette, "The brian simulator," *Frontiers in Neuroscience*, vol. 3, p. 26, 2009, ISSN: 1662-453X. DOI: 10.3389/neuro.01.026.2009. [Online]. Available: https://www.frontiersin.org/article/10.3389/neuro.01.026.2009.

[20] P. Diehl, “Stdp-mnist,” *Github*, 2015. [Online]. Available: https://github.com/peter-u-diehl/stdp-mnist.

[21] F. Zenke, “Spytorch,” *Github*, Mar. 2019. [Online]. Available: https://github.com/fzenke/spytorch.

[22] P. Diehl and M. Cook, “Unsupervised learning of digit recognition using spike-timing-dependent plasticity,” *Frontiers in Computational Neuroscience*, vol. 9, p. 99, 2015, ISSN: 1662-5188. DOI: 10.3389/fncom.2015.00099. [Online]. Available: https://www.frontiersin.org/article/10.3389/fncom.2015.00099.

[23] D. Querlioz, O. Bichler, P. Dollfus, and C. Gamrat, “Immunity to device variations in a spiking neural network with memristive nanodevices,” *IEEE Transactions on Nanotechnology*, vol. 12, no. 3, pp. 288–295, 2013.

[24] Y. Dan and M. Poo, “Hebbian depression of isolated neuromuscular synapses in vitro,” *Science*, vol. 256, no. 5063, pp. 1570–1573, 1992, ISSN: 0036-8075. DOI: 10.1126/science.1317971. eprint: https://science.sciencemag.org/content/256/5063/1570.full.pdf. [Online]. Available: https://science.sciencemag.org/content/256/5063/1570.

[25] N. Hohn and A. Burkitt, “Shot noise in the leaky integrate-and-fire neuron,” *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 63, p. 031 902, Apr. 2001. DOI: 10.1103/PhysRevE.63.031902.

[26] E. Oja, “Simplified neuron model as a principal component analyzer,” *Journal of Mathematical Biology*, vol. 15, no. 3, pp. 267–273, Nov. 1982, ISSN: 1432-1416. DOI: 10.1007/BF00275687. [Online]. Available: https://doi.org/10.1007/BF00275687.

[27] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, ISSN: 0018-9219.

[28] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. Liu, and M. Pfeiffer, “Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing,” in *2015 International Joint Conference on Neural Networks (IJCNN)*, 2015, pp. 1–8.

[29] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, “Going deeper in spiking neural networks: Vgg and residual architectures,” *Frontiers in Neuroscience*, vol. 13, p. 95, 2019, ISSN: 1662-453X. DOI: 10.3389/fnins.2019.00095. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2019.00095.

[30] N. Rathi, G. Srinivasan, P. Panda, and K. Roy, "Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=B1xSperKvH.

[31] S. Grossberg, "Competitive learning: From interactive activation to adaptive resonance," *Cognitive Science*, vol. 11, no. 1, pp. 23–63, 1987, ISSN: 0364-0213. DOI: 10.1016/S0364-0213(87)80025-3. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0364021387800253.

[32] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, "Overcoming catastrophic forgetting by incremental moment matching," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., 2017, pp. 4652–4662. [Online]. Available: http://papers.nips.cc/paper/7051-overcoming-catastrophic-forgetting-by-incremental-moment-matching.pdf.

[33] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive Neural Networks," *arXiv e-prints*, arXiv: 1606.04671, arXiv: 1606.04671, Jun. 2016. arXiv: 1606.04671 [cs.LG].

[34] R. K. Srivastava, J. Masci, S. Kazerounian, F. Gomez, and J. Schmidhuber, "Compete to compute," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2013, pp. 2310–2318. [Online]. Available: http://papers.nips.cc/paper/5059-compete-to-compute.pdf.

[35] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. e. Pritzel, and D. Wierstra, "PathNet: Evolution Channels Gradient Descent in Super Neural Networks," *arXiv e-prints*, arXiv:1701.08734, arXiv:1701.08734, Jan. 2017. arXiv: 1701.08734 [cs.NE].

[36] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 12, pp. 2935–2947, Dec. 2018, ISSN: 0162-8828. DOI: 10.1109/TPAMI.2017.2773081.

[37] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3521–3526, 2017, ISSN: 0027-8424. eprint: https://www.pnas.org/content/114/13/3521.full.pdf. [Online]. Available: https://www.pnas.org/content/114/13/3521.

[38] P. Bashivan, M. Schrimpf, R. Ajemian, I. Rish, M. Riemer, and Y. Tu, "Continual Learning with Self-Organizing Maps," *arXiv e-prints*, arXiv: 1904.09330, arXiv: 1904.09330, Apr. 2019. arXiv: 1904.09330 [cs.NE].

[39] J. Wang, A. Belatreche, L. Maguire, and T. M. McGinnity, "Dynamically evolving spiking neural network for pattern recognition," in *2015 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2015, pp. 1–8. DOI: 10.1109/IJCNN.2015.7280649.

[40] J. Wang, A. Belatreche, L. Maguire, and T. M. McGinnity, "An online supervised learning method for spiking neural networks with adaptive structure," *Neurocomputing*, vol. 144, pp. 526–536, 2014, ISSN: 0925-2312. DOI: 10.1016/j.neucom.2014.04.017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231214005785.

[41] S. G. Wysoski, L. Benuskova, and N. Kasabov, "On-line learning with structural adaptation in a network of spiking neurons for visual pattern recognition," in *Proceedings of the 16th International Conference on Artificial Neural Networks - Volume Part I*, ser. ICANN'06, Athens, Greece: Springer-Verlag, 2006, pp. 61–70, ISBN: 978-3-540-38625-4. DOI: 10.1007/11840817_7. [Online]. Available: http://dx.doi.org/10.1007/11840817%5C_7.

[42] X. Du, G. Charan, F. Liu, and Y. Cao, "Single-Net Continual Learning with Progressive Segmented Training (PST)," *arXiv e-prints*, arXiv:1905.11550, arXiv:1905.11550, May 2019. arXiv: 1905.11550 [cs.LG].

[43] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., Cham: Springer International Publishing, 2018, pp. 144–161, ISBN: 978-3-030-01219-9.

[44] A. Rios and L. Itti, "Closed-Loop Memory GAN for Continual Learning," *arXiv e-prints*, arXiv:1811.01146, arXiv:1811.01146, Nov. 2018. arXiv: 1811.01146 [cs.LG].

[45] J. Wang, A. Belatreche, L. P. Maguire, and T. M. McGinnity, "Spiketemp: An enhanced rank-order-based learning approach for spiking neural networks with adaptive structure," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 1, pp. 30–43, Jan. 2017, ISSN: 2162-237X. DOI: 10.1109/TNNLS.2015.2501322.

[46] K. Dhoble, N. Nuntalid, G. Indiveri, and N. Kasabov, "Online spatio-temporal pattern recognition with evolving spiking neural networks utilising address event representation, rank order, and temporal spike learning," in *The 2012 International Joint Conference on Neural Networks (IJCNN)*, Jun. 2012, pp. 1–7. DOI: 10.1109/IJCNN.2012.6252439.

[47] H. P. Op de Beeck, J. A. Deutsch, W. Vanduffel, N. G. Kanwisher, and J. J. DiCarlo, "A Stable Topography of Selectivity for Unfamiliar Shape Classes in Monkey Inferior Temporal Cortex," *Cerebral Cortex*, vol. 18, no. 7, pp. 1676–1694, Nov. 2007, ISSN: 1047-3211. DOI: 10.1093/cercor/bhm196. eprint: http://oup.prod.sis.lan/cercor/article-pdf/18/7/1676/17299757/bhm196.pdf. [Online]. Available: https://doi.org/10.1093/cercor/bhm196.

[48] M. J. Lee and J. J. DiCarlo, "Comparing novel object learning in humans, models, and monkeys," *Journal of Vision*, vol. 19, no. 10, 114b, Sep. 2019. DOI: 10.1167/19.10.114b.

[49] Hailin Jiang, M. Marek-Sadowska, and S. R. Nassif, "Benefits and costs of power-gating technique," in *2005 International Conference on Computer Design*, 2005, pp. 559–566.

[50] N. Frémaux and W. Gerstner, "Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules," *Frontiers in Neural Circuits*, vol. 9, p. 85, 2016, ISSN: 1662-5110. DOI: 10.3389/fncir.2015.00085. [Online]. Available: https://www.frontiersin.org/article/10.3389/fncir.2015.00085.

[51] S. El-Boustani, J. P. K. Ip, V. Breton-Provencher, G. W. Knott, H. Okuno, H. Bito, and M. Sur, "Locally coordinated synaptic plasticity of visual cortex neurons in vivo," *Science*, vol. 360, no. 6395, pp. 1349–1354, 2018, ISSN: 0036-8075. eprint: https://science.sciencemag.org/content/360/6395/1349.full.pdf. [Online]. Available: https://science.sciencemag.org/content/360/6395/1349.

[52] F. Zuo, P. Panda, M. Kotiuga, J. Li, M. Kang, C. Mazzoli, H. Zhou, A. Barbour, S. Wilkins, B. Narayanan, *et al.*, "Habituation based synaptic plasticity and organismic learning in a quantum perovskite," *Nature communications*, vol. 8, no. 1, pp. 1–7, 2017.

[53] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, L. M. L. Cam and J. Neyman, Eds., vol. 1, University of California Press, 1967, pp. 281–297.

[54] G.-q. Bi and M.-m. Poo, "Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type," *Journal of Neuroscience*, vol. 18, no. 24, pp. 10 464–10 472, 1998, ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.18-24-10464.1998. eprint: https://www.jneurosci.org/content/18/24/10464.full.pdf. [Online]. Available: https://www.jneurosci.org/content/18/24/10464.

[55] C. Lee, G. Srinivasan, P. Panda, and K. Roy, "Deep spiking convolutional neural network trained with unsupervised spike-timing-dependent plasticity," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 11, pp. 384–394, 2019.

[56] W. Zhang and D. J. Linden, “The other side of the engram: Experience-driven changes in neuronal intrinsic excitability,” *Nature Reviews Neuroscience*, vol. 4, pp. 885–900, 2003.

[57] L. Liu and M. T. (Özsu, “Curse of dimensionality,” *Encyclopedia of database systems*, 2009.

[58] D. Querlioz, O. Bichler, P. Dollfus, and C. Gamrat, “Immunity to device variations in a spiking neural network with memristive nanodevices,” *IEEE Transactions on Nanotechnology*, vol. 12, pp. 288–295, May 2013. DOI: 10.1109/TNANO.2013.2250995.

[59] A. Afifi, A. Ayatollahi, and F. Raissi, “Implementation of biologically plausible spiking neural network models on the memristor crossbar-based cmos/nano circuits,” in *2009 European Conference on Circuit Theory and Design*, 2009, pp. 563–566.

[60] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, “Long-term recurrent convolutional networks for visual recognition and description,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015.

[61] P. Pinheiro and R. Collobert, “Recurrent convolutional neural networks for scene labeling,” in *Proceedings of the 31st International Conference on Machine Learning*, E. P. Xing and T. Jebara, Eds., ser. Proceedings of Machine Learning Research, vol. 32, Bejing, China: PMLR, Jun. 2014, pp. 82–90. [Online]. Available: http://proceedings.mlr.press/v32/pinheiro14.html.

[62] M. Liang and X. Hu, “Recurrent convolutional neural network for object recognition,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015.

[63] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://doi.org/10.1162/neco.1997.9.8.1735. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735.

[64] J. Schmidhuber, “Learning complex, extended sequences using the principle of history compression,” *Neural Computation*, vol. 4, no. 2, pp. 234–242, 1992. DOI: 10.1162/neco.1992.4.2.234. eprint: https://doi.org/10.1162/neco.1992.4.2.234. [Online]. Available: https://doi.org/10.1162/neco.1992.4.2.234.

[65] V. Mnih, N. Heess, A. Graves, and k. kavukcuoglu koray, “Recurrent models of visual attention,” in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2014, pp. 2204–2212. [Online]. Available: http://papers.nips.cc/paper/5542-recurrent-models-of-visual-attention.pdf.

[66] E. O. Neftci, H. Mostafa, and F. Zenke, “Surrogate Gradient Learning in Spiking Neural Networks,” *arXiv e-prints*, arXiv:1901.09948, arXiv:1901.09948, Jan. 2019. arXiv: 1901.09948 [cs.NE].

[67] C. Lee, S. Shakib Sarwar, and K. Roy, “Enabling Spike-based Backpropagation in State-of-the-art Deep Neural Network Architectures,” *arXiv e-prints*, arXiv:1903.06379, arXiv:1903.06379, Mar. 2019. arXiv: 1903.06379 [cs.NE].

[68] L. Deng, Y. Wu, Y. Hu, L. Liang, G. Li, X. Hu, Y. Ding, P. Li, and Y. Xie, *Comprehensive snn compression using admm optimization and activity regularization*, 2020. arXiv: 1911.00822 [cs.NE].

[69] T. Poggio and Q. Liao, “Theory ii: Deep learning and optimization,” *Bulletin of the Polish Academy of Sciences: Technical Sciences*, vol. 66, no. No 6, pp. 775–787, 2018. DOI: 10.24425/bpas.2018.125925.

VITA

jasonallred.com

Jason Allred and his wife, Celisse, are the parents of four young children–three of whom were born during Jason's doctoral studies at Purdue University. Jason will be joining the faculty at Brigham Young University–Idaho in the Computer Science and Electrical Engineering Department.

EDUCATION:

- **Ph.D. Electrical and Computer Engineering**, Purdue University, School of Electrical and Computer Engineering, August 2021.
  - Thesis: *Adapting Neural Network Machine Learning Algorithms for Neuromorphic Implementations.*
- **Teaching Certificate: Teaching and Learning in Engineering**, Purdue University, School of Engineering Education, May 2021.
- **M.S. Computer Engineering**, Utah State University, August 2013.
  - Thesis: *Designing, Optimizing, and Sustaining Heterogeneous Chip Multiprocessors to Systematically Exploit Dark Silicon.*
- **B.S. Computer Engineering**, Brigham Young University–Idaho, July 2011. *Summa Cum Laude.* Presidential Scholar.

PROFESSIONAL EXPERIENCE:

- **Electronics Engineer**. Department of Defense. Hill Air Force Base. Clearfield, UT. 2013-2015.
- **Software Intern**. LabVIEW FPGA Hardware Team, National Instruments. Austin, TX. Certified LabVIEW Associate Developer. 2011.

ACADEMIC RESEARCH EXPERIENCE:

- **Graduate Research Assistant**. Purdue University, Electrical and Computer Engineering Department. West Lafayette, IN. Research performed with Professor Kaushik Roy. [Nanoelectronics Research Lab, CBRIC: Center for Brain-Inspired Computing]. 2015-2021.
- **Graduate Research Assistant**. Utah State University, Electrical and Computer Engineering Department. Logan, UT. 2011-2013. Research performed with Professor Sanghamitra Roy and Professor Kaushik Chakraborty. [BRIDGE LAB].

ACADEMIC TEACHING EXPERIENCE:

- **Graduate Lecturer / Instructor of Record**. Purdue University, Electrical and Computer Engineering Department. West Lafayette, IN. 2020.
  - Course: ECE 368 Data Structures [and Algorithms]
- **Graduate Teaching Assistant**. Utah State University, Electrical and Computer Engineering Department. Logan, UT. 2012-2013.
  - Courses: ECE 5/6460 VLSI Design Automation and ECE 5/6470 VLSI Design.
  - Outstanding Graduate T.A. of the Year Award (2012)
- **Undergraduate Teaching Assistant**. Brigham Young University–Idaho, Computer Science and Electrical Engineering Department. Rexburg, ID.
  - Courses: ECE 224 Intro to Digital Design, ECE 324 Computer Architecture, CS 345 Operating Systems, and ECE 440 Data Communications and Networking. 2010-2011.
- **Undergraduate Lead Lab Assistant**. Brigham Young University–Idaho, Computer Science and Electrical Engineering Department. Rexburg, ID. 2010-2011.

PUBLICATIONS:

- Z. Zhang, S. Mondal, S. Mandal, **Jason M. Allred**, N. A. Aghamiri, A. Fali, Z. Zhang, H. Zhou, H. Cao, F. Rodolakis, J. L. McChesney, Q. Wang, Y. Sun, Y. Abate, K. Roy, K. M. Rabe, and S. Ramanathan, "Neuromorphic learning with Mott insulator NiO," *Proceedings of the National Academy of Sciences*, to appear.
  - We highlight a functional similarity between habituation and sensitization behaviors observed in NiO and dynamic, localized plasticity rules useful for unsupervised, lifelong learning in neural networks.
- **Jason M. Allred**, Steven J. Spencer, Gopalakrishnan Srinivasan, and Kaushik Roy. "Explicitly Trained Spiking Sparsity in Spiking Neural Networks with Backpropagation," *arXiv preprint*, Mar 2020, update to appear.
  - We take advantage of recently developed approximate spiking backpropagation machine learning rules to add spiking activity to the loss function, explicitly training for reduced spiking computations.
- **Jason M. Allred** and Kaushik Roy. "L4-Norm Weight Adjustments for Converted Spiking Neural Networks." *manuscript in progress.*
  - We consider the statistical variation of spiking neurons' membrane potentials to adjust for a functional mismatch when converting pre-trained non-spiking neural networks to spiking neural networks.
- **Jason M. Allred** and Kaushik Roy. "Controlled Forgetting: Targeted Stimulation and Dopaminergic Plasticity Modulation for Unsupervised Lifelong Learning in Spiking Neural Networks." *Frontiers in Neuroscience*, Jan 2020.
  - We design a spiking neural network that achieves lifelong learning on a disjoint dataset by leveraging dopaminergic neuromodulation in targeted subsets of the network to avoid catastrophic forgetting.
  - (Selected as a *2021 Neuromorphic Engineering Editors' Pick.*)

- Priyadarshini Panda, **Jason M. Allred**, Shriram Ramanathan, and Kaushik Roy. "ASP: Learning to Forget with Adaptive Synaptic Plasticity in Spiking Neural Networks." *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2017.
  - We incorporate adaptive weight decay into traditional STDP for controlled replacement learning that avoids catastrophic forgetting in spiking networks.
- **Jason M. Allred** and Kaushik Roy. "Convolving over time via recurrent connections for sequential weight sharing in neural networks." *Proceedings of the 2017 International Joint Conference on Neural Networks* (IJCNN), 2017.
  - Using recurrent synaptic connections, we design a network topology that temporally shares weights of a convolutional kernel to enable localized learning without weight synchronization.
- **Jason M. Allred** and Kaushik Roy. "Unsupervised Incremental STDP Learning Using Forced Firing of Dormant or Idle Neurons." *Proceedings of the 2016 International Joint Conference on Neural Networks* (IJCNN), 2016.
  - We stimulate the firing of spiking neurons when input patterns are unrecognized to quickly learn novel information while avoiding catastrophic forgetting without data reinforcement.
- Dean Anjacas, Koushik Chakraborty, Sanghamitra Roy, and **Jason M. Allred**. "Tackling QoS-induced Aging in Exascale Systems through Agile Path Selection." *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis* (CODES+ISSS), 2014.
  - A network-on-chip routing policy that mitigates aging while preserving quality of service.
- **Jason M. Allred**. Master's Thesis: "Designing, Optimizing, and Sustaining Heterogeneous Chip Multiprocessors to Systematically Exploit Dark Silicon." *Utah State University* (USU), 2013.

- **Jason M. Allred**, Sanghamitra Roy, and Koushik Chakraborty. "Long Term Sustainability of Differentially Reliable Systems in the Dark Silicon Era." *Proceedings, IEEE International Conference on Computer Design* (ICCD), 2013.
  - We construct a feedback control thread-to-core mapping framework to extend the energy efficiency of a heterogeneous multicore system with varying levels of reliability.
- **Jason M. Allred**, Sanghamitra Roy, and Koushik Chakraborty. "Dark Silicon Aware Multicore Systems: Employing Design Automation with Architectural Insight." *IEEE Transactions on Very Large Scale Integration Systems* (TVLSI), 2013.
  - We optimize heterogeneous microarchitectures for energy efficiency in the presence of dark silicon.
- **Jason M. Allred**, Sanghamitra Roy, and Koushik Chakraborty. "Designing for Dark Silicon: A Methodological Perspective on Energy Efficient Systems." Proceedings of the 2012 ACM International Symposium on Low Power Electronic Devices (ISLPED), 2012.
  - We propose a metric and optimization algorithm for choosing design-time per-core maximum voltage-frequency domains in a DVFS heterogeneous multicore system with dark silicon.

VOLUNTEER WORK:

- *External Reviewer.* Wiley/WIRES 2020, ICCAD 2019, DAC 2015, TCAS 2013, ASQED 2013, ISQED 2012, ASQED 2012, and ICCD 2012.
- *IEEE Student Chapter Chairman.* Brigham Young University–Idaho. 2010.
- *Religious Missionary.* The Church of Jesus Christ of Latter-day Saints. Brazil São Paulo South Mission. 2007-2009. Fluent in Brazilian Portuguese.
- *Scoutmaster.* Troop 365. Trapper Trails Council. 2014-2015.

AWARDS:

Academic Awards

- *Ross Fellowship*, School of Electrical and Computer Engineering, Purdue Univ. (2015).
- *Outstanding Graduate Teaching Assistant of the Year Award*, Electrical and Computer Engineering Department, Utah State University (2012-2013).
- *Summa Cum Laude*, Brigham Young University-Idaho (2011).

Declined PhD Fellowships

- *ECE Wisconsin Distinguished Graduate Fellowship*, University of Wisconsin-Madison (2015).
- *Department of Electrical and Computer Engineering Fellowship*, University of Texas at Austin (2015).
- *Royal E. Cabell Fellowship*, Northwestern University (2015).
- *Institute of Technology Dean's Fellowship*, Carnegie Mellon University (2013).
- *College of Technology and Innovation Dean's Fellowship*, Arizona State University (2013).
- *Wayne Brown Graduate Fellowship*, University of Utah (2013).
- *Royal E. Cabell Fellowship*, Northwestern University (2013).

Patents

- **US 9213577 B2** - Sustainable differentially reliable architecture for dark silicon.

CERTIFICATIONS:

- *Teaching and Learning in Engineering.* Purdue University. (2021).
- *Certified LabVIEW Associate Developer.* National Instruments. (2011-2013).
- *Franklin Covey Leadership Foundations.* Franklin Covey. (2011).