Spring 5-23-2021

Inference and Learning in Spiking Neural Networks for Neuromorphic Systems

Amar Shrestha
Syracuse University

Abstract

Neuromorphic computing is a computing field that takes inspiration from the biological and physical characteristics of the neocortex system to motivate a new paradigm of highly parallel and distributed computing to take on the demands of the ever-increasing scale and computational complexity of machine intelligence esp. in energy-limited systems such as Edge devices, Internet-of-Things (IOT), and cyber physical systems (CPS). Spiking neural network (SNN) is often studied together with neuromorphic computing as the underlying computational model . Similar to the biological neural system, SNN is an inherently dynamic and stateful network. The state and output of SNN do not only dependent on the current input, but also dependent on the history information. Another distinct property of SNN is that the information is represented, transmitted, and processed as discrete spike events, also referred to as action potentials. All the processing happens in the neurons such that the computation itself is massively distributed and parallel. This enables low power information transmission and processing.

However, it is inefficient to implement SNNs on traditional Von Neumann architecture due to the performance gap between memory and processor. This has led to the advent of energy-efficient large-scale neuromorphic hardware such as IBM's TrueNorth and Intel's Loihi that enables low power implementation of large-scale neural networks for real-time applications. And although spiking networks have theoretically been shown to have Turing-equivalent computing power, it remains a challenge to train deep SNNs; the threshold functions that generate spikes are discontinuous, so they do not have derivatives and cannot directly utilize gradient-based

optimization algorithms for training. Biologically plausible learning mechanism spike-timing-dependent plasticity (STDP) and its variants are local in synapses and time but are unstable during training and difficult to train multi-layer SNNs.

To better exploit the energy-saving features such as spike domain representation and stochastic computing provided by SNNs in neuromorphic hardware, and to address the hardware limitations such as limited data precision and neuron fan-in/fan-out constraints, it is necessary to re-design a neural network including its structure and computing. Our work focuses on low-level (activations, weights) and high-level (alternative learning algorithms) redesign techniques to enable inference and learning with SNNs in neuromorphic hardware.

First, we focused on transforming a trained artificial neural network (ANN) to a form that is suitable for neuromorphic hardware implementation. Here, we tackle transforming Long Short-Term Memory (LSTM), a version of recurrent neural network (RNN) which includes recurrent connectivity to enable learning long temporal patterns. This is specifically a difficult challenge due to the inherent nature of RNNs and SNNs; the recurrent connectivity in RNNs induces temporal dynamics which require synchronicity, especially with the added complexity of LSTMs; and SNNs are asynchronous in nature. In addition, the constraints of the neuromorphic hardware provided a massive challenge for this realization. Thus, in this work, we invented a store-and-release circuit using integrate-and-fire neurons which allows the synchronization and then developed modules using that circuit to replicate various parts of the LSTM. These modules enabled implementation of LSTMs with spiking neurons on IBM's TrueNorth Neurosynaptic processor. This is the first work to realize such LSTM networks utilizing spiking neurons and implement on a neuromorphic hardware. This opens avenues for the use of neuromorphic hardware in applications involving temporal patterns.

Moving from mapping a pretrained ANN, we work on training networks on the

neuromorphic hardware. Here, we first looked at the biologically plausible learning algorithm called STDP which is a Hebbian learning rule for learning without supervision. Simplified computational interpretations of STDP is either unstable and/or complex such that it is costly to implement on hardware. Thus, in this work, we proposed a stable version of STDP and applied intentional approximations for low-cost hardware implementation called Quantized 2-Power Shift (Q2PS) rule. With this version, we performed both unsupervised learning for feature extraction and supervised learning for classification in a multilayer SNN to achieve comparable to better accuracy on MNIST dataset compared to manually labelled two-layered networks.

Next, we approached training multilayer SNNs on a neuromorphic hardware with backpropagation, a gradient-based optimization algorithm that forms the backbone of deep neural networks (DNN). Although STDP is biologically plausible, its not as robust for learning deep networks as backpropagation is for DNNs. However, backpropagation is not biologically plausible and not suitable to be directly applied to SNNs, neither can it be implemented on a neuromorphic hardware. Thus, in the first part of this work, we devise a set of approximations to transform backprogation to the spike domain such that it is suitable for SNNs. After the set of approximations, we adapted the connectivity and weight update rule in backpropagation to enable learning solely based on the locally available information such that it resembled a rate-based STDP algorithm. We called this Error-Modulated STDP (EMSTDP). In the next part of this work, we implemented EMSTDP on Intel's Loihi neuromorphic chip to realize online in-hardware supervised learning of deep SNNs. This is the first realization of a fully spike-based approximation of backpropagation algorithm implemented on a neuromorphic processor. This is the first step towards building an autonomous machine that learns continuously from its environment and experiences.

INFERENCE AND LEARNING IN SPIKING NEURAL NETWORKS FOR NEUROMORPHIC SYSTEMS

By

Amar Shrestha

B.S., Kathmandu University, Dhulikhel, Nepal, 2013

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

Syracuse University

May 2021

Acknowledgements

Firstly, I would like to express my sincere gratitude to my doctoral advisor, Dr. Qinru Qiu, for her guidance, support and endless encouragement that helped me through my study, research, and life during this doctoral endeavour.

Secondly, I would like to thank the rest of my doctoral committee members, Dr. Lixin Shen, Dr. Garrett Ethan Katz, Dr. Senem Velipasalar, Dr. Chilukuri K. Mohan and Dr. Bryan S. Kim for their encouragement and insightful comments.

Thirdly, I would like thank my fellow lab mates at Syracuse University, Dr. Khadeer Ahmed, Dr. Qiuwen Chen, Dr. Zhe Li, Haowen Fang, Krittaphat Pugdeethosapol, Yilan Li, Ziyi Zhao, Chen Luo, Zhao Jin, Mingyang Li, Zaidao Mei and Daniel Patrick Rider. I am very grateful for their help and teamwork.

And finally, I would like to thank my family, Krishna Kumar Shrestha, Sunita Shrestha, and Eliza Shrestha for their full support and encouragement during this endeavour.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Outline and Main contributions

The ever-increasing scale and computational complexity of machine intelligence have been posing challenges on the traditional Von Neumann architecture and motivates a new paradigm of highly parallel and distributed computing inspired by biological neural systems, namely neuromorphic computing which aims to learn from the biological and physical characteristics of the neocortex system to provide energy efficient solutions to state-of-the-art machine intelligence problems.

Spiking neural network (SNN) is often studied together with neuromorphic computing as the underlying computational model. SNN is an inherently dynamic and stateful network and has more biologically plausible features than conventional artificial neural networks (ANNs) [1]. One of the most distinct differences of SNN from ANN is the ability to process temporal information. The state and output of SNN do not only dependent on the current input, but also dependent on the history information. Another distinct property of SNN is that the information is represented, transmitted, and processed as discrete spike events, also referred to as action potentials [2]. Spikes are electrical pulses in biological neural systems. In SNN mathematical

models, spike is usually represented by Dirac Delta functions. All the processing happens in the neurons such the computation itself is massively distributed and parallel. Communication through spikes and massively distributed computation in neurons enables low power information transmission and processing.

However, due to the performance gap between memory and processor, it is inefficient to implement SNNs on traditional Von Neumann architecture. This has led to the advent of energy-efficient large-scale neuromorphic hardware that enables low power implementation of large-scale neural networks for real-time applications. One of the examples is IBM's Neurosynaptic Processor, "TrueNorth". Operating in the spiking domain, TrueNorth has achieved close to state-of-the-art inference results in various pattern recognition tasks [3] with very high energy efficiency. Converting pretrained networks to an SNN has also produced good results in pattern recognition [4] on platforms other than TrueNorth as well. TrueNorth has no facility for training SNNs in-hardware. Loihi [5] is a digital neuromorphic chip recently developed by Intel which provides a programmable microcode learning engine for on-chip SNN training.

Although spiking networks have theoretically been shown to have Turing-equivalent computing power [6], it remains a challenge to train deep SNNs; the threshold functions that generate spikes are discontinuous, so they do not have derivatives and cannot directly utilize gradient-based optimization algorithms for training. Biologically plausible learning mechanism spike-timing-dependent plasticity (STDP) [7] and its variants are local in synapse and time, thus the plasticity of the synaptic weights depend on the relative timings of the pre-and post-synaptic spikes. STDP is also unstable during training. Thus, it is difficult to train multilayer SNNs by utilizing only the local information.

To better exploit the energy-saving features such as spike domain representation and stochastic computing provided by SNNs in neuromorphic hardware, and to ad-

dress the hardware limitations such as limited data precision and neuron fan-in/fan-out constraints, it is necessary to re-design a neural network including its structure and computing. Our work focuses on low-level (activations, weights) and high-level (alternative learning algorithms) redesign techniques to enable inference and learning with SNNs in neuromorphic hardware. The former approach transforms a trained artificial neural network (ANN) to a form that suitable for neuromorphic hardware implementation, and the later approach adopts neural networks with innovative structures and/or learning algorithms that are more biologically plausible.

In this thesis, we first review the literature surrounding SNNs including neuron and synapse models along with spike encoding and learning with spikes. Then, we will introduce neuromorphic hardware along with briefly reviewing two popular neuromorphic systems; TrueNorth and Loihi. The rest of the chapters are ordered from inference to learning with SNNs and in the neuromorphic hardware.

As mentioned previously, we start with a chapter dealing with inference with SNNs in neuromorphic hardware, mainly TrueNorth.

In Chapter 2 we present a design flow that maps a special case of recurrent networks called Long Short-Term Memory (LSTM) onto a spike-based platform. The framework utilizes various approximation techniques; activation discretization, weight quantization, scaling, and rounding; spiking neural circuits that implement complex gating mechanisms, and a store-and-release technique to enable neuron synchronization and faithful storage. While the presented techniques can be applied to map LSTM to any Spiking Neural Network (SNN) simulator/emulator, here we choose the TrueNorth chip as the target platform by adhering to its hardware constraints. Three LSTM applications, parity check, Extended Reber Grammar and Question classification, are evaluated. The tradeoffs among accuracy, performance, and energy tradeoffs achieved on TrueNorth are demonstrated. This is compared to the performance on an SNN platform without hardware constraints, which represents the upper

bound of the achievable accuracy.

In the next chapters we discuss learning with SNNs. In Chapter 3 we present a low-cost, simplified, yet stable STDP rule for layer-wise unsupervised and supervised training of a multilayer feed-forward SNN. We propose to approximate Bayesian neurons using Stochastic Integrate and Fire (SIF) neuron model and introduce a supervised learning approach using teacher neurons to train the classification layer with one neuron per class. A SNN is trained for classification of handwritten digits with multiple layers of spiking neurons, including both the feature extraction and classification layer, using the proposed STDP rule. Our method achieves comparable to better accuracy on MNIST dataset than manually labelled two-layer networks for the same sized hidden layer. We also analyze the parameter space to provide rationale for parameter fine-tuning and provide additional methods to improve noise resilience and input intensity variations. We further propose a Quantized 2-Power Shift (Q2PS) STDP rule, which reduces the implementation cost of digital hardware while achieves comparable performance.

In Chapter 4, we present an approximation of the backpropagation algorithm completely with spiking neurons and extend it to a local weight update rule which resembles a biologically plausible learning rule spike-timing-dependent plasticity (STDP). This will enable error propagation through spiking neurons for a more biologically plausible and neuromorphic implementation friendly backpropagation algorithm for SNNs. We test the proposed algorithm on various traditional and nontraditional benchmarks with competitive results.

In Chapter 5, we present a spike-based backpropagation algorithm with biological plausible local update rules and adapt it to fit the constraint in a neuromorphic hardware. The algorithm is implemented on Intel's Loihi chip enabling low power in-hardware supervised online learning of multilayered SNNs for mobile applications. We test this implementation on MNIST, Fashion-MNIST, CIFAR-10, and MSTAR

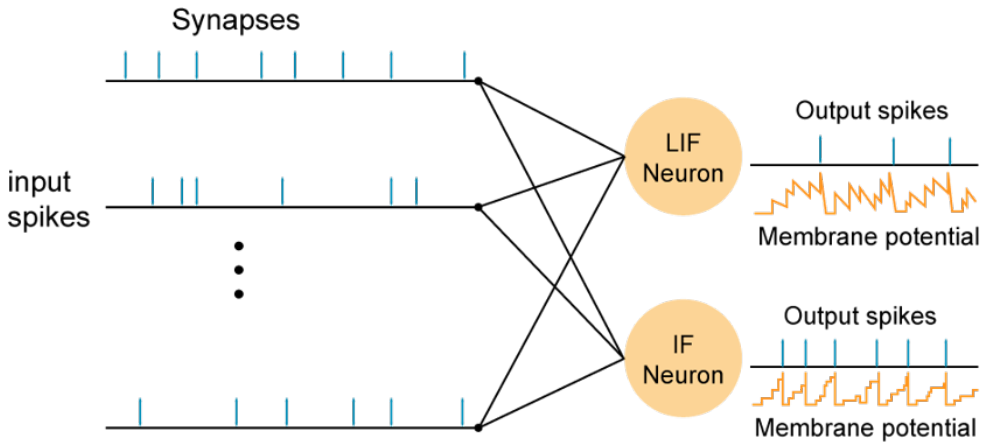


Figure 1.1: Responses of LIF and IF spiking neurons

datasets with promising performance and energy efficiency, and demonstrate the possibility of incremental online learning with the implementation.

Finally, in Chapter 6 of this thesis, we summarize all the above-mentioned works and discuss future research directions.

1.2 Spiking Neural Networks

Biological neurons communicate with each other by generating and propagating electrical pulses called spikes [8, 9]. At the high abstraction level, all spiking models share the following common properties inspired by a biological neuron: (1) they process information coming from many inputs and produce single or multiple spikes; (2) the probability of spike generation is increased by excitatory inputs and decreased by inhibitory inputs; (3) at least one state variable is used to characterize their dynamics and the model is supposed to generate one or more spikes when the internal variables of the model reach a certain state.

A sequence of the spike events, i.e., a spike train, can be described as the following

$$S(t) = \sum_{f} \delta\left(t - t^{f}\right) \tag{1.1}$$

where f = 1, 2, ... is the label of the spike and $\delta(.)$is a Dirac function with $\delta(t) \neq 0$for t=0 and $\int_{-\infty}^{\infty} \delta(t)\, dt = 1$. The basic assumption underlying most spiking neuron models is that it is the timing of spikes rather than the specific shape of the spikes that carries neural information [10].

1.3 Neuron Models

The existing neuron models can mainly be categorized into two groups, conductance-based models and spike-based models. Conductance-based models are based on an equivalent circuit representation of a cell membrane as first put forth by Hodgkin and Huxley [11]. These models are more biologically plausible and are used for computational neuroscience purposes. Spike-based models are more efficient in terms of computational complexity and thus are used in spike-based computational systems. Among the spike-based models, the IF and LIF models [10] are the most widely used. Both models abstract biological neurons as point dynamical systems. The dynamics of the LIF unit is described by the following formula:

$$C\frac{\mathrm{d}u}{\mathrm{d}t}(t) = -\frac{1}{R}u(t) + \left(i_o(t) + \sum w_j i_j(t)\right) \tag{1.2}$$

where $u(t)$ is the membrane potential, C is the membrane capacitance, R is the input resistance, $i_o(t)$ is the external current driving the neural state, $i_j(t)$ is the input current from the j-th synaptic input, and w_j represents the strength of the j-th synapse. When $R \to \infty$, Eq. 1.2 gives the IF model. In both IF and LIF models, a neuron is supposed to fire a spike, whenever the membrane potential $u$reaches a certain value υcalled a firing threshold. Immediately after a spike the neuron state is reset to a new value $u_{res}<\upsilon$ and hold at that level for the time interval representing the refractory period. The responses of IF and LIF neurons for incoming spike trains are shown in Figure 1.1.

The Spike Response Model (SRM) model interprets the neuron dynamics in Eq. 1.2 as the convolution of an impulse response of a filter with the input spike train. It expresses the membrane potential at time t as an integral over the past input and kernel responses. A typical SRM model is defined as the following:

$$V(t) = \sum_{i=1}^{N} w_i \sum_{t_i^j < t} K\left(t - t_i^j\right) - V_{th} \sum_{t_{spike}^j} h\left(t - t_{spike}^j\right), \tag{1.3}$$

where $V(t)$ is the membrane potential, $K(t)$ and $h(t)$ are two convolution kernels associated to synaptic dynamics and membrane potential reset events. Each input spike at time t_i^j convolves with the kernel $K(t)$, which incurs a voltage called postsynaptic potential (PSP). When $V(t)$ exceeds the threshold V_{th}, the neuron generates a spike output indicated by the spike time t_{spike}^j.

1.4 Synapse Models

Neurons connect and communicate with one another through specialized junctions called synapses [12, 13]. The arrival of a presynaptic spike at a synapse triggers an input signal into the postsynaptic neuron. This signal corresponds to the synaptic electric current flowing into the biological neuron [9]. It causes a change in the neuron membrane potential, which is referred to as postsynaptic potential (PSP). In a general form, the time course of PSP can be described by the convolution of the presynaptic spike train $S(t)$ and a kernel function $K(t)$ scaled by a weight coefficient w as the following:

$$PSP(t) = w \int_0^\infty S(s-t) K(s)\, ds = w \sum_{t^j < t} K\left(t - t^j\right) \tag{1.4}$$

$K(t)$ can be an exponential, dual exponential, or an alpha kernel. The simplest kernel is an ideal all-pass lossless filter.

1.5 Neural Coding and spike timing

Neural coding refers to the way in which information is represented by discrete spikes. Neural coding is tightly coupled with the neuron model and determines the performance of SNN and hardware implementation. Hence, it is an essential part of the SNN. However, exactly how brain and sensory systems encode information has not been fully discovered.

The most straightforward way is to represent a value by the number of spikes in a unit time. This type of encoding is referred to as rate coding. It agrees with the observation that the sensory nerves' spike frequency increases as the input stimulus intensity increases. Rate coding is widely adopted. To represent a large value requires high spike frequency, which impairs energy efficiency, and may even pose a challenge to neuromorphic chip design [14, 15, 16]. 3) Rate coding has quantization error. When spikes are generated as stochastic events, there is also sampling error. These errors make SNN less accurate.

There are various coding schemes that take spike timing into account. They are referred to as temporal coding [17, 18, 19]. In such a coding scheme, the temporal structure of the spike train can convey information.

1.6 Learning

The ability of synaptic connections to change their strength is referred to as synaptic plasticity. This is considered the basic mechanism underlying learning and memory in biological neural networks [20]. Various forms of synaptic plasticity co-exist. Some are determined only by the history of presynaptic stimulation, independently of the postsynaptic responses [21, 22, 23]. Others depend on the temporal order of pre- and postsynaptic activity [7, 21, 24].

In general, synaptic potential is observed when presynaptic spikes precede post-

synaptic spikes, as it indicates a causal relationship. The reversed order of spikes induces synaptic depression. This phenomenon is called Spike-Timing-Dependent-Plasticity (STDP). It can be used for unsupervised learning. A popular choice for the STDP rule [25] for potentiation Δw_+ and depression Δw_- is given as:

$$\begin{aligned} \Delta w_+ &= +A_+ \exp\left(-\Delta t/\tau_+\right) \quad \text{for } \Delta t > 0 \\ \Delta w_- &= -A_- \exp\left(+\Delta t/\tau_-\right) \quad \text{for } \Delta t < 0 \end{aligned} \tag{1.5}$$

where τ_+ and τ_- determine the ranges of pre- to postsynaptic interspike intervals over which synaptic strengthening and weakening occur. A_+ and A_- determine the maximum amount of synaptic modification. And Δt is the time of postsynaptic spike minus the time of the presynaptic spike.

A wide range of SNN algorithms that can learn temporal spike patterns employ more biologically realistic LIF neuron models with alpha or dual-exponential synapses [26, 27, 28, 29, 30]. In these models, the PSP decays exponentially over time, hence, can be utilized as a metric to reflect temporal dependency. These type of neuron does not simply accumulate weighted spikes as a membrane potential, instead, it integrates weighted time-varying PSP, hence exhibiting complex temporal dynamic behavior. Various STDP learning rules also rely on spike timing by updating the weight based on traces of different forms. Those traces are similar as PSPs. They are decaying state variables reflecting the temporal history of input and output spikes. A pairwise STDP rule using traces [31] is given as:

$$\Delta w = A_+ x\left(t\right) S_x\left(t\right) - A_- y\left(t\right) S_y\left(t\right) \tag{1.6}$$

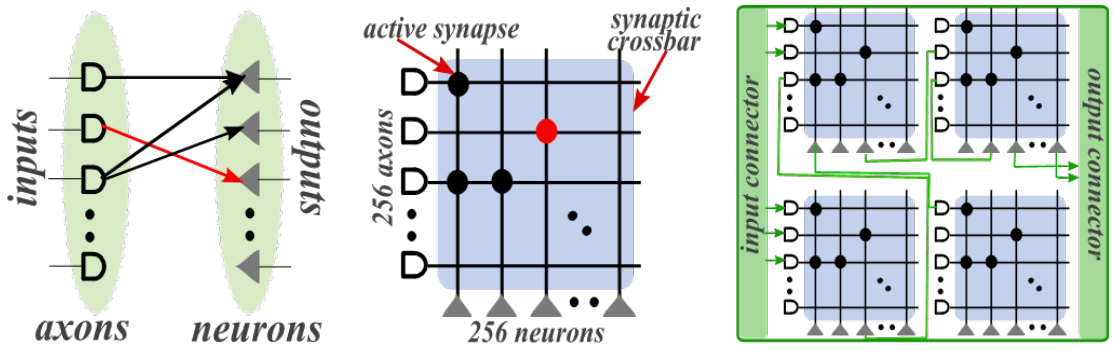
where $x(t)$and $y(t)$are the pre- and postsynaptic traces. $S_x\left(t\right) = \sum_x \delta\left(t - t^x\right)$and $S_y\left(t\right) = \sum_y \delta\left(t - t^y\right)$ are the pre- and postsynaptic spike trains.

The most straightforward way to implement supervised learning is to use Hebbian Learning [32, 30]. Supervision is introduced in Hebbian learning by an additional

'teaching' signal that reinforces the postsynaptic neuron to fire at target times and to remain silent at other times. The 'teaching' signal is usually transmitted to the neuron in the form of synaptic currents or as intracellularly injected currents. Recently, the backpropagation algorithm has been approximated in various forms for SNNs for supervised learning [3, 33, 34, 35, 36, 37].

1.7 Neuromorphic Hardwares

The concept of neuromorphic computing was first proposed by Carver Mead in the 1980s [38, 39, 40, 41]. The early works in this area focused at emulating the analog behavior of neural systems. It is observed that biological systems can achieve many orders of magnitude higher efficiency than digital systems when performing certain cognitive tasks. [38] and [41] credit such advantage to the fundamental differences between digital circuits and biological systems. Digital systems eliminate noise as much as possible to guarantee bit-level accuracy, and the mathematical operations are explicitly defined and implemented as Boolean functions. Rather than encoding information as binary forms, the neural system utilizes analog quantities, such as membrane potential, synapse current, etc., as computing primitives. Although inherently noisy, neural systems are naturally adaptive, noise can be eliminated by feedback or jointly processing multiple signals. The early works in neuromorphic computing tried to bridge the gap between the lower level physical details of biological systems and the higher-level computational functionality. [38, 42] claim that, due to its adaptability, neuromorphic systems are more resistant to noise and component failure and have the potential to be more energy efficient. [41] suggest that the basic operations of neural computational primitives such as conservation of charge, amplification, exponentiation, integration, thresholding, etc., can be emulated by physical processes in analog devices implemented using complementary metal-oxide-



(a) (b) (c)

Figure 1.2: (a)Structural view (b)Functional view of a core (c) corelet [86]

semiconductor (CMOS) Very Large-Scale Integration (VLSI) technology.

The scope of neuromorphic computing has been expanding in the recent decade. Now it includes but not limited to the following areas 1) applying digital, analog or mixed-signal Application Specific Integrated Circuits (ASICs) to implement biological neuron and synapse networks at different plausibility levels for machine intelligence applications, 2) leveraging large-scale computing systems to emulate biological neural networks to facilitate the understanding of brain and cognition; and 3) utilizing emerging materials and devices such as memristors, phase-changing materials, photonic circuits to model the analog behavior of neurons and synapse plasticity. In our work, we mainly utilize digital ASICS thus, we will review two popular digital neuromorphic systems; TrueNorth and Loihi.

1.7.1 TrueNorth

This work uses TrueNorth neurosynaptic processor as an example hardware platform to discuss the mapping of SNN based LSTM. The TrueNorth processor is a highly efficient, scalable, and flexible neuromorphic hardware. It consists of 4096 cores [43] each with 256 neurons and 256 axons connected via 256x256 directed synaptic connections, thus providing 1 million programmable neurons and 268 million configurable synapses. The communication between those neurons is in the form of spike events

represented as address event representation (AER). These spike events are sparse in time. Since the active power is proportional to the firing activity, the event-driven nature of the architecture gives very high energy efficiency. In the normal mode, the membrane potential is processed, and spike events routed asynchronously inside the chip within 1 ms timesteps called a tick. A spike generated by a neuron can target any single axon on the chip. Figure 1.2a shows a structural view of a TrueNorth core with axons as input and neurons as outputs and synapses linking them. This representation is similar to a traditional neural network. Figure 1.2b shows a functional view of a core as a crossbar, where horizontal lines are axons, cross points are individually programmable synapses, vertical lines are neuron inputs, and triangles are neurons. Spikes flow from axons via active synapses to neurons.

Truenorth's neuron model is based on the classic leaky integrate-and-fire neuron (LIF) with five basic operations: synaptic integration, leak integration, threshold comparison, spike generation, and membrane potential reset. The membrane potential $V_j(t)$ of the neuron j is updated according to these five operations as summarized in Equations 1.7, 1.8 and 1.9.

Synaptic integration:

$$V_j\left(t\right) = V_j\left(t-1\right) + \sum_{i=0}^{N-1} x_i\left(t\right) s_i \tag{1.7}$$

Leak integration:

$$V_j\left(t\right) = V_j\left(t\right) - \lambda_j \tag{1.8}$$

Membrane potential reset:

$$V_j\left(t\right) \geq \alpha_j,\ \textit{spike and}\ V_j\left(t\right) = R_j \tag{1.9}$$

This LIF neuron model is augmented by configurable and reproducible stochas-

ticity [44]. Each individual neuron can be configured to have stochastic synaptic inputs, leaks, and threshold to enable rich dynamics across population and time. The neuron model allows for four leak modes that bias the internal state dynamics in four different ways so that neurons can have radically different responses to identical inputs. The leaks can be either positive or negative to let the membrane potential to diverge away from or converge towards a resting potential. The neuron model also provides two types of thresholds; deterministic and stochastic, so that neurons can fire at different patterns even with the same accumulated membrane potential. It has six reset modes to determine the membrane potential after firing, enabling a rich finite-state transition behavior.

Exploiting the provided configurability, users can use TrueNorth neurons to implement a wide variety of computational functions, including arithmetic, control, data generation, logic, memory, classical neuron behavior, signal processing, and probabilistic computation. The programmable leakage and threshold give neurons the capacity to support a variety of neural codes including rate, population, binary, and time-to-spike coding. By composing multiple neurons together, an extremely rich and diverse array of complex computations and behaviors from simpler library elements can be synthesized. They were able to qualitatively replicate the 20 behaviors of the Izhikevich dynamical neuron model using a small number of elementary neurons.

The synapses themselves are binary (1: connected, 0: disconnected). Each synapse connected to a neuron in the crossbar is allocated a choice of four 8-bit signed weights.

The synaptic connections and their weights between axons and neurons are captured by a crossbar matrix at an abstract level where the weight of the corresponding synapse in the crossbar is selected from 4 possible integers determined by the axon type at each neuron. A Corelet [45] describes a network on the TrueNorth cores by encapsulating all details except external inputs and outputs as shown in Figure 1.2c. The creating, composing, and decomposing of corelets is done in an object-oriented

Corelet Language in the programming paradigm for TrueNorth called Corelet Programming Environment (CPE). Programming TrueNorth includes creating corelets with specific neuron behaviors, synaptic connections, weights, and delays to achieve the desired functionalities. Multiple corelets can be combined through their input and output connectors.

1.7.2 Loihi

Loihi [5] is a digital neuromorphic chip recently developed by Intel. Loihi is fabricated in Intel's 14-nm process. The 128-neuromorphic cores implement 130,000 artificial CUBA leaky-integrate-and-fire neurons and 130 million synapses. Loihi also provides a programmable microcode learning engine for on-chip SNN training. A Loihi chip also consists of 3 Lakemont cores which help with advanced learning rules and with managing the neuromorphic cores. The Loihi design supports scaling up to 4,096 on-chip cores and 16,384 chips.

Loihi adopts a variation of the well-known CUBA leaky integrate-and-fire model that has two internal state variables, the synaptic response current $u_i(t)$ and the membrane potential $v_i(t)$. The synaptic response current is the sum of filtered input spike trains and a constant bias current:

$$u_{i(t)} = \sum_{j \neq i} w_{i,j} \left(\alpha_u * \sigma_j\right)(t) + b_i \tag{1.10}$$

where w_{ij} is the synaptic weight from neuron j to i, $\alpha_u(t) = \tau_u^{-1} \exp\left(-t\tau_u\right) H(t)$ is the synaptic filter impulse response parameterized by the time constant τ_u with $H(t)$ the unit step function, and b_i is a constant bias. The synaptic current is further integrated as the membrane potential, and the neuron sends out a spike when its membrane potential passes its firing threshold θ_i.

$$\dot{v_i}(t) = -\frac{1}{\tau_v} v_i(t) + u_i(t) - \theta_i \sigma_i(t) \tag{1.11}$$

As shown in Eq. 1.11, the integration is leaky, as captured by the time constant τ_v. v_i is initialized with a value less than θ_i, and is reset to 0 right after a spiking event occurs. Loihi approximates the above continuous time dynamics using a fixed-size discrete timestep model.

Each synapse corresponds to a 5-tuple: (i, j, weight, delay, tag), where i, j are the source and destination neuron indices of the synapse, and weight, delay and tag are integer-valued properties of the synapse. Synaptic delays enable advanced temporal codes by delaying the accumulation of an incoming spike, while tags are useful as an additional scratch variable within the learning engine. Each synapse also associates with multiple presynaptic traces, and whereas the compartment with postsynaptic traces. They use different exponential smoothing parameters with decay α and impulse magnitude δ and evaluated as follows:

$$x[t] = \alpha \cdot x[t-1] + \delta \cdot s[t] \tag{1.12}$$

These traces enter the learning engine as input variables for synaptic weight adaptation. SNN synaptic weight adaptation rules must satisfy a locality constraint: each weight can only be accessed and modified by the destination neuron, and the rule can only use locally available information, such as the spike trains from the presynaptic (source) and postsynaptic (destination) neurons. Based on the locality constraint, the learning engine supports simple pairwise STDP rules and also much more complicated rules such as triplet STDP, reinforcement learning with synaptic tag assignments, and complex rules that reference both rate-averaged and spike-timing traces. The functional form of adaptation rules to apply to the synapse's state variables is described in sum-of-products form in terms of a set of microcode operations associated with the

synapse:

$$zz + \sum_{i=1}^{N_p} S_i \prod_{j=1}^{n_i} (V_{i,j} + C_{i,j}) \tag{1.13}$$

where $z$is the transformed synaptic variable (weight, delay or tag), $V_{i,j}$refers to some choice of input variable available to the learning engine, and $C_{i,j}$ and S_i are microcode-specified signed constants.

Chapter 2

Modular Spiking Neural Circuits for Mapping Long Short-Term Memory on a Neurosynaptic Processor

2.1 Introduction

Feedforward neural networks are unable to build representations and learn patterns of dynamic data, where the series of data have temporal dependencies. While recurrent neural networks (RNNs) address this issue with feedback connections, it is difficult to learn long temporal dependencies using vanilla RNNs [46]. Long Short-Term Memory (LSTM) improves upon RNN with a complex gated mechanism, which allows it to selectively forget, remember and output information [47], making it effective in capturing long-term temporal dependencies. Thus, LSTM has become a prominent and successful model for time-series processing. It has had recent successes in applications such as machine translation [48], image captioning [49], image generation [50], video

to text [51], etc.

The aforementioned neural network architectures are highly computationally intensive and power hungry compared to our brain. Whereas, spiking neural networks (SNN), which uses spikes for communication and computations, has the potential to be very efficient as each neuron works asynchronously in an event-driven manner with sparse spiking activity. Thus, converting these neural network architectures to SNNs can greatly reduce their energy requirements.

However, it is inefficient to implement SNNs on traditional Von Neumann architecture due to the performance gap between memory and processor. This has led to the advent of energy-efficient large-scale neuromorphic hardware that enables low power implementation of large-scale neural networks for real-time applications. One of the examples is IBM's Neurosynaptic Processor, "TrueNorth". Converting pre-trained networks to an SNN has also produced good results in pattern recognition [52] on platforms other than TrueNorth. However, almost all of these applications aim at non-recurrent networks, such as convolutional neural networks. Capturing the temporal dynamics of a recurrent network using spiking neurons is inherently difficult. With added hardware constraints in connectivity and synaptic weight precision, implementing recurrent neural networks (RNNs) for temporal sequence processing in the spike domain is still at the proof of concept level [53].

This chapter presents a design flow that overcomes the aforementioned difficulties and maps LSTM, a special case of RNN, onto a spike-based platform, and implement it using the TrueNorth processor. The main contributions of this work are summarized as the following,

1. A modular approach is presented that converts a standard LSTM to a Spiked-based LSTM and incrementally maps it onto a neurosynaptic processor.

2. To have a faithful representation of inputs, outputs and internal activation of an LSTM in spike-domain, we adopted an encoding heuristic to maintain the

consistency of spike representation throughout the network.

3. Novel neural circuit designs are presented that approximate the sigmoid and hyperbolic tangent functions. The relationship between stored membrane potential, the random firing threshold, and the firing rate is analyzed.

4. To synchronize the gated modules and achieve recurrent processing in the Spike-based LSTM, we developed a store-and-release mechanism using locally generated and globally consistent store and release clock spikes.

2.2 Spiking Neural Modules for LSTM

Most implementations of SNNs [43, 54, 55] including the TrueNorth processor, have event-driven neuron operation and asynchronous intercore communication. It reduces the hardware active power, however, also imposes a fundamental challenge to realize the LSTM. As shown in Figure 2.1a and 2.1b, the inputs and outputs of an LSTM require synchronization. The output vector in $t-1$ time step must concatenate with the input vector x_t in t time step to calculate the new output vector. This level of synchronization requires h_{t-1} special neural circuits on an event-driven hardware platform. Other challenges include the difficulty in representing both positive and negative numerical values with spikes, and a lack of direct support of nonlinear activation functions such as sigmoid and tanh in the spike domain.

In this section, we address the aforementioned challenges and present some key techniques that facilitate Spike-based LSTM and its mapping onto any crossbar-based neurosynaptic processor. We will first discuss how values are represented using different spike encoding schemes. Then we will describe the spike-based constituent modules of LSTM and how to maintain the temporal relation of these modules' activities with specifics for TrueNorth Corelet configurations. Finally, we present the mapping algorithm.

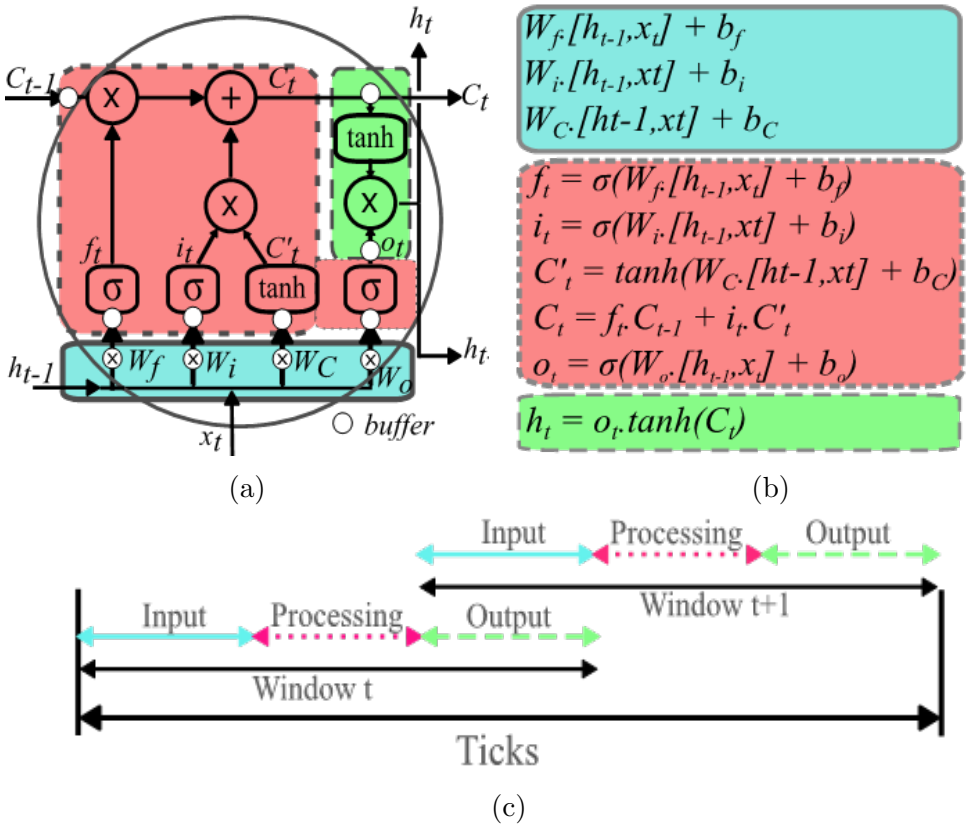


Figure 2.1: (a) LSTM color coded based on operation phase (b) LSTM equations color coded to represent operations in specific phases (c) 3 phases and partial pipelining

2.2.1 Temporal Behavior of Neurons in LSTM

Synchronization is necessary to maintain the temporal dynamics of recurrence in a recurrent network. It is also the key to enable the complex gating mechanism of the LSTM, which requires the synchronized interactions between inputs, gate outputs and the cell state feedback. Given the inherent difficulty in representing this synchronization with asynchronously spiking neurons, we propose a store-and-release mechanism. It is implemented with special neural circuits where neurons operate in two modes, store and release. During the store mode, the neurons gate their outputs, receive input spikes, and accumulate their membrane potential. During the release mode, the neurons issue output spikes at an average rate proportional to its membrane potential, either stochastically or in a burst. Two internal clock signals control

when a neuron enters or exits the store or release modes by applying a high negative or positive potential respectively. How to configure and connect asynchronous neurons to form such synchronous neural circuits will be discussed in Section 2.2.1.

Using the store-and-release mechanism, the entire LSTM network is divided into three partitions as shown in Figure 2.1a and 2.1b. They are referred to as input, processing, and output partitions and color-coded using blue, red, and green, respectively. The system operation has two phases, processing phase and I/O phase. Each partition must only generate output spikes during its designated phase, i.e., processing partition spikes during the processing phase and input and output partitions spike during the I/O phases. Please note that during the I/O phase, if the output partition is working on the ith data, then the input partition is working on the $(i+1)$th data.

The output of each functional module is a spike train. These spike trains are buffered at every partition and then released during the associated phase. Except for the input partition, all inputs of the processing and output partitions are buffered using the store-and-release neurons, which are shown as white circles in Figure 2.1a. The inputs to the processing partition store during the I/O phase and release in the processing phase, while the inputs to the output partition store in the processing phase and release in the I/O phase. For the input partition, one of its inputs h_{t-1} is released in the I/O phase, and the other input, x_t, is an external signal. By careful control, we can make sure that x_t is released only during the I/O phase as well. Thus, we can essentially overlap the activities of input and output partition in the I/O phase. The x_t and h_{t-1} go through the input partition and are buffered at the input of the processing partition during the I/O phase. During the implementation, the store-and-release neurons will be merged into their subsequent function modules and be implemented as store-and-release tanh or store-and-release sigmoid, as we will present in Section 2.2.3. The only exception is the store-and-release neurons before the dot product, which will stay stand-alone.

Figure 2.1c shows how these three partitions operate alternatively. The duration of each phase is referred as phase length (PL). The input partition works on the matrix-vector multiplications to generate the operands for the forget, input and output gates during the I/O phase. They are buffered by the store-and-release neurons at the input of the processing partition. In the processing phase, these neurons release what they have stored, and the processing partition generates the cell state (C_t) and partial output (o_t), which are buffered by the store-and-release neurons at the input of the output partition. In the next I/O phase, the C_t and o_t is released and used to calculate the h_t, which is then further forwarded to the input partition to calculate the matrix-vector multiplications again. The transition of phases is maintained through local clock (globally consistent) spikes which produce the store and release spikes for all the store-and-release capable neurons.

2.2.2 Encoding and Spike Representation

Since spikes are binary (on-off), they are inherently difficult to be used to represent both positive and negative values. This is another challenge as in an LSTM it is difficult to avoid numerical values (i.e. inputs, outputs and activations) ranging in both negative and positive directions as the weights learnt can be negative and also the tanh function outputs values in the range -1 to 1. A simple solution is to constrain the values during training by replacing the hyperbolic tangents with ReLUs, which have no negative range and have an open-ended positive range. ReLU improves the performance of vanilla RNNs [56] because of to its ability to stop vanishing gradients. However, vanishing gradients is no longer a problem in LSTM due to its gating scheme. On the contrary, using unbounded activation functions like ReLU in an LSTM can cause it to diverge, thus resulting in worse performance [57]. Therefore, we choose to keep the Tanh function but solve the data range problem by using positive and negative channels of spikes respectively.

The inputs and outputs of an LSTM are rate-coded, where the firing rate is determined by the phase length (PL) and the max value (mx) to be represented in that phase. If we scale up the trained weights with a scaling factor sf, the input and output should be scaled down by the same factor. Therefore, the number of spikes (nS) needed to represent value 1 can be calculated as: $nS = \frac{PL}{(mx*sf)}$.

To represent a numerical value Iv, the spike firing rate is set to $\frac{(Iv*nS)}{PL}$, and n spikes in a phase represent the value: $RPValue = \frac{n}{nS}$.

The choice of mx , phase length PL, and scaling factor sf defines the precision of values that can be represented by spikes, as the minimum resolution is one spike, which represents $1/nS$(i.e. $\frac{(mx*sf)}{PL}$) in terms of numerical value. For example, let the phase length $PL = 100$and max value in the phase be $mx = 5$with scaling factor $sf = 1$. The number of spikes needed to represent value 1 is $nS = \frac{100}{(5*1)=20}$. That means a single spike in that phase represents a value of 1/20. So, if we want to represent a value $RPValue = 3$, then we expect to have $3 * nS = 60$ spikes in the phase. Scaled weights can also be balanced by scaling the activation which we discuss in Section 2.3.2.

All internal variables are rate-coded, except C_t. We found that the cell state C_t needs to have more accurate representation, because any error on this variable will be accumulated due to the feedback path. The stochastic rate coding is convenient when implementing multiplication as it requires only an AND function, however, it introduces not only rounding error but also random error due to stochastic sampling. Previous work shows that the spike burst code, where the numerical value is represented by the number of spikes that burst in a window, has a much higher correlation with the numerical value to be represented [58]. Therefore, we encode C_t using spike burst code and use the spike-burst neurons for the sum function.

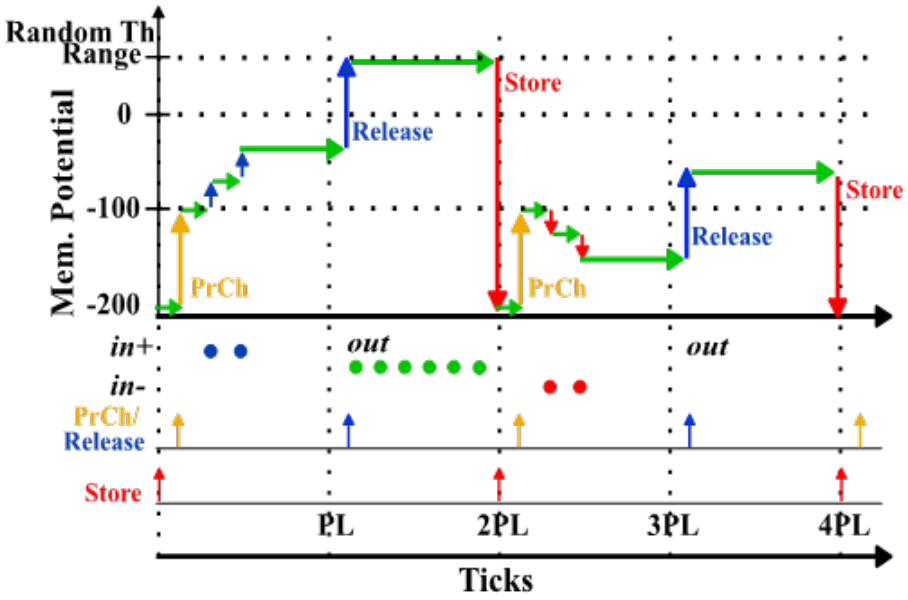


Figure 2.2: Store-and-release mechanism

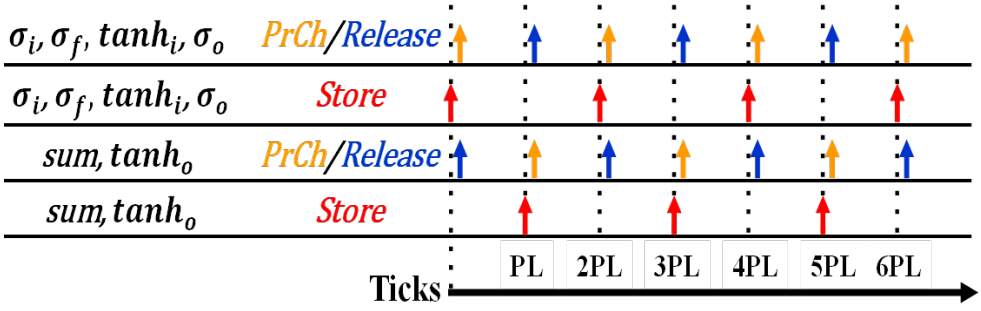


Figure 2.3: Store and release clock spikes for all gates

2.2.3 Spike-based LSTM Constituent Modules

Figure 2.1b shows a general LSTM unit consisting of the sigmoid gates, hyperbolic tangent, dot products and sum. To implement a Spike-based LSTM (S-LSTM), we approximate these modules using spiking neurons. The structure of these modules can be implemented on any spike-based platform. On a crossbar-based neurosynaptic hardware, these modules will be mapped across cores based on the consideration of the fan-in and fan-out and other hardware constraints. On TrueNorth, these modules are in the form of corelets. The corelets will be further connected to form the full spike-based LSTM (S-LSTM.)

2.2.3.1 Store-and-Release neurons

Store and release mechanism is implemented using a neuron with a high negative threshold where it saturates. The store and release clocks are two inputs associated with large negative and positive weights, respectively. A spike on the store clock turns on the store mode by pushing the membrane potential to the negative threshold, during its membrane potential rises as it accumulates the input spikes. The negative initial state guarantees that the raised membrane potential is still below the firing threshold, therefore, no output spikes are generated during the store mode. A spike on the release clock pushes the membrane potential to 0 or higher if input spikes are received during the store mode, and the neuron starts generating output spikes.

A drawback of the above scheme is that the neuron is unable to collect negative spikes at the beginning of the store mode as its membrane potential is already at the negative threshold where it saturates. We get around this problem by issuing a pre-charge spike on the release clock immediately upon entering the store mode to push the membrane potential to an intermediate level between the negative threshold and 0 to allow collecting negative spikes. Figure 2.2 shows an example where the neuron enters the store mode through a negative potential of -250 and enters release mode through two positive potentials of $+100$ administered by the store (red), pre-charge (yellow) and release (blue) clock spikes respectively. Figure 2.3 shows all the clock signals in the S-LSTM.

At the beginning of the release phase, the neuron's accumulated membrane potential (AMP) equals to the total number of net input spikes that it collected during the store mode. This is preserved throughout the release phase. The $RPValue$ stored in the neuron can be calculated as $RPValue = \frac{AMP}{nS}$. To generate output spikes, a random number drawn in the range [0, $RThR$], where $RThR$ stands for the random threshold range, becomes the threshold. If this number is less than the AMP, then an output spike is generated. During a phase window PL, the expected number of spikes

generated in this way is $PL \times \frac{AMP}{RThR}$. When we set $RThR$ to PL, the number of the output spikes equals to the total number of net input spikes, and the neuron relays input to the output without any transformation. In the actual implementation, almost all store-and-release neurons are merged to its subsequent sigmoid and tanh gate. $RThR$ should be selected differently due to the squash and linear transformation of these functions.

2.2.3.2 Input Collection Module (IC Module)

In an LSTM, the inputs to each gate are fully connected. As in any fully connected layer in a neural network, there is a matrix-vector multiplication between inputs and the weight matrices. From the LSTM equations, we see that the weight matrices W for each gate are multiplied by the input vector x and the previous time-step's output vector h_{t-1} respectively along with the biases. We develop a parameterized module called input collection (IC) module to implement such matrix-vector multiplication. As shown in Figure 2.4d, each input and output are represented using two channels to accommodate both positive and negative values. Specifically, for TrueNorth, which allows for a maximum of four possible integer weights per neuron in a core, we decide to use 4 axons with weights 1,2,4, and 8 to approximate weights up to +15 in each channel. The absolute weight is assigned to the positive or negative output channel based on the resultant sign of the product of input and axon weights. Given the positive (+) and negative (-) channels of input x and output y, to calculate $y = ax$, if a is positive, then y- will be connected to x-, and y+ be connected to x+. Otherwise, if a is negative, then y- will be connected to x+, and y+ be connected to x-. For both cases, the synaptic weights of the connections are $|a|$. For example, in Figure 2.4d, the connection shows the relation that: $out1 = 5 \times (in1)$, and $out2 = (-3) \times (in1)$. Hence, our two-channel weight mapping with 4 axons is capable of approximating 5-bit signed precision weights instead of 4-bit precision [53] on the TrueNorth chip.

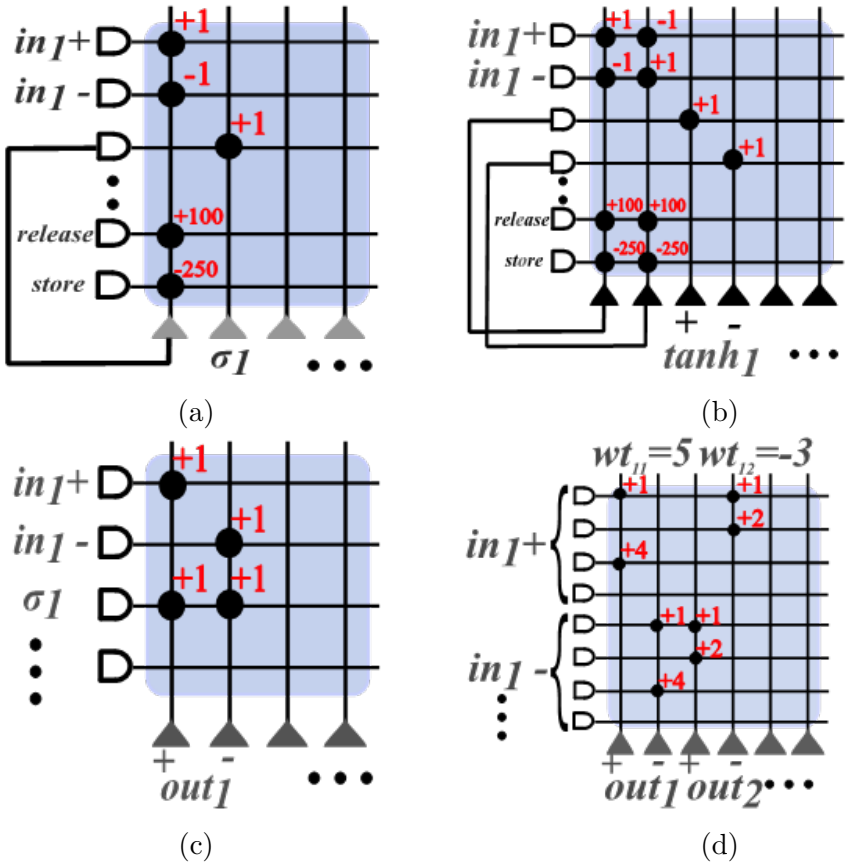


Figure 2.4: (a) Sigmoid module (b) Tanh module (c) Dot product module (d) IC module

The above design results in positive and negative outputs in their respective channels. We use ReLU neurons for the output, which produces bursts of spikes equal to the accumulated membrane potential when the threshold is 1. The two channels (i.e., positive and negative) are used in every input of the sigmoid and tanh gates in the processing partition, which is then be merged to achieve the net results.

This module is also parameterized to accommodate matrices/vectors [W_i, W_h, X, Y] of various sizes and the mapping scales across multiple cores depending on the sizes of those matrices/vectors.

2.2.3.3 Gate Modules

Due to minimal cut points and linear interpolation between those cut points, piece-wise linear functions are computationally efficient [59]. And these cut points and linearity are more conducive than a smooth non-linearity to rate coding where the spiking rate determines the computed values. Spiking rates of these gate modules have definite max and min (0 and 1) and within this range, the rate is linearly proportional to the number of input spikes or the accumulated membrane potential of a neuron.

LSTM uses sigmoid gates to allow the flow of input, cell state, and output, and uses tanh to squeeze the inputs and outputs to a range. To implement this activation function using neurons with linear responses, we develop modules which produce a hard sigmoid and a hard tanh behavior with store and release capability and use them during both training and recall.

The original sigmoid function as shown in Figure 2.5a is given as

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

Eq. 2.1 can be represented as a hard sigmoid using a piece-wise linear function with slope m_{sig}.

$$\sigma(x) = max(0,\ min(1, x * m_{sig} + 0.5)) \tag{2.2}$$

Similarly, the original hyperbolic tangent function as shown in Figure 2.5b is given as

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.3}$$

We represent Eq. 2.3 as a piece-wise linear function with slope m_{tanh} given as

$$tanh\,(x) = max(-1,\ min(1, x * m_{tanh}) \tag{2.4}$$

As positive and negative channels are used to represent the positive and negative values, we rewrite Eq. 2.4 as:

$$\begin{cases} tanh(x)\,(-) = |\max\,(-1, \min\,(0, x * m_{tanh}))|\,, \\ tanh(x)\,(+) = \max\,(0, \min\,(1, x * m_{tanh})), \end{cases} \tag{2.5}$$

During the release mode, the spikes generated by the gate modules are rate-coded. The value of Eq. 2.2 and 2.4 are encoded by the firing rates in a phase window. These activation functions are implemented using stochastic neurons, which generate stochastic rate-coded outputs. A stochastic neuron maintains a membrane potential m, which is the sum of its weighted inputs plus any possible offset. It firing threshold is generated as a uniformly distributed random number in the closed interval [0, $RThR$], where $RThR$ is the random threshold range. It is not difficult to see that the probability that an output spike will be generated can be calculated as

$$y = \frac{m}{RThR} \tag{2.6}$$

where y is the firing rate (i.e., the probability of $x > uniform\_random(RTHR)$) which represents the effective output of the activation function.

So, from Eq. 2.2 and 2.6, the hard sigmoid can be rewritten as

$$\sigma(x) = \min\left(1, \max\left(0, \frac{AMP + offset}{RThR}\right)\right) \tag{2.7}$$

where $x$is the RPValue (which equals to AMP/nS) accumulated from the input, AMP is the membrane potential accumulated from neuron inputs, and *offset* is the offset membrane potential applied. The firing rate saturates at 0 when $AMP + offset \leq 0$ and 1 when $AMP + offset \geq RThR$, thus producing the piece-wise

linear sigmoid activation. Since x in Eq. 2.2 is the $RPValue$ ($i.e.$ $= AMP/nS$) accumulated from the input, by setting $RThR = \frac{ns}{m_{sig}}$,and $offset = \frac{0.5ns}{m_{sig}}$, we can ensure that Eq. 2.2 and 2.7 are equivalent.

Similarly, from Eq. 2.4 and 2.6, hard tanh can be rewritten as

$$tanh\,(x) = ma\mathbf{x}\left(-1, mi\mathbf{n}\left(1, \frac{AMP}{RThR}\right)\right) \tag{2.8}$$

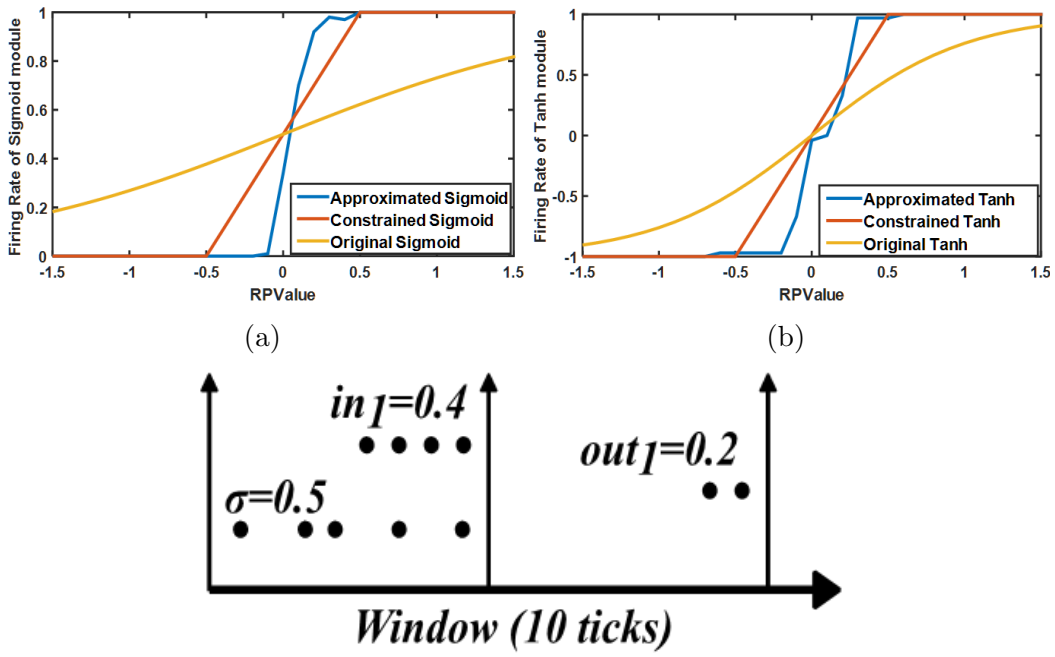
For both channels, the firing rate saturates at 0 when $AMP \leq 0$ and saturates at 1 when $AMP \geq RThR$, thus producing the piece-wise linear hyperbolic tangent activation. By setting $RThR = \frac{ns}{m_{tanh}}$ we can ensure that Eq. 2.4 and 2.8 are equivalent.

The choice of $RThR$ for both sigmoid and hyperbolic tangent depends on the slope of the linear portions of these activation functions in Eq. (17) and (23),and that slope depends on the choice of steepness and scaling of the activation function. Details of the choice of $RThR$ are discussed in Section 2.3.2 and 2.3.5.

2.2.3.4 Dot Product Module

The inputs of the dot product module are two-channeled inputs C_t, which is burst coded, and a rate-coded sigmoid as shown in Figure 2.4c. The dot product allows the gating function to regulate the flow of information. Because the output of sigmoid function has numerical values between 0 and 1 and is stochastically rate-coded, we explore the stochastic nature of the input and perform the multiplication using stochastic computing. A simple logical AND of the two spike streams is used as the multiplication.

An example is given in Figure 2.5c. In a window of 10 ticks, there are 5 spikes from the sigmoid, which represents $\sigma = 0.5$, and 4 spikes in the in1, representing $in1 = 0.4$. The logical AND produces 2 spikes at the output representing $out1 = 0.2$. The logical AND operation can be easily realized using an integrate and fire neuron



(a) (b)

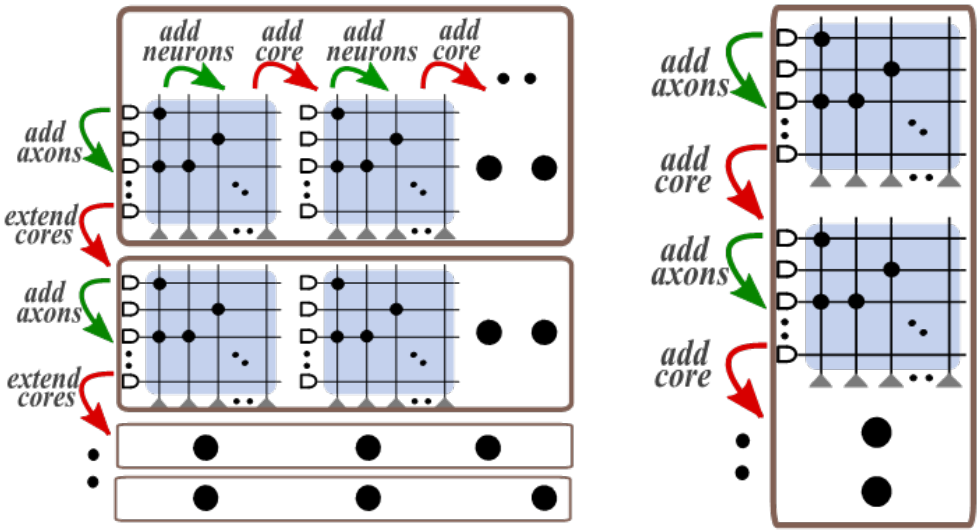
(c)

Figure 2.5: (a) Sigmoid (b) Hyperbolic Tangent activations (c) Dot product through logical AND of spikes

with threshold 2, by setting its input synapses weight and leak to be +1.

2.2.4 Mapping Algorithm

In crossbar-based neurosynaptic hardware, the number of fan-ins and fan-outs are limited, for example, a single TrueNorth core consists of 256 neurons and axons. Thus, there is a distinct fan-in and fan-out constraint. To deal with this constraint and freely map LSTM networks of arbitrary sizes, we develop an incremental mapping algorithm. The IC module's size depends on the number of LSTM units as well as the number of inputs. Thus, they are mapped and extended across two dimensions (axons and neurons). Whereas the sizes of other modules are only dependent on the number of LSTM units. Thus, these modules are mapped and extended only across one dimension (axons). As shown in Figure 2.6, the mapping is incremental. New cores are added only when the resources of the current core are filled up. This results



(a) (b)

Figure 2.6: (a) Algorithm 1 (b) Algorithm 2 in action

in the number of cores and the average neuron density (number of neurons used per core) increasing in steps of the number of LSTM.

2.3 Workflow

The LSTM network can be directly converted to SNN utilizing the spike-based constituent modules of LSTM, and the encoding and spike representation schemes described in Section 2.2.2. However, the approximations, such as using a piece-wise linear function to replace the original sigmoid and tanh and using rate or burst coding to approximate real values, will give approximation errors. Furthermore, most neurosynaptic hardware has a precision limitation on the synaptic weights. This limitation is even more visible in TrueNorth which utilizes integer weights. Thus, mapping this trained network onto TrueNorth induces potential rounding errors. It is necessary to constrain the network before and during training and accommodate these constraints during implementation. In this section, we will discuss the "constrain-then-train-then-approximate" workflow that minimizes potential errors on SNN conversion and

hardware implementation. The discussion is carried out using TrueNorth as an example hardware platform, however, the workflow can easily be adapted for other neurosynaptic chips.

2.3.1 Constrain

To avoid potential errors caused by various approximations during the SNN conversion and hardware implementation, we will consider these approximations in the training process as types of constraints. This requires us to set a plan of how to carry out the approximation before training the network. The plan includes which algorithm to be used for the approximation and how to set parameters in the algorithm.

2.3.2 Steep Activation Functions

The rounding errors introduced by the rounded weights are especially harmful when it propagates through the activation functions. The small changes of the input are visible in the output of an activation function if it falls in its linear operating region. Thus, a steeper slope of the activation function reduces the linear range of the input, hence, limits the propagation of rounding errors.

In Eq. 2.2 and 2.4, m_{sig} and m_{tanh} represent the slopes of piece-wise linear sigmoid and hard tanh respectively. To maintain the same linear operating range for the sigmoid and tanh, we must have m_{tanh}=2*m_{sig}. Here, we choose $m_{sig} = 1$ and $m_{tanh} = 2$ during training and inference such that the linear operating range for both sigmoid and tanh is from -0.5 to $+0.5$. The constrained activation functions shown in Figure 2.5a and 2.5b. For the gate modules in Eq. 2.7 and 2.8, the change in the slope changes the choice of $RThR$.

2.3.3 Weight Quantization

Binary and ternary integer weights have been used to produce close to state-of-the-art results for feed-forward network architectures [60, 61], but both our preliminary work and the existing research [62] show that it is difficult to train LSTMs with binary and ternary integer weights. Recently, quantization methods have shown promising results for RNNs [63]. But with these quantization methods, large performance degradation occurs when quantizing weights to 2-bit or l-bit numbers. This is accredited to the unbalanced distribution of the weights across the quantized values, which occurs when these quantization methods employ uniform quantization on a bell-shaped distribution of parameters with sporadic large outliers. As shown in Figure 2.7b for 2-bit quantization, for the original weights in Figure 2.7a, there is a large discrepancy between the number of weights in the two middle quantized values and the two outer quantized values. Thus, we utilize a balanced quantization approach as proposed in [64, 65] which focuses on producing balanced distributions of quantized values of data rather than preserving the outliers. The goal is to produce weights uniformly distributed across the quantized values as shown in Figure 2.7c.

Consider a random variable w in a close interval [0, 1]. As an entry in the weight matrix, w follows a bell-style distribution as shown in Figure 2.7a. Using the traditional k-bit uniform quantization, the quantized value $Q_k(w)$ can be calculated as:

$$Q_k(w) = \frac{1}{2^k - 1} \left\lfloor \left(2^k - 1\right) w + \frac{1}{2} \right\rfloor$$

The distribution of 2-bit uniformly quantized values of variables in Figure 2.7a is shown in Figure 2.7b. As we can see, a large portion of the variables are quantized into the 2 levels in the center. This effectively reduces the data precision from 2-bit to 1-bit.

A k-bit balanced quantization of w is calculated as

$$Q_k^{bal}(w) = Q_k\left(\frac{w`}{scale}\right) * scale \tag{2.9}$$

where $scale = mean(abs(W)) * 2.5$is the approximation of the parameter dependent adaptive threshold $3 * median(abs(W))$ to produce an auto-balancing effect [64] at each layer using a standardization transform to produce $w`$ with a close interval [0,1] from w using the following equation:

$$w` = \max\left(-0.5, \min\left(0.5, \frac{w}{scale}\right)\right) + 0.5$$

Scaling these weights and rounding to integer can produce rounding-off errors. Thus, to minimize the rounding-off error from scaling and rounding, we introduce a rounding in the quantization flow during training which rounds the weights to the n^{th} decimal place given as the following:

$$RQ_k^{bal}(W) = round\_nD(Q_k^{bal}(W)) \tag{2.10}$$

where $round\_nD(x) = \frac{round\_to\_integer(x*10^n)}{10^n}$. This rounding enables scaling by 10^n and produces integer weights and thus removes rounding-off error.

The balanced quantization method combined with rounding during training can be utilized to get trained weights at any fixed-point quantization level for an inference hardware that utilizes fixed-point weights.

With weight quantization, along with minimizing the rounding error, the number of axons can be reduced from 4 to 2 to allow for more efficient usage of the axons.

2.3.4 Train

The derivatives of the quantization function in Eq. 2.9 and 2.10 are equal to zero or undefined as the function $Q_k(X)$ and $RQ_k^{bal}(x)$ is non-differentiable. This causes hindrance in training the network using backpropagation. So, we adopt the Straight

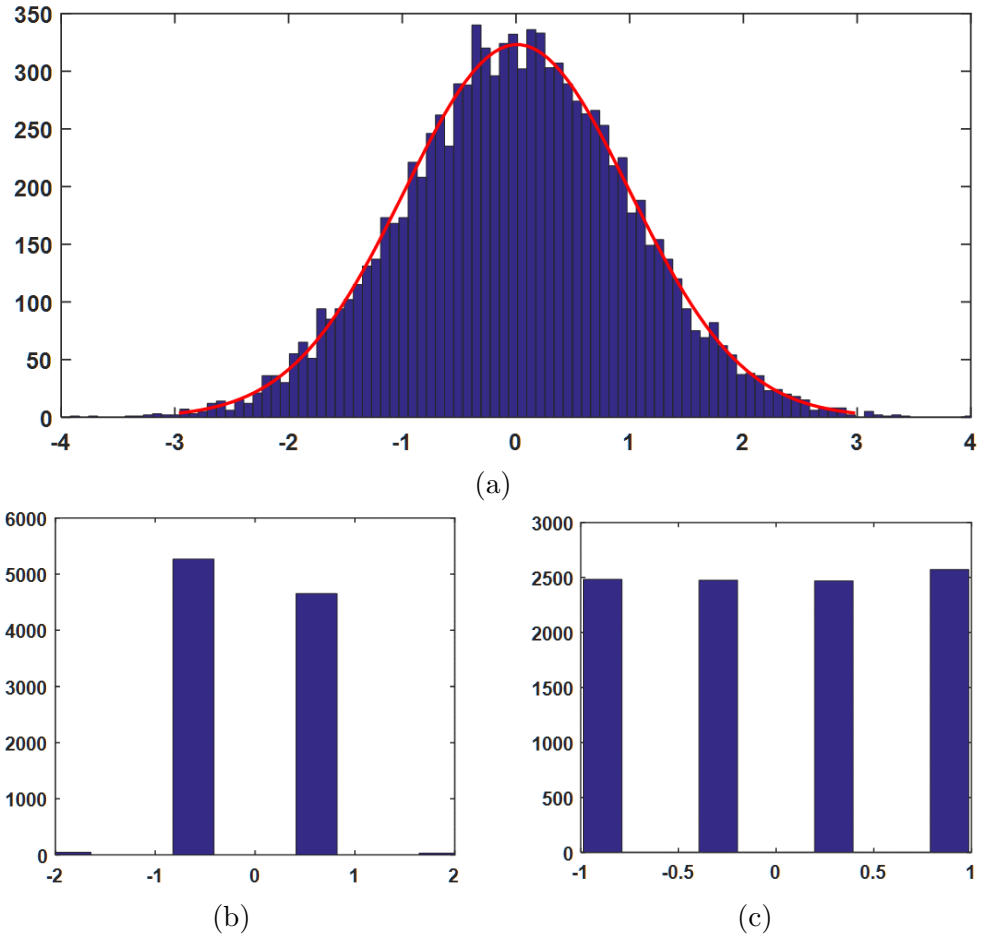


Figure 2.7: Histograms for (a)Original (b)Standard quantization (2-bit) (c) Balanced quantization weights (2-bit)

Through Estimator (STE) proposed in [66, 67], and substitute the gradient with respect to the quantized value $\frac{\partial c}{\partial y}$ for the gradient of the original value $\frac{\partial c}{\partial x}$ as if propagating the gradient through a linear function. As STE produces approximation noise while computing gradients, we limit it only to the necessary functions that are non-differentiable. In Eq. 2.9, the STE method is applied to the floor function, and the forward and backward propagation of the floor function is defined as the following

$$Forward : y \leftarrow floor(x)$$

$$Backward : \frac{\partial c}{\partial x} \leftarrow \frac{\partial c}{\partial y}$$

Similarly, in $RQ_k^{bal}(x)$ of Eq. 2.10, the STE method is applied to the round -to-integer function and ts forward and backward propagation is defined as follows.

$$Forward : y \leftarrow round\_to\_integer(x)$$

$$Backward : \frac{\partial c}{\partial x} \leftarrow \frac{\partial c}{\partial y}$$

The forward propagation is performed using balanced quantized weight coefficients, the weight is also updated based on the derivative calculated with respect to the quantized weights.

2.3.5 Approximate

After constraining and training the network, we have the quantized weights trained with n^{th} decimal precision. Now, to implement the trained network on a constrained TrueNorth chip, some linear transformation is necessary.

As the synaptic weights in TrueNorth and many other neuromorphic hardware only have integer precision, it is necessary to scale the weights and round them to integer values. With n^{th} decimal precision, the scaling is simplified by using scaling factor $sf = 10^n$, which directly generates integer weights.

To keep the output unchanged, it is necessary to counter the scaling up of the weights by a scaling-down using the same scaling factor. The weighted sum of the IC module can be written as: $x = \sum W * Input + b$. In our previous work [30], when the weights W and b is scaled up by sf, we scaled down the Input by sf to preserve the integrity of y. We call this mechanism the scaled-input approach. This scaling mechanism is straightforward but reduces the precision of the input, which is already low given that the inputs are represented in terms of rate-coded spikes. For example, in a set-up with PL=50 and mx=5, it takes 10 spikes to represent a value "1", hence the data precision is 1/10. When the scaling factor sf is set to 10, data precision

is drastically degraded to 1. Thus, in the previous work, it was necessary to use a minimal scaling factor.

Because the outputs of the IC module always go through an activation function, instead of scaling down the inputs of the IC module, in this work we integrate the down scaling factor into the activation function and we call this scaled-activation approach. This is similar to scaling the output x of the IC module instead of its input. Since x is an accumulation of the inputs, it has a larger range than each individual input, and scaling x usually has less impact on the accuracy than scaling each input individually. This scaling will preserve network accuracy and allow the use of a larger scaling factor.

In Section 2.2.2, we know that the activation functions (i.e., sigmoid and hyperbolic tangent) are formulated in terms of AMP and $RThR$. We also know that by setting $RThR = nS/m_{sig}$ and $offset = \frac{0.5ns}{m_{sig}}$, we can use a stochastic neuron to implement the hard sigmoid function given in Eq. 2.2. For scaled-activation scaling, we set $RThR = (nS * sf)/m_{sig}$, and set the $offset = 0.5 * RThR$.

Similarly, without scaling, we set $RThR = nS/m_{tanh}$ to implement the hard hyperbolic tangent function using a stochastic neuron. However, with activation-scaling, the random threshold range should be set to $RThR = (nS * sf)/m_{tanh}$.

The blue lines in Figure 2.5a and 2.5b plot the activation function with the scaling factor $sf = 10$ when implemented using a TrueNorth neuron. Compared to the original function and the constrained function, it has a steeper slope due to the scaling factor. As we utilize stochastic neurons to approximate the constrained activation functions, the quality of the approximation is highly dependent on the random threshold generation and the length of the window over which the approximation is made. In Figure 2.5a and 2.5b, the approximated activation functions using TrueNorth have some irregularities, which can be attributed to imperfect random number generation for the random threshold and limited window length for the approximation.

2.4 Experiments

In the following experiments, we compare the Spike-based LSTM mapped on TrueNorth with the standard LSTM implemented using Keras with TensorFlow backend. We use spike-based LSTM simulated on a full precision SNN simulator [68], as an ideal case scenario as no scaling and rounding are necessary. Its result is denoted as FPW-SNN. Please note that for FPW-SNN, the synaptic weights are not quantized, however, the neuron activities are quantized to 1/ns precision. All other results with prefix S-LSTM are results from TrueNorth. The results with prefix LSTM are results from a non-spiking LSTM implemented on Tensorflow.

For all experiments, the network is comprised of one hidden layer of LSTM units. The networks are trained with two setups: scaled-input scaling without weight quantization and scaled-activation scaling with 2-bit weight quantization. The quantization is done with a single decimal place precision (n=1) such that we set *sf=10* to achieve quantized integer weights. And to account for the large *sf* which reduces the slope, we train with a steep slope for both sigmoid (m_{sig}=*1*) and tanh (m_{tanh}=*2*). The networks are trained also with weights constrained to the range -1 to 1. Spike-based LSTM is set to mx=5 and we vary the phase lengths to show how data precision can affect the performance, speed, and accuracy. A longer phase length means high precision of the internal data of the LSTM.

2.4.1 Parity check / XOR problem

Parity check of a bit stream is a classic problem that is difficult to solve with a standard feed-forward network, but simple to solve with an RNN. In this problem, we have to determine at each input of a sequence of binary inputs whether the number of 1's observed so far in the sequence is even or odd. Here, we train LSTM with one hidden layer containing 2, 4, and 10 LSTM units on 9000 varying length binary

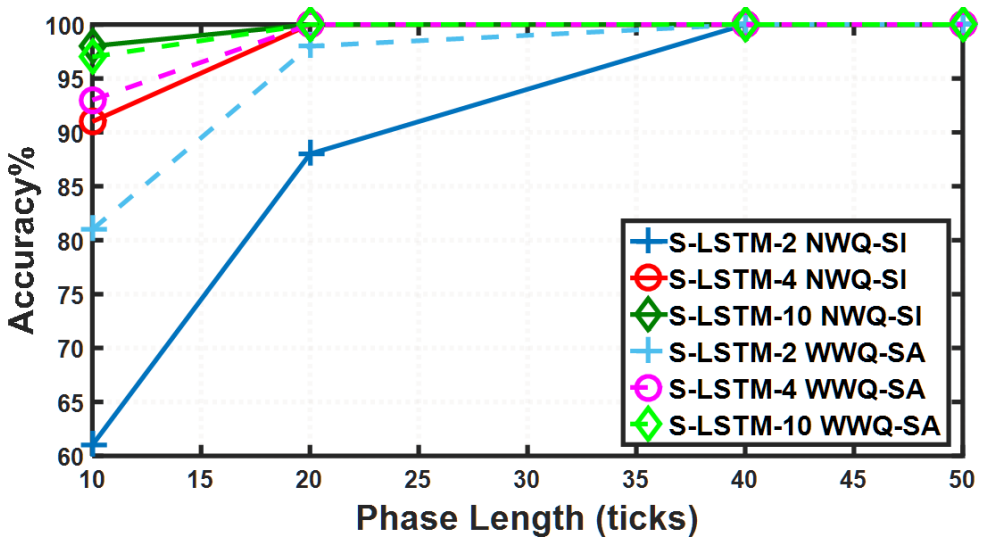
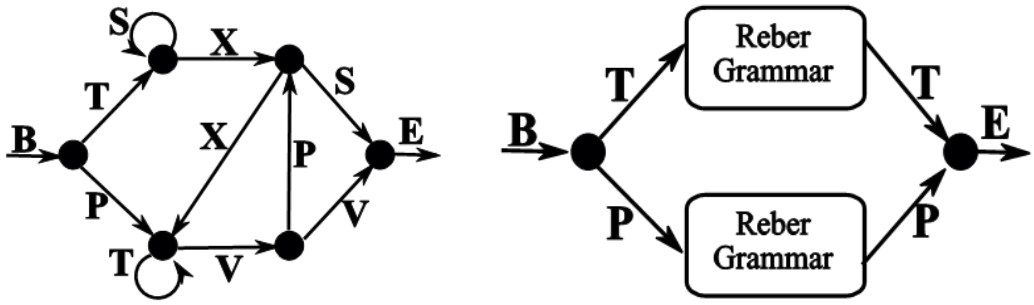


Figure 2.8: Accuracy vs Phase Length for Spike-based LSTM [NWQ-SI: No Weight Quantization with Scaled Input, WWQ-SA: With Weight Quantization with Scaled Activation] (Keras/Tensorflow counterparts have 100 % for all cases)



(a) (b)

Figure 2.9: (a) Reber Grammar (b) Embedded Reber Grammar

sequences with a maximum length of 20. Then it is tested with 1000 sequences on TrueNorth with varying phase lengths.

In Figure 2.8, we can see that the accuracy improves with the phase length as this increases the precision of activations. The accuracy also increases when the number of LSTM units is increased. The scaled-activation and weight quantization method also improve the results as it removes rounding-off error from weights and reduces the quantization error by preventing a reduction in precision. The full precision simulation of the spike-based LSTM led to the accuracy close to (>98%) or equal to the results on Tensorflow.

2.4.2 Embedded Reber Grammar

Embedded Reber grammar (ERG) is a popular RNN benchmark [47] and is useful for training sequences with short time lags. Figure 2.9a shows a Reber Grammar graph which is extended to an Embedded Reber Grammar in Figure 2.9b. An ERG sequence starts from the leftmost node 'B' of the ERG graph and sequentially generates a finite number of symbols by following edges until the rightmost node 'E' is reached. At some nodes, there can be two possible paths and the choice is made randomly. A sequence that follows Reber Grammar defines a path in the Reber Grammar graph. During training, the system is presented with sequences following Reber Grammar, and during testing, the system is expected to generate sequences according to Reber Grammar.

Input and target patterns are represented by 7-dimensional binary vectors, representing one symbol each, in the training set. And the task is to read the symbols one at a time and to continually predict the next possible symbol(s). Input vectors have exactly one nonzero component, but the target vector could have one or two nonzero components representing one or two possible paths. The prediction is considered correct if it predicts either one or both possible symbols. Here, we train LSTM networks with one hidden layer containing 10, 30, and 50 LSTM units on 5000 ERG sequences with maximum sequence length 36. We tested on 500 ERG sequences on TrueNorth with varying phase lengths.

Again, we see that the phase length has a direct impact on accuracy. For networks without weight quantization and with scaled-input (NWQ-SI), when the phase length increases above 10, the accuracy drastically improves, which means it requires the minimum data precision that can be represented by 1 spike to be higher than 1/10. The trend is noticeable in all 3 network sizes as shown in Figure 2.10. With weight quantization and scaled-activation (WWQ-SA), the accuracy gets closer to the full precision spike-based LSTM (FPW-SNN) as seen in Figure 2.10a, 2.10b and 2.10c and

achieves ˜5% improvement over NWQ-SI. The improvement is higher (5% - ˜30%) when the phase length is small. This is understandable as at lower *PL,* the precision reduced by scaled-input scaling is more noticeable than at higher *PL.*

The network using rate-coded internal cell state (S-LSTM-50RCt) instead of burst-coded one (S-LSTM-50) performs significantly worse as shown in Figure 2.10d. In average, using burst coding on C_t gives ˜30% improvement in accuracy. This, as mentioned in Section 2.2.2, is due to the accumulation of rounding error and additionally the sampling error while the spikes move from one buffer to another and then feeds back. The improvement is higher (up to 35%) when the phase length is large and lower (˜20%) when the phase length is small. This is because at smaller phase lengths the low precision of inputs and internal data has a higher impact than the sampling errors.

The scaled-activation and weight quantization (WWQ-SA) methods improve the results in this example by ˜15%. Keeping the internal cell state burst-coded is still important under WWQ-SA. It gives an extra ˜15% improvement in accuracy in average. This is because scaled-activation and weight quantization do not change the representation of the internal data, hence cannot be used to reduce the sampling error introduced by rate coding.

For networks without weight quantization, when trained without any constraints, the range of learned weights varies. If that range is wide, it is hard to find a scaling factor that can raise smaller values to the hardware supported range with low rounding errors without causing the larger values to overflow. So, we simply set the scaling factor sf to be 1. Compared to the networks trained with constraints, the rounding error of the unconstrained network is higher. However, setting $sf = 1$ leads to higher nS and better precision (i.e. lower $1/nS$) than setting $sf > 1$. Therefore, the unconstrained network (S-LSTM-50NoC) produces better accuracy than the constrained network (S-LSTM-50 <35%) at lower PL, because it allows higher data

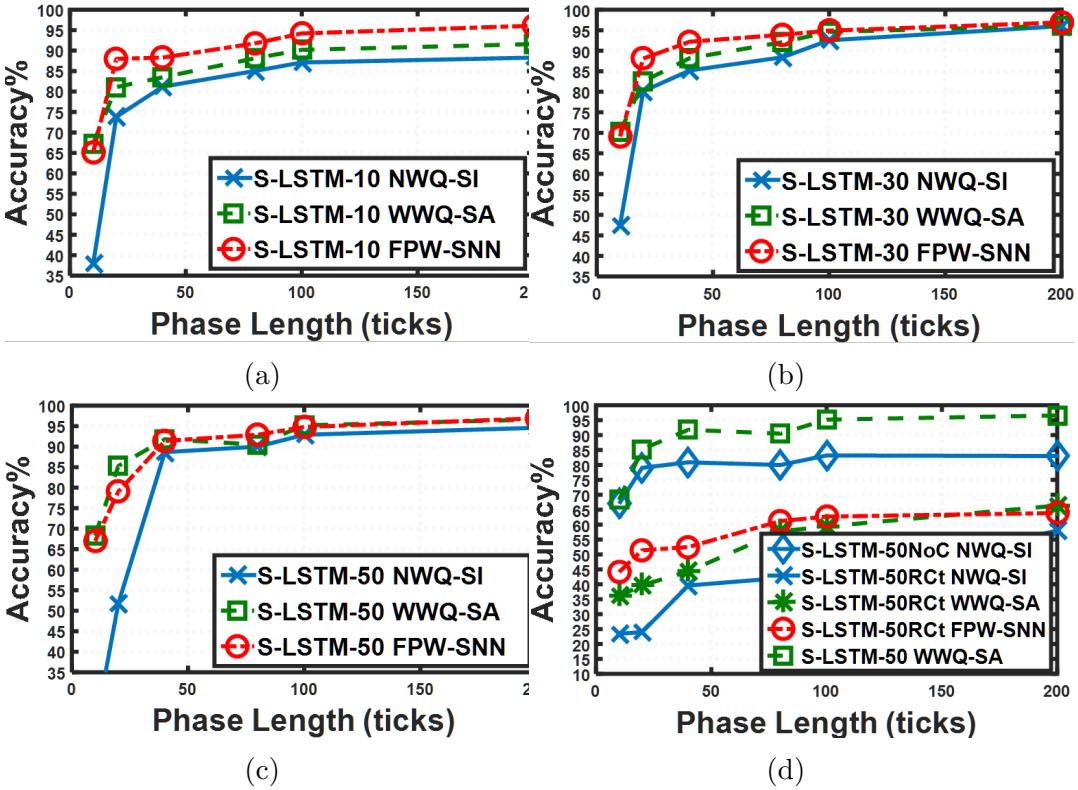


Figure 2.10: Accuracy vs Phase Length (PL) for Spike-based LSTM with (a)10 (b)30 (c)50 units (d) with rate-coded Ct and no-constraint setup [NWQ-SI: No Weight Quantization with Scaled Input, WWQ-SA: With Weight Quantization with Scaled Activation, FPW-SNN: Full Precision Weights SNN, RCt: Rated coded Ct, NoC: No Constraints](Keras/Tensorflow counterparts have 100 % for all cases)

precision. When the PL is high, the large window size already ensures reasonable data precision, so the constrained network performs better than the unconstrained version as shown in Figure 2.10d. However, for networks with weight quantization, as they produce discrete weights with single decimal precision, we always set *sf=10*.

2.4.2.1 Power Analysis

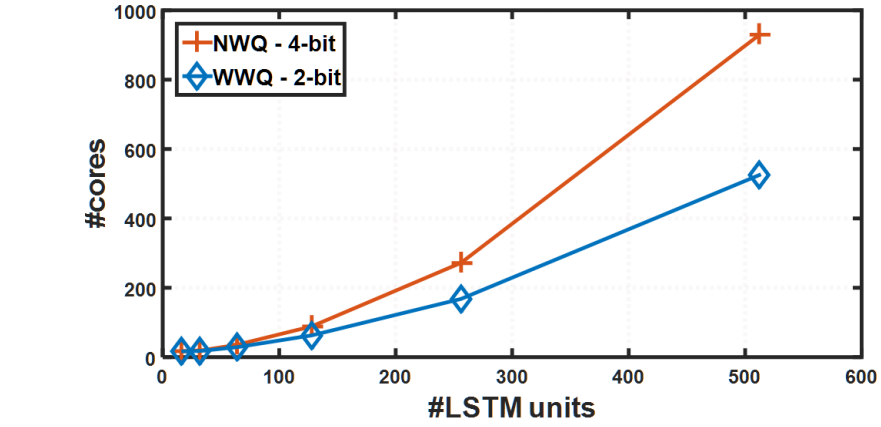
Table 2.1 shows the results of 50-unit LSTM network with scaled-input scaling and without weight quantization, and the respective TrueNorth implementations with 200 (high accuracy) phase lengths used for Embedded Reber Grammar. The table demonstrates the energy versus performance trade-off between TrueNorth and the GPUs. Compared to NVIDIA Tegra X1 (20nm technology) and NVIDIA Titan X

Table 2.1: Power and performance on different platforms (*running at faster than real time (3.5x faster)

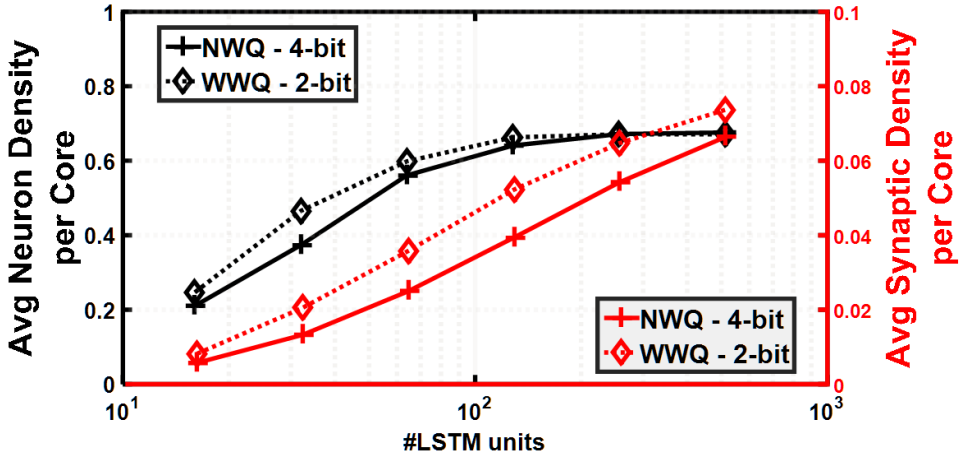
Network	**Devices**	**Time/Sample (ms)**	**Active Power (W)**	**Energy/Sample (ms)**
50-LSTM	NVIDIA (Tegra X1)	1.928	2.45	4.72
	NVIDIA (Titan X)	0.5	46.5	23.3
50-LSTM (200-PL)	IBM (TrueNorth)	400	0.00014	0.056
	*IBM (TrueNorth)	114	0.00025	0.0285

(16nm technology), the TrueNorth networks (running at normal operating frequency 1 kHz) is more energy efficient. It consumes only 56 μJ at 0.8V for 200 PL per sample and is up to 84x and 416x energy efficient than Tegra X1 and Titan X respectively. At faster than real time operation (3.5 kHz operating frequency) and the same voltage level of 0.8V, TrueNorth performs even better with 165x and 817x higher energy efficiency compared to Tegra X1 and Titan X. All measurements were taken only during inference.

Table 2.1 also shows the TrueNorth implementation is slower than the GPU implementations. In practical deployment, this factor of time delay is unsuitable, and it points to a large performance/energy tradeoff. This tradeoff is drastic mainly due to the inability to reliably train LSTM networks with very limited precision. In the experimental setup, we chose $PL = 200$ and $sf > 1$ for comparison to ensure reasonable data precision and comparable accuracy as the GPU implementation. If we can train an LSTM network with limited precision, then sf can be set to 1 and a much lower PL can be used without reducing spike's data representation precision. This will reduce the delays to a tolerable level or even close to the GPU implementation with the same or even lower energy footprint.



(a)



(b)

Figure 2.11: (a) Total number of cores used vs LSTM units, (b) Average Neuron density and Synaptic density vs LSTM units

2.4.2.2 Chip utilization

The actual power consumption only for the resources utilized on TrueNorth chip is computed by measuring chip idle power which is the leakage power P_{leak} and total power P_{total} with the network running [85]. Active power is $P_{active} = P_{total} - P_{leak}$. The scaled leakage power for the cores utilized is $P_{leak\_s} = P_{leak} * cores/4096$. Therefore, the total power consumed for the utilized resources is $P_{total\_s} = P_{active} + P_{leak\_s}$. For a given network, the active power is directly proportional to the spiking activity however, the same network can be designed to be mapped utilizing a different number of cores. To ensure minimum power consumption, it is critical to pack as many neurons as

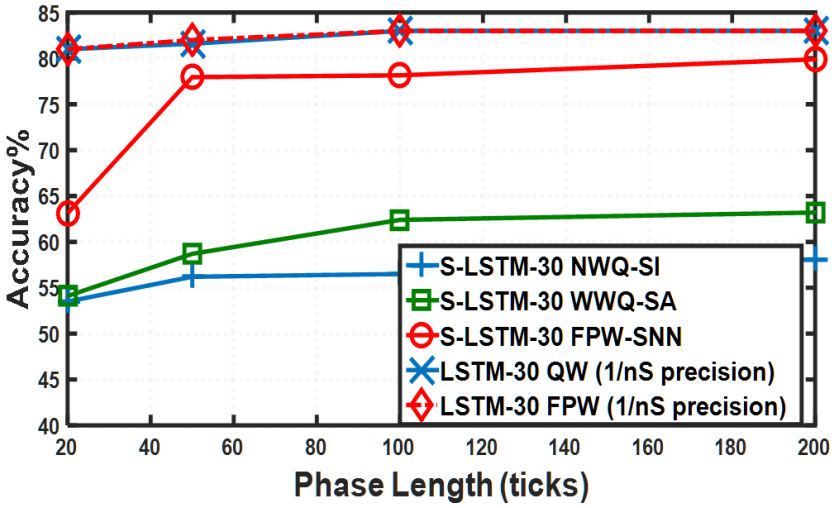


Figure 2.12: Accuracy vs Phase Length for Spike-based LSTM [NWQ-SI: No Weight Quantization with Scaled Input, WWQ-SA: With Weight Quantization with Scaled Activation, 1/nS: Tensorflow LSTM network tested with 1/nS input precision, QW:Tensorflow LSTM trained with quantized weights, FPW: with full precision weights]

possible on to each core for minimizing P_{total_s} as P_{leak_s} is the only free variable.

Packing/Efficient utilization of the TrueNorth chip can be described in terms of the number of cores used and the average neuron and synaptic density. The incremental mapping algorithm shown in Section 2.2.4 attempts to pack as many neurons as possible in a core, incrementally adding cores as required. This behavior is illustrated in Figure 2.11b which shows the average neuron and synaptic density increasing with the number of LSTM units. This suggests that with the increasing number of LSTM units, the cores are more densely packed and thus better utilized.

With weight quantization, the weights can be discretized to 2-bit or even 1-bit. This allows for utilizing only 2 axons per input channel in the ICB module instead of 4 axons. Utilizing just 2 axons per channel results in a reduced number of cores and improved neuron and synaptic density as shown in Figure 2.11a and 2.11b with the increase in the number of LSTM units. Thus, with weight quantization, the scalability of mapping networks on TrueNorth is improved.

2.4.3 Question Classification

In the previous tasks, the inputs are one-hot encoded, thus requiring no discretization of the input. In the third experiment, we utilize a task requiring discretization of the input as well. The goal of this task is to categorize a question into six coarse classes (ABBREVIATION, ENTITY, DESCRIPTION, HUMAN, LOCATION and NUMERIC VALUE) based on the sequence of words in the question sentence. The training dataset [69] consists of 5500 sentences and the test set contains 500 sentences. The LSTM network is trained on these sentences using a 50-dimensional vector representation for words called GloVe [70], learnt on aggregated global word-word co-occurrence statistics from a corpus. For this problem, each sentence is limited to 10 words. Thus, the input is a sequence of 10 50-D word vectors and the output is one of the six classes. As these word vectors are continuous values, the input is also rounded to a precision given by 1/nS.

With 30 LSTM units, the LSTM network achieves ˜86% accuracy on Tensorflow. It reduces to ˜84% when using 2-bit weights with full precision inputs. Rounding the inputs to *1/nS* precision, the performance decreases by ˜2% as shown in Figure 3.12 by the LSTM-QW data series. It is understandable as there is always some loss of information associated with quantization. Using the full precision weights on an SNN simulator, our S-LSTM achieves 79.9% accuracy which is within ˜3% of the Tensorflow implementation. However, the networks on TrueNorth still lag behind (both NWQ-SI and WWQ-SA) due to the added constraints.

2.5 Conclusion

This work presents a methodology to convert a general LSTM to a Spike-based LSTM which can be deployed in any neuromorphic hardware. A standard LSTM is divided into modules and separately approximated using spiking neurons. Here

we target benchmark applications on IBM TrueNorth Neurosynaptic Processor. On TrueNorth, the modules are in the form of corelets, which are then combined, connected, and mapped to form Spike-based LSTM networks and synchronized using a store-and-release mechanism. These networks are tested on three RNN benchmarks with promising accuracy results and high power efficiency.

Chapter 3

Stable Spike-Timing Dependent Plasticity Rule for Multilayer Unsupervised and Supervised Learning

3.1 Introduction

STDP is a learning rule that potentiates or depresses a synapse depending on the relative timing between single pre and post-synaptic spikes [10]: long-term potentiation (LTP) occurs if the pre-synaptic spike arrives before the post-synaptic spike and long-term depression (LTD) occurs otherwise. It is local as it always pertains to a pair of pre- and post-synaptic neurons. And although it is correlation-based and causality is important for the synaptic plasticity, it differs from the Hebbian learning rule as STDP also requires temporal precedence [10]. Purely rate-based Hebbian rules on its own lead to runaway processes of potentiation causing instability. That requires additional constraints and mechanisms [71, 25] for containment. Whereas

many theoretical studies concerned with non-Hebbian rules of plasticity look for desirable properties, such as a trend towards inherent stability in weight distribution, neural activity, and competition among correlated inputs. STDP rules, which are independent of the current synaptic weight (additive STDP) [72] induces competition but need mechanisms to prevent weights from either disappearing or exploding. Weight-dependent STDP rules (multiplicative STDP) [73] are inherently stable producing unimodal weight distribution but induce weak competition, thus requiring additional mechanisms to induce competition [74]. These additional mechanisms increase the complexity in their implementation. This is undesirable when it comes to large-scale or neuromorphic hardware implementations. In this work, we introduce a variation of a weight-dependent STDP rule which is inherently stable and yet simple and lends well to computationally efficient and inexpensive implementations.

STDP is known to be selective to patterns. When used in a network with lateral inhibition producing a Winner Take All (WTA) effect, STDP allows for learning discriminative features without supervision [52]. These features could either be used as intermediate features [75] or could be labelled [52] for classification. In addition to improving the STDP rule for unsupervised learning, in this work we also apply it for supervised learning by introducing teacher neurons. The unsupervised feature learning and unsupervised classifier learning are stacked forming a multilayer SNN trained with STDP.

The main contributions of this work are summarized as following.

1. We present a simplified approximation of conventional Bayesian neuron and an improved STDP rule with extended LTD window, exponential weight dependence and different learning rates during LTD and LTP. Experimental results show that the modified STDP rule provides stable and competitive learning.

2. A layer-wise approach is used to combine unsupervised and supervised STDP learning and train a multilayer SNN on MNIST dataset. It achieves comparable

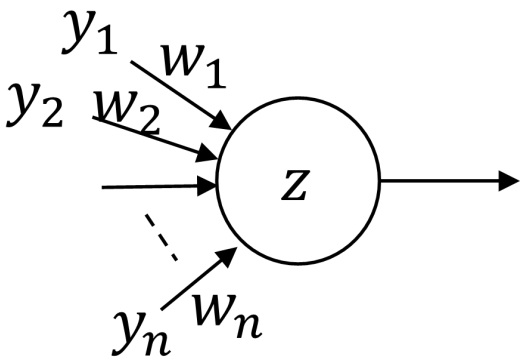


Figure 3.1: Generic neuron model

or better accuracy for handwritten digit recognition than other existing STDP approaches with the similar size of trainable parameters and manually labelled features for classification. The parameter space is analyzed to further fine-tune the network.

3. We introduce an approximation of our STDP rule named Quantized 2-Power Shift (Q2PS) rule which is hardware implementation friendly. This approximation produces comparable results on the handwritten recognition to the original STDP rule.

3.2 Simplified Bayesian Neuron and STDP rules

In this section, we describe the neuron and the synapse model and discuss the simplifications that allow for an efficient implementation. We also discuss the network architecture for the MNIST digit pattern recognition, and the methods used for its implementation.

3.2.1 Neuron Model

We utilize the generic Bayesian neuron model proposed in [76] as the starting point and simplify the overall computation model as mentioned in [68]. For this model, we

propose a stable and simplified STDP rule for efficient online learning. Bayesian model has two computational stages compared to the regular integrate and fire neuron. The first being the exponential function and the other being the Poisson firing. In the generic neuron model as shown in Figure 3.1, the membrane potential $u(t)$ of neuron Z is computed as

$$u\left(t\right) = w_0 + \sum_{i=1}^{n} w_i \cdot y_i\left(t\right) \tag{3.1}$$

where w_i is the weight of the synapse connecting Z to its i^{th} presynaptic neuron y_i, $y_i(t)$is 1 if y_i issues a spike at time t, and w_0 models the intrinsic excitability of the neuron Z. The stochastic firing model for Z, in which the firing probability depends exponentially on the membrane potential, is expressed as

$$prob\left(Z\ fires\ at\ time\ t\right) \propto exp(u\left(t\right)) \tag{3.2}$$

In Eq. 3.2, small variations of $u\left(t\right)$ resulting from the synaptic weight changes will have an exponential impact on the firing probability, which is not desirable. A range mapping function is proposed as detailed in [68] to mitigate this effect. However, this introduces additional complexities.

Since the probability of the Bayesian neuron's output firing rate has an exponential dependence on the membrane potential, the computation must be limited to a small region of the exponential curve. This keeps the neuron computation within its dynamic range, else it will saturate quickly or the firing probability builds up extremely slowly. This small region of the exponential curve can be safely approximated with a linear equation. However, the accumulation of weighted spikes approximates a linear function therefore a good dynamic range can be achieved.

For a Bayesian neuron, output firing pattern resembles a Poisson process. To model this, we randomly vary the threshold after every spike generation. By limiting

the range of threshold change to a small interval which satisfies the exponential distribution with unit rate, we achieve a firing pattern which is similar to Poisson spiking behavior as described in the model. With these simplifications and approximations, we can replace the Bayesian neuron model with an Integrate and Fire neuron with a stochastic threshold model. The general behavior of the neuron is still similar to the Bayesian neuron model even with these simplifications. Now the membrane potential $u(t)$ of neuron Z is computed as

$$u(t) = w_0 + u(t-1) + \sum_{i=1}^{n} w_i \cdot y_i(t) \tag{3.3}$$

The neuron Z spikes when the membrane potential crosses the threshold and is set to 0 (reset potential).

3.2.2 Stable STDP Rule

STDP forms the basis for learning in our synapse model. We use a multiplicative STDP rule where the amount of weight increase scales inversely with present weight size. Thus, learning is inherently stable and robust producing a unimodal weight distribution. However, it lacks synaptic competition which is an attractive feature which enables learning discriminative features in the input. In sharp contrast, in additive STDP the weight change is independent of the current weight, producing a bimodal distribution of weights and strong competition. However, without any hard weight constraints, the learning is fragile and unstable.

We model our multiplicative STDP rule such that the weight change of a synapse has an exponential dependence on its current weight as shown in Figure 3.2a. Thus, in the text we refer to this rule as the Exp rule. Update for the weight w_i of i^{th} synapse of the neuron Z from Eq. 3.3 is calculated as below

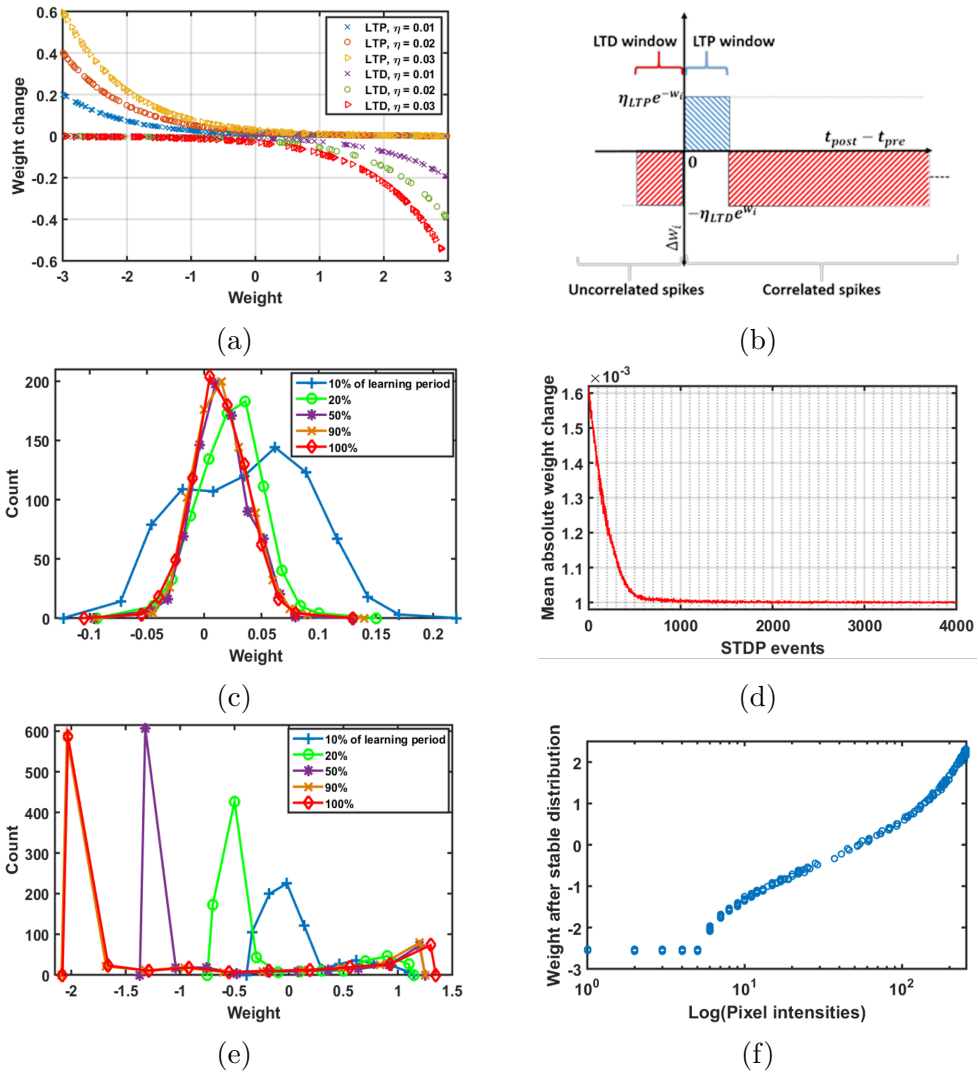


Figure 3.2: (a) Current weight vs Weight change for learning rates (b) STDP windows (c) Stable unimodal distribution over period of learning (20, 60 and 100 % of final distribution) (d) Convergence of mean absolute weight change (e) Bimodal distribution over period of learning (f) Learnt weights for pixel intensities

$$\text{If } t_{\text{post}} - t_{\text{pre}} < \tau_{\text{LTP}} \text{then, } \Delta w_i = \eta_{LTP} e^{-w_i}, w_i = w_i + \Delta w_i \tag{3.4}$$

$$\text{If } t_{\text{post}} - t_{\text{pre}} > \tau_{\text{LTP}} \text{or } t_{\text{pre}} - t_{\text{post}} < \tau_{LTD} \text{ then, } \Delta w_i = \eta_{LTD} e^{w_i}, w_i = w_i - \Delta w_i \tag{3.5}$$

where t_{pre} and t_{post} are the time steps at which the pre and post-synaptic neuron spikes, are the LTP and LTD window and η_{LTP} and η_{LTD} are the LTP and LTD learning rates respectively. The intrinsic excitability w_0 of neuron Z from Eq. 3.3 is potentiated when neuron Z fires and depressed when it does not fire with the same exponential weight dependency as for the synaptic weights. These updates are linearly added to their current counterpart.

Plasticity is implemented with LTP and LTD windows as shown in Figure 3.2b: when a postsynaptic spike occurs after a presynaptic spike and is within the LTP window, the synapse is potentiated according to Eq. 3.4. We assume all the STDP events to be independent such that only the first postsynaptic spike causes potentiation on that synapse even when the subsequent postsynaptic spikes are within the LTP window. If the postsynaptic spike falls outside the LTP window or there is no presynaptic spike, then the synapse is depressed as per Eq. 3.5. And similarly, only the first presynaptic spike within the LTD window after a given spike depresses the synapse; subsequent presynaptic spikes do not depress the synapse further before another postsynaptic spike occurs. Whereas a presynaptic spike outside the LTD window neither potentiate nor depress the synapse.

Ubiquity and fidelity of STDP as a general learning rule [77, 78] has regularly been in question despite it being biologically realistic. Phenomenological STDP rules that have simple biological precedence such as rate-based and spike-timing based models, are inherently unstable [74]. For proper utilization, they demand complementary mechanisms. Biologically-inspired mechanisms and ad-hoc mechanisms such as weight constraints, weight normalization [79], firing rate normalization, and homeostasis [52]

are usually used. These add computational complexity when simplicity is necessary, especially for an efficient and large-scale implementation.

The above-mentioned complementary mechanisms are not required for our STDP rule presented in Eq. 3.4 and 3.5 as it is inherent stable. The stability of a STDP rule can be shown with three main properties [80, 81]: (a) shape of the weight distribution is stable over time such that even if the synaptic weights change, the distributions follow similar patterns, (b) unimodal distribution such that all the weights are not concentrated at the boundaries and (c) limited weights without hard weight constraints such that no synaptic weights explode. We check our STDP rule for these properties through simulations to verify its stability empirically.

Figure 3.2c shows the distribution of synaptic weight during different learning stages when the input to the network with one SIF neuron and 28x28 input neurons is random and uncorrelated, and the synapse is updated through our STDP rule. Pertaining to the properties of a stable STDP rule. Figure 3.2c shows that our method resonates those properties very closely. The distribution achieved is unimodal, and its shape is stable over the period of learning. And the weights are naturally constrained between the soft limits imposed by the formulation of the rule itself and not with any complimentary mechanism. Figure 3.2a shows that the weight updates are exponentially proportional to the current weights during both LTP and LTD; if the current weight is highly positive, then it is penalized with a lower weight update for LTP case (larger weight update for LTD case). On the contrary, if the current weight is highly negative, then a larger weight update is produced for LTP case (lower weight update for LTD case). Because of this weight dependence, strong synapses experience a net depression, whereas weak synapses experience a net potentiation whose magnitudes are controlled separately through separate learning rates for LTP and LTD in our STDP rule. This net depression and potentiation confines the synaptic weights and stabilizes the weight distribution. As the mean absolute weight change converges

asymptotically, the distribution reaches equilibrium and becomes stationary. We can see that in Figure 3.2c where the change in weight distribution decreases and becomes stationary as the mean absolute weight change shown in Figure 3.2d converges and remains in that equilibrium.

Usually in weight-dependent STDP models, there is a lack of competition [74]. Competition between inputs allows for a specific set of input synapses to drive a neuron into firing. This is important for discriminative applications. In STDP models in which potentiation and depression are independent of the synaptic weight, there is strong competition [73, 82]. In these models of constrained plasticity, the potentiation mainly occurs if the input has caused the spike. When one input starts driving the postsynaptic spikes and its weight increases, the other inputs will become less correlated with the postsynaptic spikes, and these inputs will effectively be depressed. This pushes the distribution of the weights towards the applied hard constraints forming a bimodal distribution.

To replicate such competitive behavior amongst the inputs and still maintain a stable distribution, [74] utilizes Activity-dependent scaling of the synaptic weights. Activity-dependent scaling is a homeostatic mechanism in which the neuron reacts to changes in the postsynaptic activity, scales all synapses to keep the activity of the neuron within bounds. The scaling is multiplicative. If one synapse is potentiated, the postsynaptic activity rises, and the activity-dependent scaling kicks in to reduce all the synaptic weights. The shape and stability of the weight distribution are not affected by the scaling.

In our STDP model, a behavior similar to activity-dependent scaling is induced through the elongated symmetric STDP window used as shown in Figure 3.2b. LTD not only happens when the presynaptic spike falls in the LTD window but also when the postsynaptic spike falls outside LTP window or there is no presynaptic spike at all. So, when a certain synapse is potentiated due to strong correlation, synapses

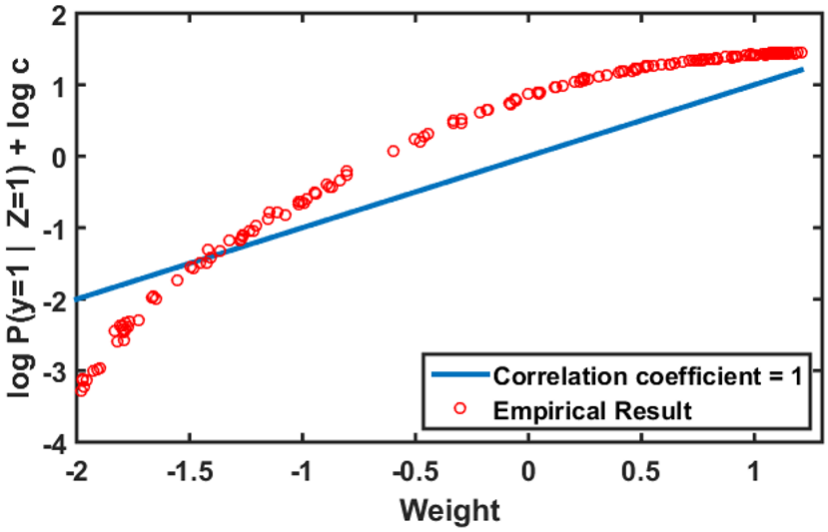


Figure 3.3: Correlation graph between synaptic weight and log conditional probability with constant c=30

with weak correlation are depressed along with synapses with no correlation. This induces good competition and learns discriminative separation even between correlated inputs. When trained on a MNIST image, this induces good competition producing a distribution similar to that of the input or a bimodal distribution with sparse strong synapses and dense weak or silent synapses [83] as seen in Figure 3.2e. And Figure 3.2f shows that higher pixel intensities induce higher weights, whereas lower pixel intensities induce lower to negative weights in the corresponding synapses driving weights to clusters in the opposite ends reiterating the sharpness of the discriminative ability.

One important feature of the Bayesian neuron model is that it allows for neural sampling [76] such that the weight of the synapse is the log conditional probability of pre-synaptic neuron firing given the post-synaptic neuron has fired with a log constant ($\log P\left(y=1 \mid Z=1\right)+$) and each spike is a sample of the posterior distribution. This allows for Bayesian inference. Using an exponential dependence of weight update on the current weight allows to retain the log conditional property as shown empirically by the correlation graph in Figure 3.3. As the theoretical results remain true [76] for any $c > 1$, we choose $c = 30$. Under this condition, the correlation coefficient of the

synaptic weight and the log conditional probability ($\log P\,(y = 1 \mid\ Z = 1) + \log 30$) is 0.9885. This also accounts for the learnt weights being positive and negative.

3.2.3 Quantized 2-Power Shift Rule

In terms of computation, the proposed STDP rule requires an exponential and a multiplication operation for both LTP and LTD for each synapse. From the perspective of efficient digital hardware implementation, these are expensive operations in terms of circuit area and computation time as explained in [68].Thus, substituting these with simplified operations is highly desirable. In the next, we introduce a Quantized 2-power shift rule (Q2PS), which approximates our STDP rule in Eq. 3.4 by removing both multiplication and exponential. The approximation is summarized in

$$\begin{aligned} &\text{If } t_{\text{post}} - t_{\text{pre}} < \tau_{\text{LTP}}, \Delta w_i = \eta_{LTP} 2^{-w_i} = 2^{\eta' LTP - w_i} \\ &\text{where } \eta'_{LTP} = \log_2 \eta_{\text{LTP}} \end{aligned} \tag{3.6}$$

$$\begin{aligned} &\text{If } t_{\text{post}} - t_{\text{pre}} > \tau_{LTP} \text{ or } t_{\text{pre}} - t_{\text{post}} < \tau_{LTD}, \Delta w_i = \eta_{LTD} 2^{w_i} = 2^{\eta'_L L \bar{D} + W_i} \\ &\text{where } \eta'_{LTD} = \log_2 \eta_{LTD} \end{aligned} \tag{3.7}$$

We denote $Q = \ \eta'_{LTP} - w_i$ for LTP and $Q = \ \eta'_{LTD} + w_i$ for LTD, also let $\bar{Q}$ be the quantization of Q through priority encoding. This encoding converts the binary representation of Q into a new binary representation with the index of the most significant active input bit as the highest priority. For example, if $Q = 12$ then its binary representation is 1100. The priority encoding of this is 1000, hence $\bar{Q} = 8$. After the quantization, the change of the synaptic weight is calculated by shifting the value 1 by $\bar{Q}$, either left or right, based on the sign of that result. In other words, for both cases

$$\Delta w_i = \begin{cases} 1 \ll \left|\bar{Q}\right| \ , & if\ \bar{Q} > 0 \\ 1 \gg \left|\bar{Q}\right| \ , & if\ \bar{Q} < 0 \end{cases} \tag{3.8}$$

where $\ll$ and $\gg$ represent binary shift left and shift right operations, respectively. This approximation allows implementation of the STDP rule presented in Eq. 3.4 and 3.5 on digital hardware by using a priority encoder, negligibly small lookup to determine $\left|\bar{Q}\right|$ from the encoded value, barrel shifter and an adder circuit. Please note that, based on Eq. 3.6 and 3.7, Δw_i should be calculated as 2^Q, which can be obtained by shifting the value 1 by Q. We refer to this as 2P approximation. However, because the value of $\bar{Q}$ has much coarser resolution than Q, the implementation of Q2PS approximation is much simpler than the 2P approximation. We refer to the original STDP rule in Eq. 3.4 and 3.5 as Exp rule. Figure 3.4 compares the Δw_i calculated using the *Exp*, *2P* and *Q2PS* rules, with a learning rate of 0.08 for all the cases. As we can see, the Q2PS rule provides multilevel quantization, which enables similar quality of trained weights even with approximations when compared to Exp rule. This way, we can perform online learning in a quick and efficient manner from the hardware perspective with a smaller footprint in terms of circuit area and energy consumption.

3.3 Experiments

We build a spiking neural network to classify MNIST [84] handwritten digits and perform the analysis of our proposed STDP learning rules. Both supervised learning and unsupervised learning for training the entire network including the classifier layer are discussed. We use a SNN simulator SpNSim [68] to simulate all the experiments presented in the paper. SpNSim is a multithreaded and scalable simulation platform built using C++. It is flexible and vectorized for efficient evaluation and capable of

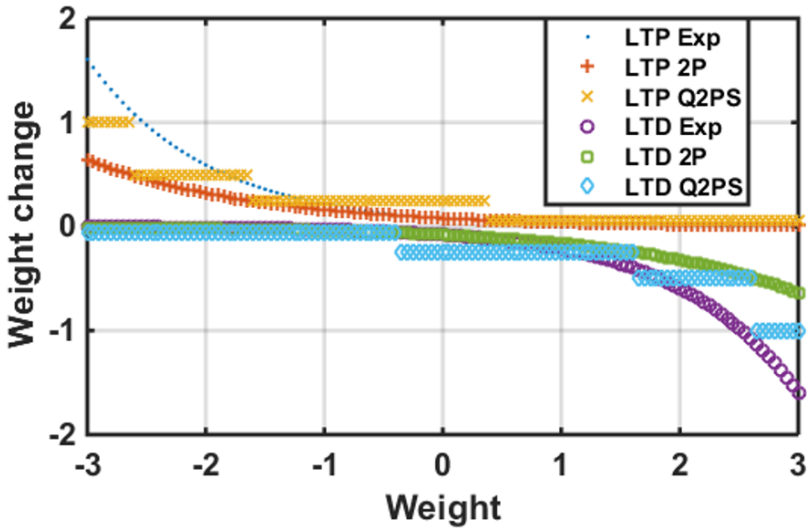


Figure 3.4: Comparison of the introduced STDP rules

handling large-scale networks of mixed neuron models.

3.3.1 Network Architecture

We create a 3-layer network as shown in Figure 3.5a. The input layer contains 28x28 neurons (one per pixel), the second layer has variable number of stochastic integrate and fire (SIF) neurons for different trials (the hidden layer), and the third layer is the classification layer with 10 SIF neurons (one per class). The input is fed to the input layer which encodes the pixel intensities with 0 Hz – 300 Hz firing rates and the classification result is obtained from the classification layer. Input layer is fully connected to the hidden layer and hidden layer to the classification layer, and all these synapses are plastic.

Along with the SIF neurons, hidden and classification layer neurons there are supporting ReLU neurons [68] for lateral inhibition. At both layers, the ReLU neurons (inhibitory) form WTA (Winner Take All) circuits with a connectivity as mentioned in [68]; one-to-one connection from SIF neurons to the ReLU neurons and the ReLU neurons are connected to all the SIF neurons except for the one from which it receives a connection. Hard or soft WTA behavior can be achieved based on the degree of

inhibition delivered. Hard WTA happens when the inhibition is strong such that it brings down the firing rate of the non-preferred SIF neurons to zero, resulting in only one neuron with the highest excitation being active. On the other hand, if a plural voting action is required within the set, the degree of inhibition is tuned to be moderate. This makes SIF neurons fire with different stable rates, which is a soft WTA behavior where firing rate is proportional to their relative excitation levels.

The hidden layer learns features of the MNIST images and more than one neurons could learn a feature concerning a certain class, thus requiring multiple neurons in the layer to be firing. Thus, the hidden layer is set to a soft WTA inhibition level. Whereas in the classification layer, a neuron is associated with a class in a one-to-one basis, thus requiring only one neuron to fire in a period. Hence, the classification layer is set to a hard WTA inhibition level. And these synapses are not plastic.

The stochasticity of the SIF neuron is important during learning to allow the neurons in both layers learn unique features. However, having similar stochasticity during recall becomes counterproductive as inappropriate inputs could excite a certain feature in the hidden layer, and similarly an incorrect feature could excite an incorrect neuron in the classification layer producing an incorrect classification. Thus, during the recall phase, we disable learning and fix each neuron's spiking threshold to make them deterministic.

3.3.2 Learning

Using our STDP rule, we perform both unsupervised and supervised learning. STDP is known to have the effect of concentrating high synaptic weights on afferents that systematically fire early. It makes neurons naturally selective to patterns that are reliably present in the input. These patterns are learnt without supervision and can be used for categorization and discrimination. In this way, the hidden layer learns patterns from the input layer as shown in Figure 3.5a.

For classification purposes, supervision is necessary to label the neurons that have learnt discriminative capabilities, thus categorization is in order during recall. In [52], the feature learning neurons are manually labeled after training and their collective firing rates are used to classify digits. This methodology creates issues in cases when the learnt feature is not of a distinguishable class or it is common to several classes, e.g., slanted "1" is a sub feature of "7", "3" has feature elements common to "5" and "8" and so on. Thus, labeling these features to one class or the other could lead to mislabeling of undistinguished features or underutilization of a common feature. To avoid such situations, we add a classification layer which learns intermediate features from all the available learnt features in the hidden layer without having to manually label them. The classification layer has one neuron per class to perform categorization such that the neuron with the highest firing rate is the predicted class. And to train the classification layer, we introduce a supervised learning approach using our STDP rule on top of the already trained hidden layer.

During supervised training in ANN, the neurons in the classification layer are driven by the class labels and the error is propagated down the network layers. Here we add a neuron per class which fires at a specific rate based on the class being observed. We call this Teacher neuron. These neurons are connected one-to-one to the classification layer as shown Figure 3.5a. They excite one neuron per class to fire at a specific rate when the label is presented. The teacher neurons to classification layer neuron synapses are not learnt. However, it excites the connected classification neuron and creates differentiable time dependent relations between the classification layer and the hidden layer. Based on our STDP rule, the active input synapses from the classification layer to the hidden layer are potentiated and the inactive ones are depressed, thus propagating the error down through the incoming synapses

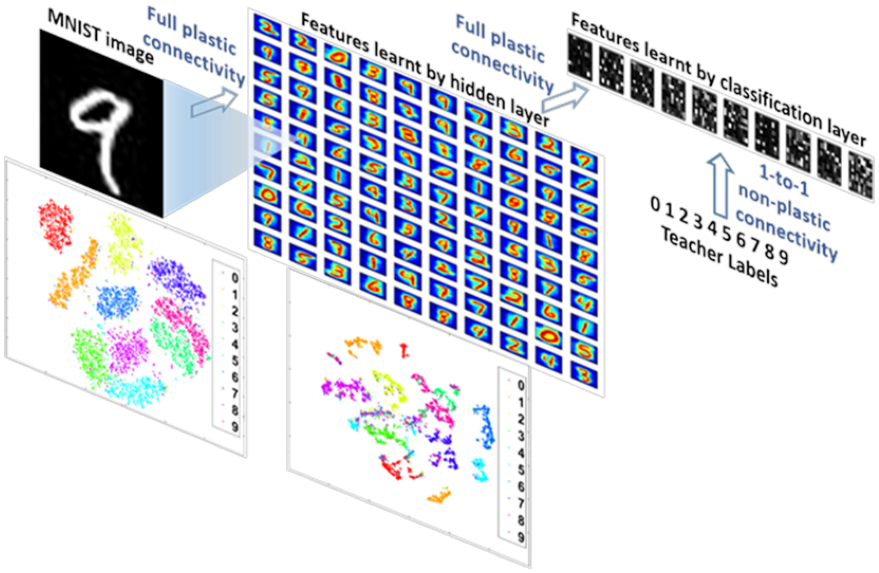
3.4 Results and Discussions

In this section, we present the results on the handwritten digits classification. We also analyze the performance for different sets of parameters to present the rationale on how to tune the network, and what could be adapted in the case of noise and variable intensities in the input images. Finally, we discuss the quality of the features learnt.

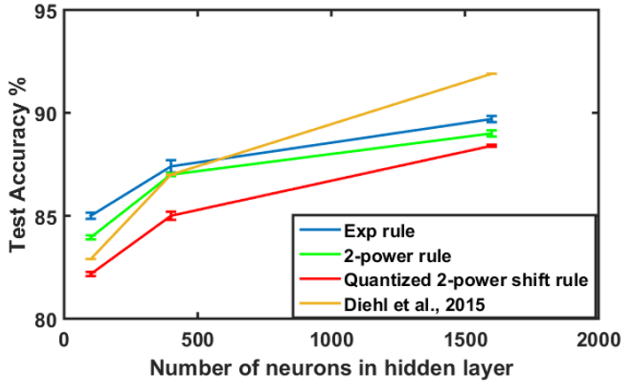
3.4.1 Handwritten Digits Classification

We trained a 3-layer network as presented in this section with 10 classification neurons and with 3 variations of hidden layers consisting of 100, 400, and 1600 neurons. The training is done layer-wise; the inputs to hidden layer synapses are trained without supervision by presenting the complete training set once, and then the hidden to classification layer synapses are trained with supervision on top of the trained hidden layer by presenting the complete training set again. Each input neuron is connected to one pixel in the image and fires at a rate (maximum rate is 300Hz) that is proportional to the pixel intensity. No preprocessing is done on the images and we present each training image for 200 time steps during training. The learnt features and hidden to classification layer synaptic weights are shown visualized in Figure 3.5a. The same parameters are used throughout all these experiments.

Figure 3.5a also shows the t-SNE visualization for the input layer and the hidden layer which provides a qualitative analysis. This technique performs t-distributed stochastic neighbor embedding, which maps high-dimensional data that lie on several different, but related, low-dimensional manifolds to lower dimensions by capturing the local structure of the high-dimensional data. In our case 784 dimensions of the input layer and 100 dimensions of the feature layer are mapped to 2 dimensions, respectively. These visualizations are made using the firing rates of all the neurons in



(a)



(b)

Figure 3.5: (a) Network architecture showing the connectivity, input, learnt features by hidden and classification layer, the labels and t-SNE visualizations (b) Test accuracies for different sized networks

that layer. It is clear from the figure that the proposed STDP rule can form a tight clustering of the input space when mapping it into the feature space. This generates features that can be more easily classified by the classification layer.

The three networks; 100, 400, and 1600 hidden neurons achieved an average classification accuracy of 85, 87.4, and 89.7% for our STDP rule, respectively, as shown in Figure 3.5a. We achieve better classification accuracy for networks with 100 and 400 hidden neurons compared to [52] for their respective network sizes, whereas for 1600 hidden neurons our results are slightly lower. This lower accuracy for a high

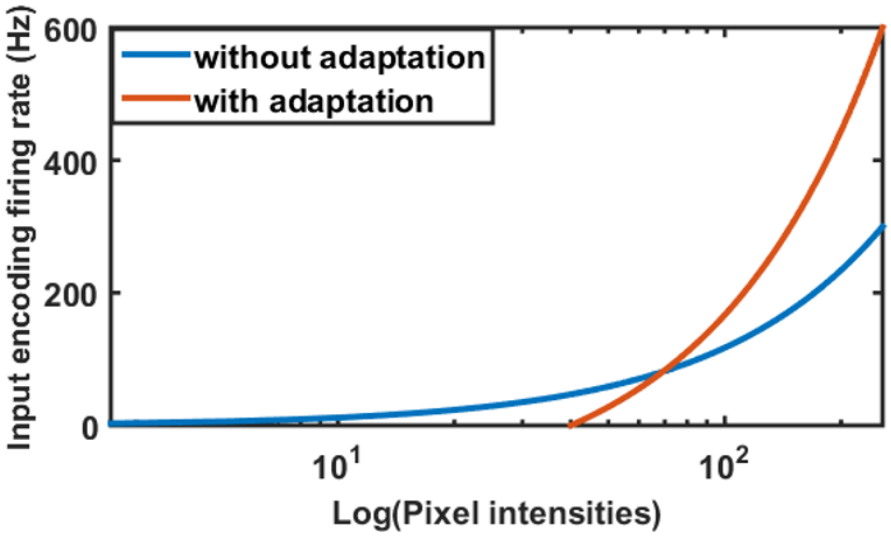


Figure 3.6: Firing rate curve with and without adaptation for different pixel intensities

number of hidden neurons can be attributed to overfitting due to the large number of trainable parameters per neuron for the classification layer. [52] don't face this issue as they manually labelled the features for classification instead of a classification layer trained on top of those learnt features. We could resolve this issue in the future by using dropout while training. The accuracies of 2P and Q2PS STDP rules for these networks are also shown in Figure 3.5b. As can be seen from the results, even with approximations and quantization, the Q2PS rule achieves accuracies reasonably close to the Exp STDP rule.

3.4.1.1 Classifying Images with White Noise

Background clutter is a common form of noise which reduces the effective contrast in a real-world image inputs. This reduces the discriminative properties between images. To test the resilience of our network at different levels of background clutter, we create artificial MNIST test sets with Additive White Gaussian Noise (AWGN) of different SNRs (15, 20 and 25). The results are shown in Table 3.1.

With the current temporal sparseness (maximum rate 300 Hz) for encoding the input, the AWGN noise has a significant impact on the accuracy. In biology, the

Table 3.1: Performance with and without adaption in presence of AWGN noise

AWGN noise SNR	**Without adaptation**	**With adaptation**
15	19.1%	34.5%
20	40.9%	83.9%
25	78.6%	85%

Table 3.2: Performance with and without NWTA for different intensity MNIST images

Intensity level	**Without NWTA**	**With NWTA**
25%	62.9%	79.7%
50%	81.5%	84.1%
100%	85%	84.9%
Mixed	82.7%	84.4%

LGN neuron in the early visual pathway employs an adaptive strategy [85] which decreases the temporal sparseness (increasing the firing rate) and shifts stimulus–response curves toward higher stimulus intensities when the effective contrast is reduced due to white noise. This reduces the spike-timing jitters induced by lower effective contrast by increasing the level of discrimination (in terms of firing rate) between similar stimuli and ignoring low intensity stimuli in the background. We apply a similar adaptation strategy to deal with the AWGN noise. As shown in Figure 3.6, we shift the intensity-firing rate (stimulus-response) curve towards higher intensities and increase the range of the firing rate to 600 Hz. This leads to a dramatic improvement in accuracy in the presence of AWGN noise. But this comes with the decrease in sparsity and reduction in sensitivity to lower intensity discriminative properties.

3.4.1.2 Classifying Images with Variable Intensity

Resistance to variation in the image is important. The most common variation is the intensity of the image. In real life, lighting conditions result in different intensity images. Thus, we create artificial MNIST test sets with 25% intensity, 50% intensity

and mixed intensity to test our network on. The results are shown in Table 3.2.

This leads into the discussion of when normalization becomes useful in our network. Normalization mechanisms allow for intrinsic modulation of the firing rates of the neurons in a layer; increase if the rate is low, and decrease if the rate is high. Thus, keeping all the neurons approximately within a range of firing rates. With low intensity images, there can be a propensity of a lack of excitation for the neurons in the hidden layer to fire and thus for the classification layer. This negatively impacts the classification layer's ability to accurately categorize the image's class.

Normalization mechanisms such as homeostasis [52] and weight normalization or scaling [4, 86] add complexities onto the neuron model and the learning rule. With our aim of simplicity, we utilize the normalized winner-take-all (NWTA) mechanism as described in [68]. It adds three additional neurons in the hidden layer; upper limiter (UL), lower limiter (LL), and exciter (Ex), which checks if the upper limit of the desired firing rate range is reached and increases lateral inhibition, checks if the *lower limit o*f *th*e d*esired firing* r*at*e is re*ached a*nd *p*rovides additional excitation, respectively. We incorporate NWTA in the hidden layer as low intensity image directly impacts its firing rates. The results with and without NWTA mechanism is shown in Table 3.2. for a network with 100 neurons in the hidden layer averaged over three experiments.

From Table 3.2, we can see that normalization results in improvement for lower intensity images, whereas for normal intensity images normalization is redundant as the network was trained using normal intensity images.

3.5 Conclusion

In this work, we presented a simplified neuron model and introduced a stable and competition inducing STDP learning rule. With simplicity in terms of computation

and hardware implementation being a motivating factor, we also introduced Q2PS rule which can be implemented with minimal digital hardware resources. We also combined unsupervised and supervised learning with STDP using a layer-wise training approach, applied the network for handwritten digit classification with competitive results, and provided analysis for parameter tuning.

Chapter 4

Approximating Back-propagation for a Biologically Plausible Local Learning Rule in Spiking Neural Networks

4.1 Introduction

In contrast to SNNs, artificial neural networks (ANNs) are comprised of neurons with continuous non-linear activation functions and communicate through high-precision values. These neurons are differentiable enabling gradient-based optimization methods and they can be stacked in multiple (deep) layers producing a locally robust universal function approximator. Recently, with the availability of computing power and large labeled datasets, ANNs have become very deep, producing breakthrough performances in various fields of pattern recognition. Despite their effectiveness, they are computationally expensive and not suitable for implementations on portable hardware.

Even though ANNs were originally inspired by the brain and the representations learned by ANNs in tasks like image classification is shown to be similar to those observed in the receptive fields of the visual cortex [87], learning the synaptic weights through backpropagation appears biologically unrealistic [88] in various ways: (a) Neuron models in ANNs send a continuous-valued output which is biologically unrealistic. Whereas, it is not trivial to compute their derivatives. (b) Neuron activities are not differentiable, while gradient descent requires local derivatives of the neuron to backpropagate the error. (c) The connection between neurons in SNN is unidirectional. A backward path must be added specifically. However, it then will create the "weight transport" problem. Two copies of the weight must be stored in both the forward path and backward path. Since the weight is constantly changing during learning, maintaining the coherence between the two copies will create significant overhead. (d) Errors in ANN are propagated as real values. This representation is not compatible with SNN.

In this chapter, we systematically solve the above biological implausibility issues with backpropagation with an intent to approximate the entire backpropagation algorithm utilizing only the spiking neurons and adapt it towards a local learning rule which resembles STDP. We validate the approximation and the local learning rule using three benchmarks. The following summarizes our contributions:

1. We formulated the backpropagation algorithm in terms of spiking neurons to approximate the non-linearity of the forward pass and the linearity of the backward pass.

2. We show that, through a proper connection between the forward and backward paths, gradient descent can be achieved using local weight update that resembles STDP. This connection is derived by adapting the back-propagated errors as the target difference in the difference target propagation algorithm.

3. We propose to use a neural circuit to calculate the derivative of the $L2$ loss in terms of spikes.

4.2 Related Works

There have been various approaches to adopt the backpropagation algorithm to train deep SNNs directly. One category of approaches keeps track of the membrane potential at spike times and back-propagate errors based on that. SpikeProp [89] is the first attempt to train an SNN using such an approach. However, SpikeProp is very limited to single-spike learning. A similar category of approach [90] [33] treats the discontinuities during spike times as noise and smoothens the membrane potential to essentially make it continuous. These approaches utilize the spike rate to compute the loss and membrane potential to compute the error derivative, and hence create a discrepancy. [34] proposed an event-driven random backpropagation (eRBP) algorithm simplifying the backpropagation chain path. However, this work requires multicompartmental neurons to enable errors to locally modulate plasticity. In [35], a supervised learning method was proposed (BP-STDP) where the backpropagation update rules were converted to temporally local STDP rules for multilayer SNNs.

In summary, existing works have some major limitations: a) they either require high-precision back-propagating error derivatives or do not adhere to the linearity of the original backpropagation algorithm. b) Each neuron must know the membrane potential of its presynaptic neighbors in order to determine the synaptic weight change.c) Each neuron must also know the error derivatives of its postsynaptic neighbors in order to calculate its own error derivative. All these limitations violate the constraints of the local communication rule in the spike domain. Meanwhile, there are approaches [91][92] to convert ANNs to SNNs, which is not condusive to neuromorphic implementation and are thus out-of-scope for this work.

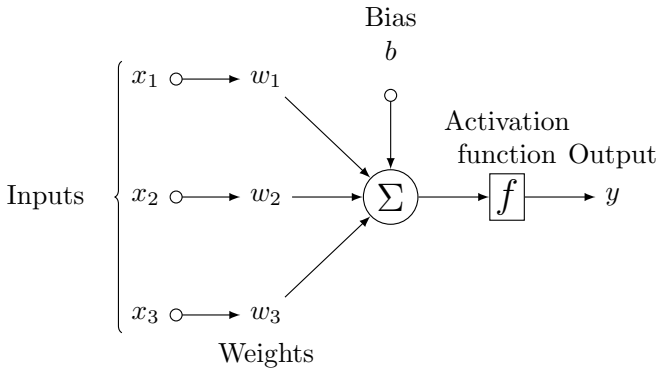


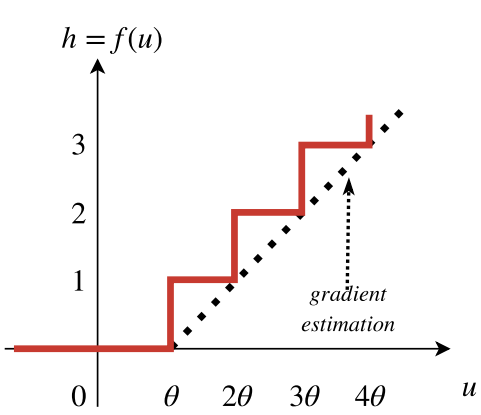
Figure 4.1: Spiking Neuron Model

4.3 Methods

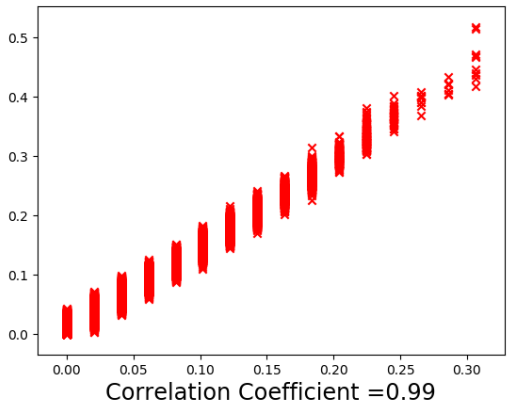
In this section, we analyze the root cause of biological implausibilities in the back-propagation algorithm and later we will show that they can be eliminated for SNNs through a suitable sequence of approximation techniques. This includes the approximation of the derivatives of spiking neurons (Section 4.3.1), representing linearly propagated errors using spiking neurons (Section 4.3.2), local learning through error spikes (section 4.3.3) and a completely spike-based computation of $L2$ loss's derivative (Section 4.3.4). The whole method is put together in the algorithm in Section 4.3.6. Table 4.1 provides a list of symbols used relating to the neuron's state and output for clarity.

Symbol Forward pass	**Symbol Backward pass**	**Meaning**
U	E	Membrane potential
u	ϵ	Accumulated sub-threshold membrane potential
h	e	Spike count
s	es	Spike train

Table 4.1: Symbols



(a) Spiking Activation Function



(b) Correlation between ANN (y-axis) and SNN (x-axis)

Figure 4.2: Approximation of the forward pass

4.3.1 Derivatives of Spiking Neuron Function

Figure 4.1 shows a general model of a simple neuron in a neural network in which there are a set of inputs (x_1, x_2, x_3), synapses from those inputs to the neuron (w_1, w_2, w_3), those inputs are accumulated and then goes through an activation function to produce an output y. In SNNs, the inputs and outputs are spike trains, and the activation function is a discontinuous threshold function. The Leaky Integrate-and-Fire (LIF) is the most popular neuron model [35] because of its simplistic representation of a spiking neuron. Here, we utilize a discrete-time variant of the non-leaky IF neuron that accumulates input spikes over time and produces a binary spike when the membrane potential crosses a clear threshold. The membrane potential of neurons in layer i at time t is:

$$U_i(t) = \sum w_{i-1,i}.s_{i-1}(t) + b_i + U_i(t-1) \tag{4.1}$$

$$\text{if} \quad U_i(t) \geq \theta \quad \text{then} \quad s_i(t) = 1, \quad \text{else} \quad s_i(t) = 0$$

where $w_{i-1,i}$ is the synaptic weight between pre-synaptic neurons in layer $i-1$

and post-synaptic neurons in layer i, b_i is the bias. $s_{i-1}(t)$ and $s_i(t)$ are pre and post-synaptic spike trains, respectively. The neuron spikes when the membrane potential reaches the threshold θ and it's membrane potential is reset to the resting potential of 0.

The spike in the spike train is represented as a delta Dirac function at the spike time t_s

$$s(t_s) = \delta(t - t_s)$$

such that the sub-threshold membrane potential of the postsynaptic neuron between consecutive spikes $s_i(t_{s1})$ and $s_i(t_{s2})$ is given by

$$U_i(t) = \sum w_{i-1,i}(\sum_{t=t_{s1}}^{t_{s2}} \delta(t - t_s)) + b_i(t_{s2} - t_{s1}) \tag{4.2}$$

The activity of a neuron can be represented either in terms of their rate or spike count. Here, as we intend to compute the loss based on the spike counts, we utilize the spike counts as the neuron's activity which is given by the sum of delta Dirac functions over the spiking time interval T as shown in Eq. 4.3.

$$h(T) = \sum_{t=0}^{T} s(t_s) = \sum_{t=0}^{T} \delta(t - t_s) \tag{4.3}$$

where t_s are the post-synaptic spike times of neuron j.

The spike counts can also be represented in terms of the neuron 's membrane potential.

$$h(T) = f(u(T)) = \left\lfloor \frac{u(T)}{\theta} \right\rfloor \tag{4.4}$$

where θ is the spiking threshold and $u(T)$ is the accumulated sub-threshold membrane potential computed in Eq. 4.2 over the time period T

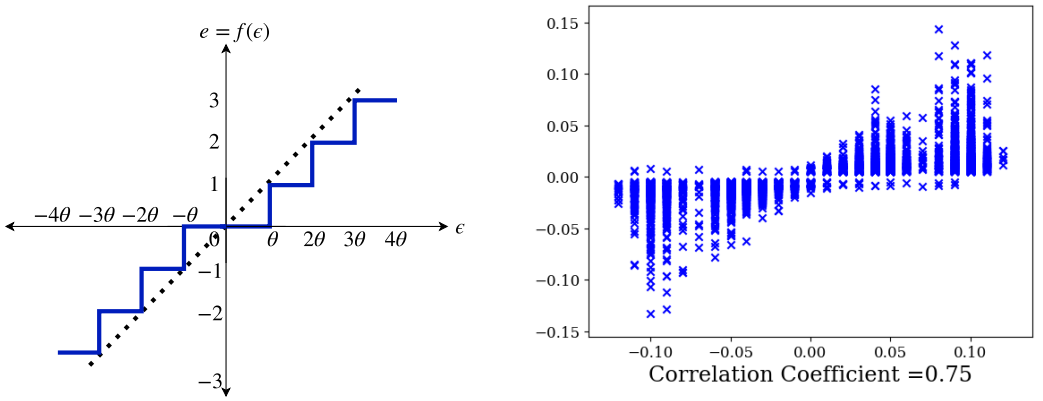
$$u(T) = \sum_{s=0}^{N} U(t_s)$$

where t_s is the post-synaptic spike time, N is the number of spikes in the post-synaptic spike train such that $T > t_N$

The Figure 4.2(a) shows the function $f(u)$ for the spike count over time T is a staircase function of the total accumulated sub-threshold membrane potential u. This represents the activation function of the spiking neurons over that time window. Figure 4.2(b) shows a strong correlation of the hidden layers in the forward pass between the ANN and the SNN with the same weight initialization and input sample in the MNIST experiments. The ladder-like effect seen in Figure 4.2(b) is due to the discretization caused by the limited time window.

The function is clearly discontinuous and has no derivatives when $u = n\theta$ where $n = 1, 2, .., T$. The derivative is zero at all other u. To estimate the function's gradient, we approximate $f(u)$ by a piece-wise linear function as shown by the dotted line in Figure 4.2(a). The function is zero for $u < \theta$, then increases linearly with a slope $\frac{1}{\theta}$. This approximation resembles a ReLU function shifted by θ. This shift by θ prevents computation of the gradient based on the membrane potential before the first spike. This is important as we compute the loss in terms of spikes and require the gradient on the basis of the spikes. Thus, the derivative of $f(u)$ is approximated as

$$h' = f'(u) = \begin{cases} \frac{1}{\theta} & \text{if } u \geq \theta \\ 0 & \text{if } u < \theta \end{cases} \tag{4.5}$$



(a) Spiking Activation Function

(b) Correlation between ANN (y-axis) and SNN (x-axis)

Figure 4.3: Approximation of the backward pass

4.3.2 Spike Representation of Error Derivatives

Let us assume a network with l layers. For a hidden layer i which is post-synaptic for layer $i-1$ and pre-synaptic for layer $i+1$ in a time period T,

$$u_i = w_{i-1,i} \cdot h_{i-1} + b_i \tag{4.6}$$

and the spike count from Eq. 4.4 is

$$h_i = \left\lfloor \frac{u_i}{\theta_i} \right\rfloor$$

Now, let us assume a loss function L such that the derivative of the loss function at the $i+1$ layer is

$$\epsilon_{i+1} = \frac{\partial L}{\partial h_{i+1}}$$

Now, using the chain rule, the error derivative at layer i is:

$$\epsilon_i = \frac{\partial L}{\partial h_i} = \frac{\partial L}{\partial h_{i+1}} \cdot \frac{\partial h_{i+1}}{\partial u_{i+1}} \cdot \frac{\partial u_{i+1}}{\partial h_i}$$

$$\epsilon_i = \epsilon_{i+1} \cdot \frac{1}{\theta_{i+1}} \cdot w_{i,i+1}^T \quad \text{when } u_{i+1} > \theta_{i+1} \tag{4.7}$$

We intend to represent the error derivative using spiking neurons such that ϵ now represents the accumulated sub-threshold membrane potential for spiking neurons without bias. Such that we utilize the same activation function as for the feedforward path given in Eq. 4.4 to get a formulation as shown in Eq. 4.6

$$\epsilon_i = e_{i+1} \cdot w_{i,i+1}^T \tag{4.8}$$

where e_3 is the error spike count such that we can model the error network with IF neurons as well. In other words, ϵ_i is the accumulated sub-threshold membrane potential of the error neuron whose input is the spikes e_{i+1} from the upper-level error neuron.

So, the membrane potential of error neurons in in layer i at time t is:

$$E_i(t) = \sum w_{i,i+1}^T . es_{i+1}(t) + E_i(t-1)$$

$$\text{if} \quad E_i(t) \geq \theta \quad \text{then} \quad es_i(t) = 1, \quad \text{else} \quad es_i(t) = 0$$

where $w_{i,i+1}$ is the synaptic weight matrix from layer i to layer $i+1$ and $es_{i+1}(t)$ is the spike train in the feedback path from layer $i+1$. The neuron spikes when the membrane potential reaches the threshold θ and it's membrane potential is reset to the resting potential of 0.

The activation function in Eq. 4.4 approximates a ReLU function which is non-linear. To approximate the linearity of the backward pass of backpropagation, we adopt a two-channel spikes approach; one for positive and another for a negative spike as mentioned in [30] and the spike count for the error derivatives is given as

$$e = \begin{cases} e^+ = \lfloor \frac{\epsilon}{+\theta} \rfloor & \text{if } \epsilon > 0 \\ e^- = \lfloor \frac{\epsilon}{-\theta} \rfloor & \text{if } \epsilon < 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

Eq. 4.9 essentially performs a linear combination of positive and negative versions of the non-linear function Eq. 4.4 to approximate a linear function as shown in Figure 4.3(a). Figure 4.3(b) shows a fairly strong correlation of the hidden layers in the backward pass between the ANN and the SNN with the same weight initialization and input sample in the MNIST experiments.

4.3.3 Local Learning

Given Eq. 4.5 and 5.3 and the chain rule, we can derive the weight update for incoming weights in layer i as

$$\Delta w_{i-1,i} = \epsilon_i \cdot h_i^{'} \cdot h_i \tag{4.10}$$

In terms of the synapse, this weight update is based on the local variables presynaptic activity and derivative of the postsynaptic activity. However, the error derivative is not local to the synapse. So, we take inspiration from the concept of an alternate credit assignment method called difference target propagation [93] and modify it to derive a local weight update formulation.

The main idea of target propagation is to provide each feedforward unit's activation a target value which is close to the activation value. Once the target is computed, the gradient of the loss between the feedforward value and the target value is propagated only locally. In the limit where the target is very close to the feedforward value, target propagation should behave like backpropagation.

Let us assume a network with l layers. Let $\hat{h_i}$ be the target spike count for the

hidden layer i, such that the local $L2$ loss is given as

$$L(\hat{h_i}, h_i) = \left\| \hat{h_i} - h_i \right\|_2^2$$

such that the local weight update using the chain rule is given as

$$\Delta w_{i-1,i} = \frac{\partial L(\hat{h_i}, h_i)}{\partial w_{i-1,i}} = \frac{\partial L(\hat{h_i}, h_i)}{\partial h_i} \cdot \frac{\partial h_i}{\partial u_i} \cdot \frac{\partial u_i}{\partial w_{i-1,i}}$$

$$\Delta w_{i-1,i} \propto (\hat{h_i} - h_i) \cdot \frac{1}{\theta_i} \cdot h_{i-1} \quad \text{when } u_i > \theta_i \tag{4.11}$$

where the threshold θ is a constant that is incorporated in the learning rate η such that the local weight update is given as

$$\Delta w_{i-1,i} = \eta \cdot (\hat{h_i} - h_i) \cdot h_{i-1} \tag{4.12}$$

In [93], the targets are determined utilizing an autoencoder with inverse synaptic weights which are learnt through reconstruction. Here, we use the error derivatives as the difference to determine the target activation value.

$$\hat{h_i} = f(\hat{u_i}) = f(u_i + e_i \cdot w_{ei})$$

From Eq. 4.4,

$$\hat{h_i} = \left\lfloor \frac{u_i + e_i \cdot w_{ei}}{\theta_i} \right\rfloor$$

Now, if $w_{ei} = \gamma \theta_i$ then

$$\hat{h_i} = h_i + \gamma e_i \tag{4.13}$$

Eq. 4.13 essentially provides the connectivity from the error neurons to the feed-forward neurons as shown in Figure 4.4(b) which is a one-to-one connection with

non-plastic synaptic weight equal to a constant γ factor of the spiking threshold, to produce the target spike count.

From Eq. 4.13 we can also conclude

$$\hat{h_i} - h_i \propto e_i \tag{4.14}$$

Interestingly, given Eq. 4.14, the local weight update in Eq. 4.12 relates closely to the formulation of the rate-based STDP rule in [94] in which the synaptic weight update is in proportion to the product of the presynaptic activity and the rate of change of the postsynaptic activity. This conclusion is supported by Figure 4.6 which shows the strong correlation between our local weight update and a basic two-factor event-based STDP rule that fits the biological findings in [7]. Thus, we call the local learning rule given by Eq. 4.12 Error-modulated STDP (EMSTDP).

4.3.4 Derivative of $L2$ Loss

We utilize the $L2$ loss for the output layer in a network with l layers given as

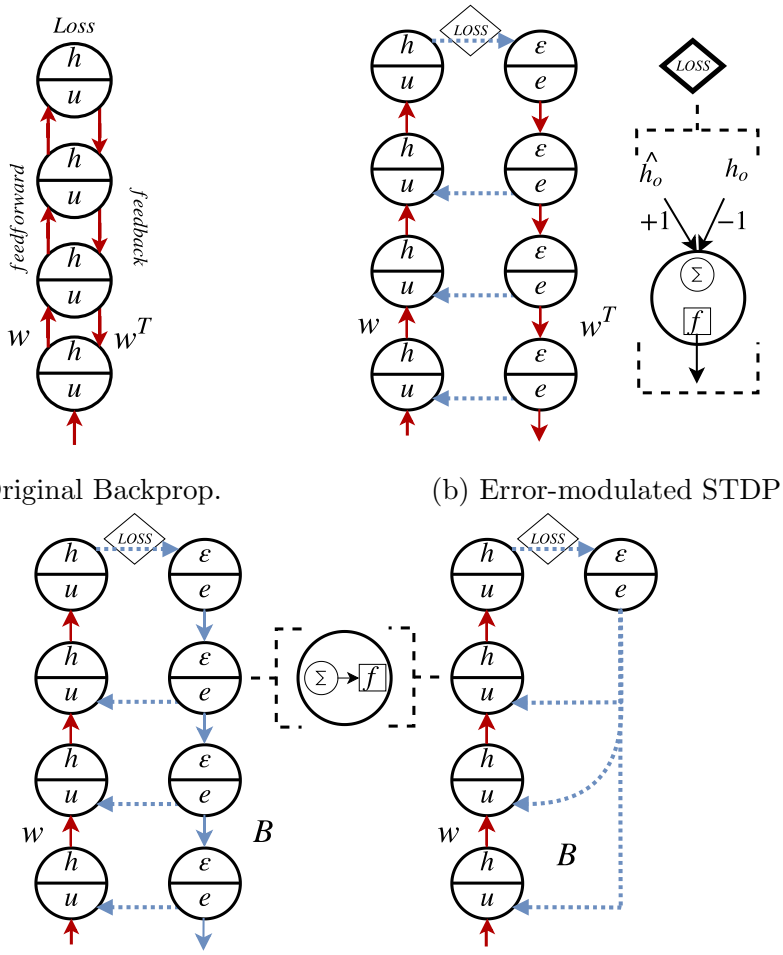
$$L(\hat{h_l}, h_l) = \left\| \hat{h_l} - h_l \right\|_2^2$$

where $\hat{h_o}$ and h_o are the target and output spike counts respectively.

$L2$ loss's derivative results in a simple linear combination of the target and the output value

$$\frac{\partial L(\hat{h_l}, h_l)}{\partial h_l} = (\hat{h_l} - h_l)$$

The choice of $L2$ loss is preferable as its derivative can be implemented simply by a specific connectivity as shown in Figure 4.4(b) of IF neurons whose membrane



(a) Original Backprop.

(b) Error-modulated STDP

(c) Error-modulated STDP with FA and DFA

Figure 4.4: Network structures

potential is given as

$$E_l(t) = w_L \hat{s}_l(t) + (-w_L) s_l(t) + E_l(t-1)$$

where $\hat{s_l(t)}$ and $s_l(t)$ are the target and output spike trains respectively and w_L is the non-plastic synaptic weight to the neuron that asynchronously computes the loss gradient in terms of spikes. The neural circuit is shown in Figure 4.4(b) with $w_L = 1$.

Thus, the accumulated subthreshold membrane potential of the output loss derivative is

$$\epsilon_l = w_L \hat{h}_l + (-w_L) h_l$$

And its propagated backwards as the error derivatives for the hidden layer as per Eq. 4.9 with a threshold θ_l. As the loss is determined per spike basis, the target output firing rate dictates the choice of θ_l. For example, if $w_L = 1$ then for a target output firing rate of 0.2 we fix the θ_l to 5.

4.3.5 Feedback Alignment

Probably, one of the most important issues is the weight transport problem in back-propagation in terms of biological plausibility. Two coherent copies of the weight must be stored in both the forward path and backward path, but there is no known biological mechanism for synapses to know the synaptic strength of other synapses. Also, since the weight is constantly changing during learning, maintaining the coherence between the two copies will create significant overhead. [95] proposed the Feedback Alignment (FA) method, which involves using fixed and random weights for the feedback path to convey error signals with no assumptions on its structure. In fully connected networks with FA, it was observed that the angle between the back-propagated gradient with symmetric weights and the FA propagated gradient converges from approximately orthogonal to roughly 45°, meaning that the FA weight updates are correlated but not identical to those with symmetric weights. [96] introduced direct feedback alignment (DFA) in which the output error signal is propagated directly to all hidden layers instead of through adjacent layers using fixed and random weights. In this work, we adopt both FA and DFA schemes to avoid the weight transport problem. The resulting networks are shown in Figure 4.4(c).

4.3.6 Error-modulated STDP pipeline

The formulation provided in the previous sections shows how to represent the error and backpropagate it in the spike domain. It also provides a guideline to design the feedback path including its connectivity and neuron model as shown in Figure 4.4. In

Algorithm 1: Error-Modulated STDP algorithm

Initialization: Time window T

Spiking neuron i in forward pass

Incoming spike train s_{i-1}

$h_i, \hat{h_i}, h_{i-1} \leftarrow 0$

begin

  while $t < T$ **do**

    $s_i(t) = 0$

    $U_i(t) += b_i$

    if $s_{i-1}(t) == 1$ **then**

      $U_i(t) += w_{i-1,i}$

      $h_{i-1}++$

    if $es_i(t) == 1$ **then**

      $U_i(t) += \gamma\theta_i$

    if $U_i(t) \geq \theta_i$ **then**

      $s_i(t) = 1$

      $U_i(t) += 0$

      if $t < T/2$ **then**

        h_i++

      else

        $\hat{h_i}++$

  $\Delta w_{i-1,i} = \eta(\hat{h_i} - h_i) \cdot h_{i-1}$

  $w_{i-1,i} += \Delta w_{i-1,i}$

Spiking neuron i in backward pass

Incoming spike train es_{i+1}

begin

  while $t < T$ **do**

    $es_i(t) = 0$

    if $t > T/2$ **then**

      if $es_{i+1}(t) == 1$ **then**

        $E_i(t) += w_{i+1,i}$

      if $E_i(t) \geq \theta_i$ **then**

        $es_i(t) = 1$

        $E_i(t) = 0$

Figure 4.4, red solid directed lines denote plastic synapses, whereas blue lines denote fixed synapses. Figure 4.4(a) denotes a vanilla multilayer perceptron (MLP) with a forward pass of continuous-valued activations and backward pass of continuous-valued error derivatives. Figure 4.4(b) denotes the resultant network structure for EMSTDP derived in Section 4.3.3 and the spiking circuitry for the derivative of $L2$ loss. Figure 4.4(c) denotes the variations of EMSTDP in which weight symmetry is avoided by using feedback alignment and direct feedback alignment methods.

All operations in the network with an arbitrary number of layers are asynchronous except for the weight update which is computed only when the error derivatives have driven the hidden layers to its targets. Algorithm 1 shows the error-modulated STDP algorithm for the SNN with an arbitrary number of layers.

Each input sample is a sequence of spikes (Poisson sampling of input intensity) which is presented for a period of T time steps. Given the local weight update rule Eq. 4.12, requires the postsynaptic spike counts before it's driven to its target and after it settles to the target, it necessitates two phases. The spikes are propagated through the forward pass for the whole period. After a warm-up period, ideally, $\frac{T}{2}$ when the firing rates for the hidden layer settle to a fixed point, the spiking circuit computing the derivative of the loss starts propagating spikes through the spiking error circuit.

The spiking error circuit drives the hidden layers in the forward pass toward its target. In the remaining time period, the neurons in the hidden layers settle towards its target firing rate. During the two phases, the local activity i.e., pre and postsynaptic spike counts are recorded, and at the end of T, the synaptic weight update is calculated by Eq. 4.12.

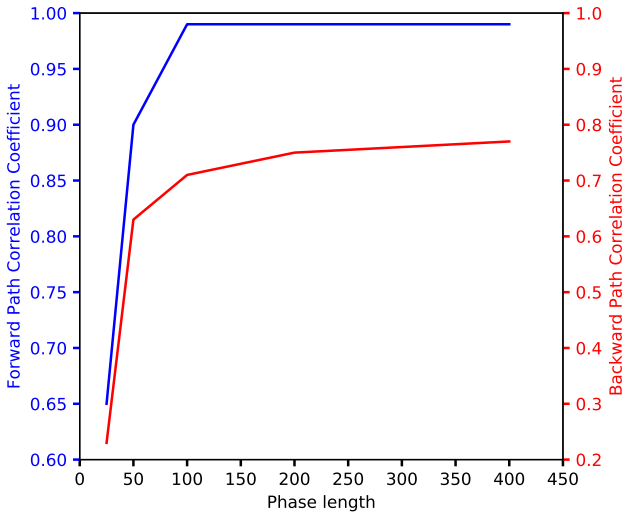


Figure 4.5: Forward and backward path correlation with phase length on MNIST for EMSTDP

4.4 Experiments and Results

In this section, we will present different experiments and their results to evaluate the performance of the error-modulated STDP algorithm. In the first experiment, we can observe the conclusions presented in [94] such that our algorithm produces a profile of weight updates similar to [7]. Then, we will test the algorithm on three datasets; MNIST digit classification, Fashion MNIST and Sign language classification.

Training an SNN can be difficult as the performance can be sensitive to some present hyperparameters. Thus, it is important to adopt and standardize hyperparameter selection during training, this means weight initialization and threshold selection for the algorithm.

In deep learning, it is essential to have a proper initialization method that should avoid reducing or magnifying the magnitude of input signals exponentially [97]. Similarly, in SNNs, it is essential to prevent vanishing and explosion of spikes through the layers. Variance matching initialization schemes attempt to keep $Var(y_L) \approx Var(y_1)$, where y_1 is the output of the first layer and y_L is the output of the last. In order to do so, the following condition must be met, for any given layer l: $\frac{1}{2} n_l Var(w_l) = 1$ where

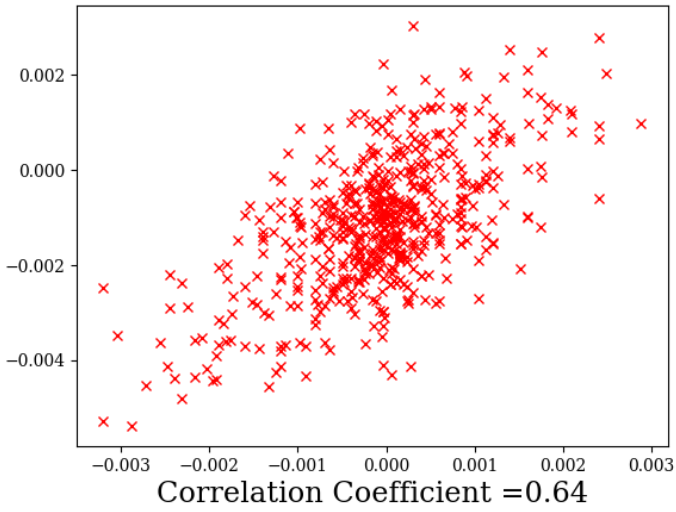


Figure 4.6: Correlation between EMSTDP and a basic STDP weight change

n_l is the number of neurons in the previous layer. Thus, based on [97], we draw the weights from a Gaussian distribution with $\mu = 0$ and $var = \frac{scale}{n_l}$.

Similarly, the choice of thresholds is also important to prevent the extreme sparsity of spikes through the layers. Here, we adopt a simple standardization of the thresholds given by $\theta_l = n_l.\sigma(w_l).\beta$ where β is the constant that determines the percentage of input neurons to spike for the output neuron to spike.

Additionally, we look at the forward and backward path correlation between the SNN and ANN networks in Figure 4.5 to determine the choice of the phase length $T/2$ i.e. window size T. As we can see from Figure 4.5, the correlation coefficients saturate towards phase length 100. Thus, we choose phase length $T/2 = 100$ i.e. window size $T = 200$ for all experiments.

And throughout this section, we will use the following notations for the SNN network structure: Input dimensions are separated by $\times$, layers are separated by $-$ and the last dimension of the network structure is the output layer. For example, $28 \times 28 - 500 - 10$ represents a network input of 28×28 dimension, a hidden layer with 500 neurons and an output layer with 10 neurons.

4.4.1 STDP Observation

[94] has shown that updating the weights in proportion to the rate of change of postsynaptic activity times the presynaptic activity yielded a behavior similar to the STDP observations. They also linked this STDP rule to stochastic gradient descent and suggested that STDP would do gradient descent on the prediction errors if the rate of change of postsynaptic activity is proportional to the gradient of the error. In our formulation of EMSTDP, we have shown that the EMSTDP algorithm satisfies the condition, i.e., the rate of change proportional to the error gradient. Thus, utilizing EMSTDP, we should achieve a strong correlation between our local weight update and a basic two-factor event-based STDP rule that fits the biological findings in [7].

In a simple network with two neurons (pre- and post-synaptic) connected through a synapse, we first simulated random sets of post-synaptic spike trains (s_j) induced by an externally driven voltage or some intrinsic bias and without any pre-synaptic spikes. This is recorded as post-synaptic spike counts (h_j). Then we simulate random sets of pre-synaptic spike trains (s_i) through the synapse with a random set of strengths. For each of these combinations, we record the new post-synaptic spike counts ($\hat{h_j}$) to calculate the respective change in post-synaptic activity. For each configuration, we now compute the weight change based on Eq. 4.12. For the same combinations, we also apply a basic two-factor event-based STDP rule with an STDP window of $10ms$ and record the corresponding total weight change. The resulting Figure 4.6 shows the relationship between weight change for EMSTDP and the basic STDP rule.

4.4.2 MNIST Digit Classification

MNIST [84] is a standard benchmark to test the performance of representation learning algorithms. The task is to classify handwritten digits from 0 to 9. The MNIST handwritten digit dataset consists of 60k samples for training and 10k for testing,

Dataset	Method	Learning	Network Structure	Accuracy%
MNIST	Lee (2016) [33]	BP	$-800-10$	98.64
	Lee (2016) [33]	BP	$-500-500-10$	98.7
	Neftci (2017) [34]	BP	$-500-10$	97.71
	Neftci (2017) [34]	BP	$-500-500-10$	97.98
	O'Connor (2016) [98]	BP	$-300-300-10$	96.4
	Jin (2019) [99]	BP	$-800-10$	98.84
	Diehl (2015) [52]	STDP	$-1600-10$	95
	Tavanaei (2017) [35]	STDP	$-1000-10$	96.6
	Tavanaei (2017) [35]	STDP	$-500-150-10$	97.2
	EMSTDP-SW	STDP	$-500-10$	97
	EMSTDP-DFA	STDP	$-500-10$	96.8
	EMSTDP-SW	STDP	$-500-500-10$	97.3
	EMSTDP-DFA	STDP	$-500-500-10$	96.8
Fashion MNIST	Vanilla BP	BP	$-100-100-10$	87.7
	EMSTDP-SW	STDP	$-500-500-10$	86.1
	EMSTDP-DFA	STDP	$-500-500-10$	85.3
Australian Sign Language	Vanilla BP	BP	$-150-150-50$	98.5
	EMSTDP-SW	STDP	$-150-150-50$	97.5
	EMSTDP-DFA	STDP	$-150-150-50$	97.1

Table 4.2: Performance Comparisons

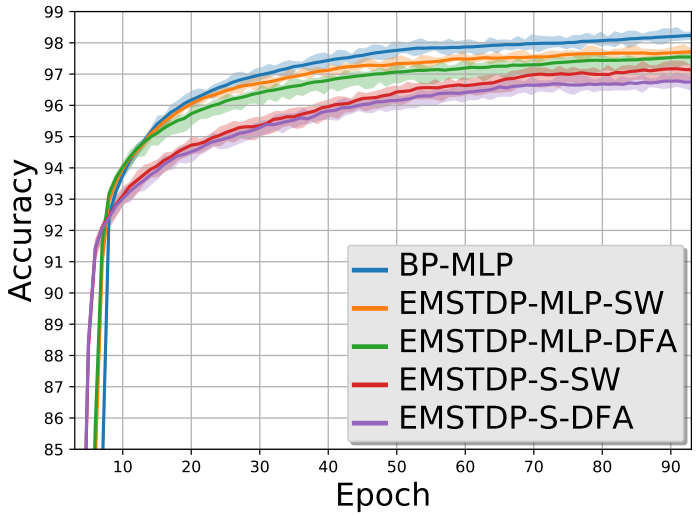


Figure 4.7: Performance comparison of vanilla BP, EMSTDP in MLP and EMSTDP in SNN with symmetric weights (SW) and direct feedback alignment (DFA). The envelopes represent the variance over 5 runs with lowest variation from 10 random initializations

each of which is a 28×28 grayscale image. We perform Poisson sampling based on the pixel intensities to flatten and encode each 28×28 image of the MNIST dataset into a 784 spike trains with duration T. The simulation time step is set to be 1ms. No pre-processing or data augmentation were done in our experiments.

The experiments are carried out for two network structures $28 \times 28 - 500 - 10$ and $28 \times 28 - 500 - 500 - 10$ for symmetric weights (SW) and direct feedback weights (DFA) each. Each configuration is run for a time window of $T = 200$ i.e. phase length of 100 for each spike sequence. Each network is trained for 200 epochs, each epoch containing randomly sampled 10k images from the training set. For comparison, we also train an MLP with vanilla back-prop (BP-MLP) and an MLP with a network structure and weight update as given by EMSTDP (EMSTDP-MLP). All the results presented in this section are an average of 5 trials after 200 epochs for the best set of parameters obtained from an extensive grid search.

Figure 4.7 shows the convergence comparison for BP-MLP, EMSTDP-MLP, and EMSTDP with spiking neurons (EMSTDP-S) with symmetric weights (SW) and di-

rect feedback weights (DFA) for a time window T and two-layered architecture. As expected, MLP with backpropagation converges faster and to a higher accuracy (98.2%) than EMSTDP-MLP (97.8%) which is a modification of difference target propagation in which the change in activity does not purely represent the error derivative. The spiking EMSTDP performed slightly worse (97.3%) that the MLP counterpart, which is again expected due to the loss in precision due to spike encoding and approximations in the forward and backward pass. The DFA versions of EMSTDP-MLP (97.5%) and EMSTDP-S (96.8%) also perform worse than their symmetric weight counterparts as DFA is a less exact optimization method compared to BP with symmetric weights [96].

In the experiments, we also observed that with a larger time window T there is faster convergence and better accuracies. This is expected as the input spikes trains are generated with Poisson sampling and a larger time window allows the spike train to better represent the input pixel values. Similarly, through the hidden layer, the precision of the approximated activation function is better with higher time window.

Table 5.1 shows our results in comparison to related works. The results show that EMSTDP produces competitive to better results as compared to other works such as [35] where the learning rule is only temporally local or performs regular forms of STDP [52]. Our work produces comparable but slightly worse results than related works which only approximate backpropagation in an SNN [34] [33] [99] where the error is non-spiking. The loss in performance can be assigned to the approximation and propagation of error derivatives using spiking neurons and also to the attempt to adapt backpropagation to a local weight update rule.

4.4.3 Fashion MNIST

Fashion-MNIST [100] is a dataset comprising of 28×28 grayscale images of 70, 000 fashion products from 10 categories. The training set has 60, 000 images and the test

set has 10,000 images and it is intended to serve as a direct drop-in replacement for the original MNIST dataset, as it shares the same image size, data format and the structure of training and testing splits. Table 5.1 shows our results in comparison to an ANN with vanilla BP. The results show that EMSTDP produces competitive results for this dataset as well.

4.4.4 Sign Language

We trained an MLP with back-prop, SNN with EMSTDP-SW and EMSTDP-DFA with network structure $45 \times 22 - 150 - 150 - 50$ to recognize Australian sign language symbols. The Australian sign language dataset [101] involves movements of the hand, wrist, and fingers. The data set is collected from two Fifth Dimension Technologies (5DT) data gloves, one right and one left. Each sample is a sequence of data frames containing sensor data. Each sign has approximately 45 frames on average. Since the number of samples is limited, we augmented the data by adding 10% noise. The objective in this domain is: given labeled recordings of different signs, learn to classify an unlabelled instance. Table 5.1 shows the corresponding results.

4.5 Conclusion

In this work, we devised a set of approximations and formulations to move back-propagation towards a more biologically plausible local learning algorithm EMSTDP. We first separate the forward and backward pass of the back-prop algorithm into two separate spiking networks. We formulate the discontinuous activation function of the forward and backward in terms of spike counts such that it approximates a ReLU function and a linear function respectively. We then apply the idea of difference target propagation to derive a local credit assignment rule which resembles as well as produce STDP characteristics. And finally, we solve the problem of weight symmetry in back-prop by utilizing direct feedback alignment. We tested this algo-

rithm on MNIST, Fashion MNIST, and Australian sign language dataset with results comparable to better than related works.

Chapter 5

In-Hardware Learning of Multilayer Spiking Neural Networks on a Neuromorphic Processor

5.1 Introduction

Emerging neuromorphic hardware that adopts the event-driven behavior of spiking neural networks (SNN) has outstanding energy efficiency and is believed to be effective for edge computing or working with a certain type of sensor, such as dynamic vision sensor (DVS), whose output is sparse by nature [102, 15, 103]. However, the lack of a unified robust learning algorithm limits the SNN to shallow networks with low accuracies. The commonly accepted synaptic plasticity rule, the Hebbian rule, is an unsupervised learning, which cannot be directly applied to solve the vast majority of machine learning problems that require supervised learning. A common approach is to train an artificial neural networks (ANNs) and convert it into spiking neural

networks [52, 104] , however, this requires the training to be performed offline.

In-hardware learning is the first step towards building an autonomous machine that learns continuously from its environment and experiences. It means performing inference and learning using the same model running on the same hardware platform. It provides the ability to compensate any device variation and/or environment noise that may exist in the inference stage of the learning process and the possibility of incremental learning when new classes become available to the system being deployed. However, due to the limited storage capacity of low cost edge devices, the training data for real life in-hardware learning is streamed instead of batched. Thus, the learning must be online. This makes in-hardware learning a more significant challenge than offline learning.

Backpropagation, a gradient-based optimization algorithm, is a standard training technique for ANNs. However, as mentioned in Chapter 4, it cannot be directly applied to the in-hardware learning of an SNN running on a neuromorphic processor due to several reasons; (1) spiking neuron's activities are not differentiable, while to backpropagate error, the derivatives of neuron's activation needs to be calculated, (2) the connections between neurons in SNN are unidirectional such that a backward path must be added explicitly with constantly updated weights during learning , (3) errors in ANN are propagated as real values and (4) weight update of a synapse is not dependent only on locally available information as required in a neuromorphic hardware [5].

Most works approximating backpropagation for SNNs have some limitations. Some require neurons to have high-precision backpropagating error derivatives [90, 35, 98]. In some other works, neuron must know the membrane potential of its presynaptic neighbors in order to determine the synaptic weight change [99]. In [35, 98, 34], each neuron must also know the error derivatives of its postsynaptic neighbors in order to calculate its own error derivative. All these limitations violate the constraints of

the local communication rule in the spike domain and require additional hardware to support the backpropagation of real-valued error derivatives and communication of membrane potentials. Meanwhile, approaches [91][92], which convert trained ANNs to SNNs, are not conducive to neuromorphic implementation.

To the best of our knowledge, the Error-Modulated Spike-timing-dependent plasticity (EMSTDP) algorithm [37], discussed in Section 4, is the only supervised learning model that applies the same type of integrate and fire (IF) neuron in the forward and backpropagation path. The algorithm enhances the biological plausibility of backpropagation by introducing a weight update rule that resembles the rate-based STDP [94]. However, there is still a gap between the EMSTDP algorithm and the in-hardware online learning on a neuromorphic hardware. First of all, EMSTDP assumes arbitrary data precision in weight coefficients and unlimited neuron and synapse resources. Secondly, the weight update rule of EMSTDP and the error modulation process still require special hardware support. Finally, many of the existing data sets were designed for ANN. The inputs are vectors of real numbers. Traditional rate coding that represents the real number as a sequence of spike trains will introduce a significant amount of I/O activity. In this work, we bridge the gap and adapt EMSTDP to fit the constraints of a neuromorphic hardware and implement it on Intel's Loihi chip. The following four approximation techniques were introduced that enable the neuromorphic hardware implementation of the EMSTDP :

- Multi-compartment neurons were adopted in the feedback path where the error spikes can be gated by the activities of the feedforward neurons.
- The change in the post-synaptic spiking rate is approximated by using the built-in post-synaptic trace counter and a two-phase operation.
- Direct feedback alignment is adopted to significantly reduce the number of neurons in the feedback path and hence lower hardware cost and improve the learn-

ing accuracy.

- Reduce the I/O activities by programming the bias of the input neurons using the real-valued inputs and generating the spike sequences inside the chip.

The cost and accuracy impact of these techniques will be discussed in the paper. This is the first work of fully spike-based online in-hardware supervised learning on a neuromorphic hardware. The implementation is tested on MNIST, Fashion-MNIST, CIFAR-10 and MSTAR datasets with promising performance and energy efficiency. We also demonstrate a possibility of incremental online learning with the implementation.

5.2 Background

5.2.1 Error-Modulated STDP

[37] extended the backpropagation algorithm for SNNs through a suitable sequence of approximation techniques. First, the forward and backward paths are separated into separate networks with spiking neurons. Simple Integrate-and-Fire (IF) neurons are used in both paths. For the forward path, the membrane potential of neurons in layer i at time t is:

$$U_i(t) = \sum w_{i-1,i}.s_{i-1}(t) + b_i + U_i(t-1) \tag{5.1}$$

$$\text{if} \quad U_i(t) \geq \theta \quad \text{then} \quad s_i(t) = 1, \quad \text{else} \quad s_i(t) = 0$$

where $w_{i-1,i}$ is the synaptic weight between pre-synaptic neurons in layer $i-1$ and post-synaptic neurons in layer i, b_i is the bias. $s_{i-1}(t)$ and $s_i(t)$ are pre and postsynaptic spike trains, respectively. The neuron spikes when the membrane potential reaches the threshold θ and is reset to the resting potential of 0. Its activation

function is approximated in terms of the spike count h, accumulated sub-threshold membrane potential u, and spiking threshold θ over a duration T:

$$h = f(u) = \left\lfloor \frac{u}{\theta} \right\rfloor \tag{5.2}$$

Its derivative is approximated as that of a shifted ReLU.

ϵ_i represents the backpropagated error derivatives of the i_{th} layer, based on the backpropagation algorithm such that

$$\epsilon_i = \epsilon_{i+1} \cdot h'_{i+1} \cdot w^T_{i,i+1} \tag{5.3}$$

And the error spike count e in the error path is represented as:

$$e_{i+1} = \epsilon_{i+1} \cdot h'_{i+1} \tag{5.4}$$

Such that,

$$\epsilon_i = e_{i+1} \cdot w^T_{i,i+1} \tag{5.5}$$

This means that we can represent the error derivative also using spikes, and use the same IF neurons (without bias) given by Eq. (5.1) in the feedback path. ϵ_i is the accumulated sub-threshold membrane potential of the error neuron whose input is the spikes e_{i+1} from the upper-level error neuron. In addition to the threshold based firing, the output of the neurons in the feedback path is also gated by h'_i, which is a constant when the neuron in the corresponding feedforward layer has output activities and zero otherwise.

L2 loss's derivative is entirely spike-based. To represent that, the accumulated subthreshold membrane potential of neurons in the first layer in the feedback path is calculated as

$$\epsilon_l = w_L \hat{h_l} + (-w_L) h_l \tag{5.6}$$

where $\hat{h}$ and h are the target and predicted spike trains.

Local learning is achieved through the idea of target propagation. In [93] , the targets are determined utilizing an autoencoder with inverse synaptic weights which are learnt through reconstruction. In EMSTDP, the error derivatives are used as the difference to determine the target spike count $\hat{h}_l$. Such that the local weight update is formulated as

$$\Delta w_{i-1,i} = \eta \cdot \left(\widehat{h_l} - h_i\right) \cdot h_{i-1} \tag{5.7}$$

where η is the learning rate. This computation is done in two phases: the first phase to compute h_i, and second phase to compute $\hat{h}_i$ and perform the weight update. Since, the weight is constantly changing during learning, maintaining the coherence between the two copies will create significant overhead. EMSTDP adopts the Feedback Alignment (FA) method [95], to relax this constraint. Instead of maintaining an exact copy of the weight in both the feedback and feedforward path, a random weight matrix is used in the feedback path. In fully connected networks, it was observed that the backpropagated gradient with symmetric weights and the FA propagated gradient are correlated but not identical.

5.2.2 Synaptic Plasticity in Loihi

Loihi [5] is a digital neuromorphic chip recently developed by Intel. Loihi's 128-neuromorphic cores implement 130,000 artificial CUBA leaky- integrate-and-fire neurons and 130 million synapses. The CUBA neurons have two internal state variables; synaptic response current $u_i(t)$ and membrane potential $v_i(t)$. The synaptic response current is the decaying weighted incoming spikes and the membrane potential is formulated as:

$$\dot{v_i}(t) = -\frac{1}{\tau_v} v_i(t) + u_i(t) \tag{5.8}$$

Here, the integration is leaky as captured by the time constant τ_v . The neuron

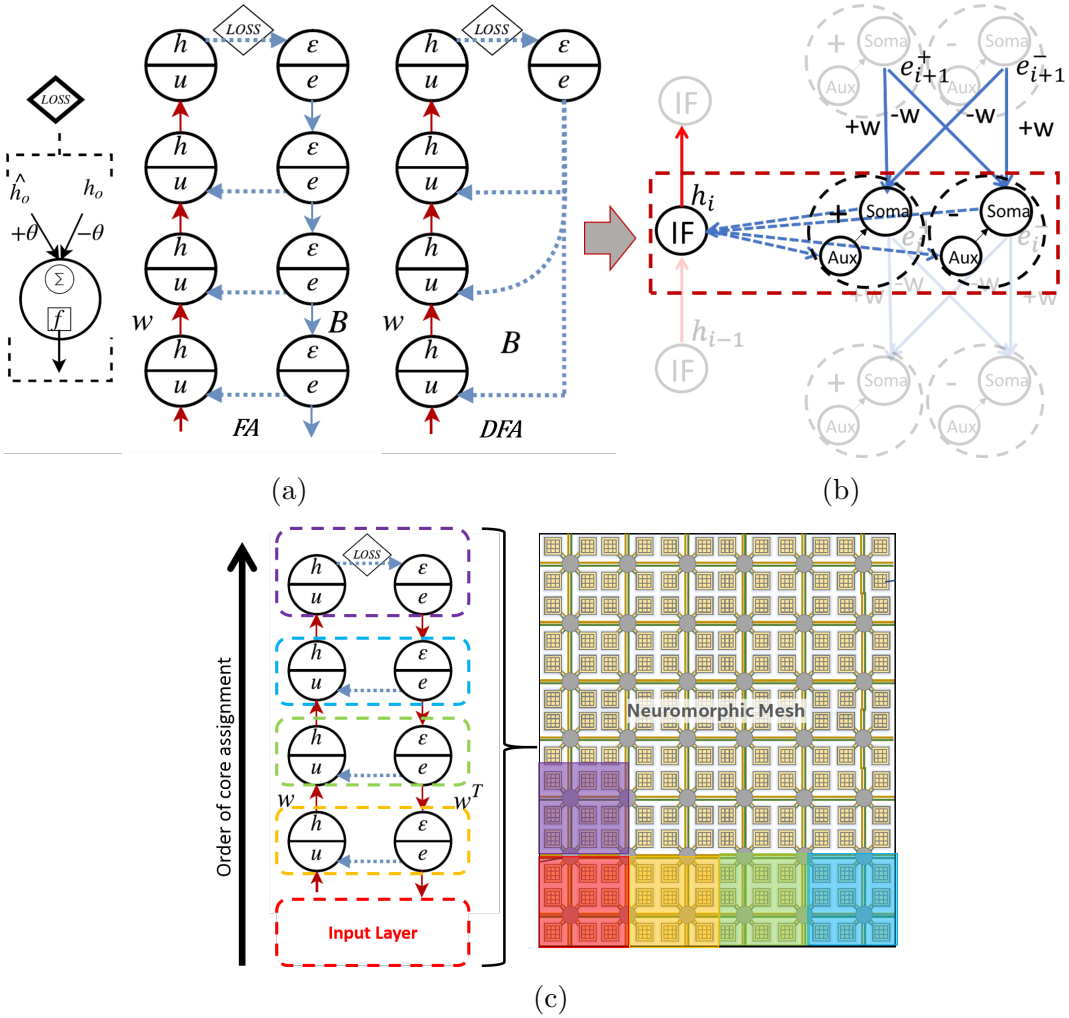


Figure 5.1: (a) EMSTDP - Network (b) Loihi - Network (c) Loihi - Mapping

sends out a spike when its membrane potential passes its firing threshold θ_i and is reset to 0 right after. The neuron parameters are tunable, thus they can be adapted to an IF neuron and other more complicated behaviors.

Loihi also provides a programmable microcode learning engine for on-chip SNN training. In Loihi, each synapse is associated with integer-valued synaptic variables and multiple presynaptic traces, and whereas compartment with postsynaptic traces. These traces enter the learning engine as input variables for synaptic weight adaptation. SNN synaptic weight adaptation rules must satisfy a locality constraint: each weight can only be accessed and modified by the destination neuron, and the update

can only depend on locally available information, such as the spike trains from the presynaptic (source) and postsynaptic (destination) neurons. The functional form of the adaptation rules to apply is described in sum-of-products form:

$$z := z + \sum_{i=1}^{N_p} S_i \prod_{j=1}^{n_i} (V_{i,j} + C_{i,j}) \tag{5.9}$$

where z is the transformed synaptic variable (weight, delay or tag), $V_{i,j}$ refers to some choice of input variable (synaptic variables and traces) available to the learning engine, and $C_{i,j}$ and S_i are microcode-specified signed constants. Regular pairwise and triplet STDP rules can be implemented along with more complicated adaptation rules utilizing this form.

Loihi has been shown to produce impressive performance and energy efficiency for various tasks both for inference-only and in-hardware learning tasks. However, there is still a lack of an online supervised learning algorithm to train multilayered SNNs in Loihi due to the implicit constraints relating to the neuromorphic hardware such as precision, connectivity across cores etc and also the constrained functional form of the learning rule. The flexibility and in-hardware learning capabilities make Loihi a suitable platform for EMSTDP learning.

5.3 Adapting EMSTDP for in-hardware online learning

5.3.1 Forward and Error Path

Since the synaptic connection between neurons in Loihi is directional, we have to keep two copies of the networks, one for forward pass and the other to backpropagate errors. And as mentioned in Section 5.2.1 and as keeping updated weights in the

error path is not viable, we utilize FA weights B. Later, we will show that the error path can be simplified away using DFA, which reduces the hardware usage and power consumption. As shown in Figure 5.1b, both networks are built with IF neurons. We simply configure the LIF neuron in Loihi into an IF neuron for the forward path. For that, we utilize the maximum time constant τ_v in Eq. (5.8) such that the membrane potential does not leak over time whereas the current decays immediately.

In order to represent both positive and negative errors in the error path, we utilize two channels of spiking neurons. Thus, it is necessary to cross-connect between the positive and negative channels of the source and target layers as shown in Figure 5.1b. In both channels, the spike rate represents the absolute value of the error. Let ϵ_i^+ and ϵ_i^- represents the accumulated sub-threshold membrane potential in the positive and negative channels in the i_{th} layer. Following Eq. (5.5), they can be calculated as the following:

$$\begin{cases} \epsilon_i^+ = e_{i+1}^+ \cdot w_{i,i+1}^T + e_{i+1}^- \cdot (-w_{i,i+1}^T) \\ \epsilon_i^- = e_{i+1}^+ \cdot (-w_{i,i+1}^T) + e_{i+1}^- \cdot w_{i,i+1}^T \end{cases} \tag{5.10}$$

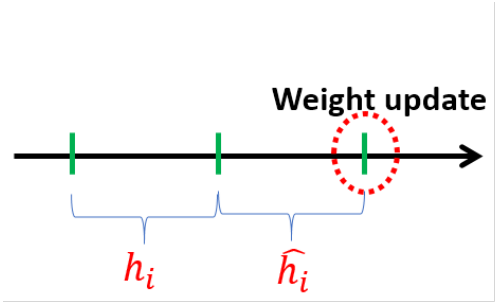
Additionally, we have to consider h', i.e., the derivative of the neuron activation function in the forward path to gate the output activities in the feedback path as described by Eq. (5.4) and Eq. (5.5). Since h' is either 0 or a constant value, this can be implemented utilizing an AND function in the case of bit streams. We utilize the multi-compartment neurons in Loihi for that purpose. Two-compartment neurons with a soma compartment and a corresponding auxiliary compartment are set up for the error path such that the spiking activity of the soma is an AND function of the activity of the soma and the auxiliary compartment. The setup of the forward and error path is shown in Figure5.1b.

The original EMSTDP with FA has a one-to-one correspondence in the feedback and feedforward neurons. It does not only double the hardware requirement but also

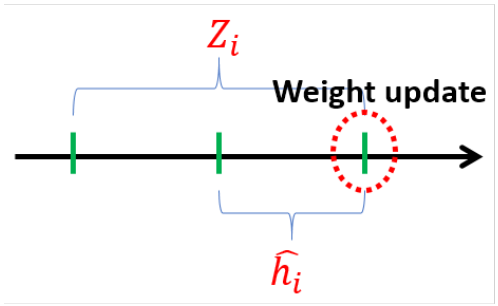
degrade the learning quality. As the error propagated through layers, the quantization errors accumulated. In this work, we further adopt the direct feedback alignment (DFA) [96]. With DFA, we broadcast the error spike directly from the spike-based loss function inEq. (5.6) with uniformly distributed random weights whereas FA requires an error network through which the error spikes propagate down. Figure 5.1a compares the networks using FA and DFA. Compared to FA, the DFA does not only eliminate the neurons on the feedback path, the number of connections on the feedback path is also reduced. This is because the FA connects the neurons in the corresponding hidden layers in the forward and feedback paths, while the DFA directly connects the error neurons in the output layer to the hidden layers in the forward path. Because the dimension of the output layer is usually smaller than the hidden layer, the weight matrix of DFA is much smaller than that of the FA. Therefore, the DFA not only reduces the number of compartments and neuron cores used in the chip, but also reduces the number of synapses and thus the amount of memory utilized by the synapses in the cores.

5.3.2 Learning

EMSTDP is set up to enable local learning. We need to calculate the product of the change in postsynaptic activity and the presynaptic activity as given by Eq. (5.7). The change in postsynaptic activity is computed through the difference of the target spike count $\hat{h}_i$ and the original spike count h_i, thus, requiring a two-phase operation occurring in a window of time T. In Phase 1 (time 0 to $T/2$), the neurons on the forward path respond to the input, and settle down at a specific spiking rate h. In Phase 2 (time $T/2$ to T), the neurons in the backward path pass the errors to the neurons in the corresponding layers in the forward path. The errors "correct" the behavior of those feed forward neurons, and drive their spiking rate to $\hat{h}$. All operations in the network with an arbitrary number of layer are asynchronous except



(a)



(b)

Figure 5.2: EMSTDP weight update (a) Original (b) Loihi

for the weight update which is computed only at the end of T.

As the weight update rule should be tailored to a sum of products form utilizing variables satisfying the locality constraint, EMSTDP update rule in Eq. (5.7) can simply be represented as

$$\Delta w_{i-1,i} = \eta \cdot \widehat{h_l} \cdot h_{i-1} - \eta \cdot h_i \cdot h_{i-1} \tag{5.11}$$

However, this representation requires preserving information from the first phase h_i to use at the end of the second phase as shown in Figure 5.2a. Loihi has no means to save this particular information for later use. Thus, we modify Eq. (5.11) such that all the information is available at the end of the second phase when the update is performed as shown in Figure 5.2b.

$$\Delta w_{i-1,i} = 2.\eta \cdot \widehat{h_l} \cdot h_{i-1} - \eta \cdot Z_i \cdot h_{i-1} \tag{5.12}$$

Where $Z_i = \widehat{h}_i + h_i$. In Loihi, we represent presynaptic spike count h_{i-1} by the pre-trace, postsynaptic target spike count $\widehat{h}_i$ by the post-trace and Z_i by the available tag synaptic variable.

5.3.3 Mapping to cores

The core-based architecture of Loihi introduces the limitations of fan-ins and fan-outs, which is not always reasonable for larger networks. Thus, it's necessary to develop a mapping algorithm to deal with those constraints. Here, we utilize a simple mapping algorithm where the neurons are mapped incrementally onto the cores satisfying the constraints a layer at a time as shown in Figure1(c) and Operation Flow 2. For this, we first generate the adjacency matrices for the connectivity between adjacent layers (convolution and dense). This provides the number of fan-ins and fan-outs for each neuron, which is used to assign the number of neurons per core. This method not only helps ensure that the constraints are in check but also utilizes the cores optimally by managing the trade-off between the execution time and power usage. This trade-off is analyzed in Section 5.4.1.2.

5.3.4 Running EMSTDP on Loihi

Operation Flow 2 shows the entire process of running the EMSTDP algorithm on Intel's Loihi. After creating the network N in Intel Loihi's SDK, we initialize the weights W and the random fixed weights B sampled from a uniform distribution and then quantize and scale them to 8 bit integers to fit Loihi's synapses. Now, we have a network which can be mapped into Loihi which is done during compilation and then deployed.

During runtime, in each epoch during training, for each sample x, the values are quantized to the length of the phase $T/2$. This allows for rate coding the input over the

phase length. Normally, inputs are provided through rate-coded spikes directly. Each input data corresponds to a sequence of spikes at the corresponding rate. Each spike insertion requires a communication between the host and the chip, thus a significant overhead. Instead of inserting spikes directly, we program the the biases of the input layer neuron, such that $U_{in}(t) = U_{in}(t-1) + i,\ 0 \leq t \leq T/2$, where i is the input and $T/2$ is the phase length. The accumulated membrane potential can be calculated as $u_{in} = \sum_{t=1}^{T/2} U_{in}(t) = i * T/2$ and the spiking rate of the input neuron is $h_{in} = \left\lfloor \frac{u_{in}}{\theta} \right\rfloor$ which is linearly proportional to input i. In other words, this allows the input layer to integrate the bias over time, thus producing spikes with the rate directly proportional to the bias. Using this setup, we need to communicate with the chip only once for every input sample. We insert the label as a bias as well to label neurons. The network then goes through the two phases of the EMSTDP algorithm, updates the weights, and then resets the network states (membrane potential and traces) to 0.

5.4 Experiments

5.4.1 Online Learning

In this experiment, we evaluate online learning using EMSTDP algorithm on Loihi. In real life applications of in-hardware learning, the training data is received as a stream, and training must be carried out in real-time as the data is received. Techniques such as batch learning, data augmentation are not feasible in such an online learning scenario. Biological neural systems may use multi-modal sensing and fusion as regularization in the learning process, however, this is not within the scope of this paper.

For all the experiments, we use a network with structure $W \times H \times C - 5 \times 5k \times 16c2s - 3 \times 3k \times 8c2s - 100d - 10d$ (W: width, H: height, C: channel of the input, k: kernel size, c: number of filters, s: stride size, d: dense layer), and learning rate

Operation Flow 2: In-Hardware Learning on Loihi

Epoch E, **Training set** X, **Testing set** Y, **Network** N;
Trainable Layers L with l_n neurons, **Phase Length** $T/2$;
Create Network N;
Initialize N **and Quantize to 8 bits**: W, B;
Compile and Deploy Network N on Loihi;
for ***each*** $l \in L$ **do**
    Build $l-1 : l$ adjacency matrix;
    Compute l_m Optimal neurons per core for l;
    Map l_n to $\frac{l_n}{l_m}$ cores;
end
for $epoch \leftarrow 0$ **to** E **do**
    for ***each*** $x \in X$ **do**
        Quantize x to $T/2$ bins;
        Set input and label bias: x;
        EMSTDP Phase 1: $T/2$ time steps;
        EMSTDP Phase 2: $T/2$ time steps;
        Update W;
        Reset network state;
    end
end
for ***each*** $y \in Y$ **do**
    Quantize y to $T/2$ bins;
    Set input bias: y;
    EMSTDP Phase 1: $T/2$ time steps;
    Evaluate;
    Reset network state;
end

$\eta = 2^{-3}$. Here, the convolutional layers are pretrained with their respective datasets before mapping on to Loihi whereas the dense layers are trained from scratch in the Loihi. This introduces opportunities for transfer learning when training such convolutional layers in-hardware is not viable.

In the experiments, we compare the learning performance of EMSTDP running on Loihi with its software (Python) implementation with batch size 1 and full precision weights.

Table 5.1: Performance

	Loihi	Python (FP)	Loihi	Python (FP)
MNIST	94.5%	98.9%	94.7%	98.9%
Fashion-MNIST	84.3%	92.7%	84.8%	92.5%
MSTAR (10 class)	78.4%	83.5%	79.5%	83.3%
CIFAR10	61.6%	64.2%	62.2%	64.4%
	FA		**DFA**	

Table 5.2: Power and Energy

	FPS	Power (W)	Energy (mJ/img)	FPS	Power (W)	Energy (mJ/img)
i7 8700	422	58	137	1536	58	37
RTX 5000	625	48	77	2857	47	16
Loihi	50	0.42	8.4	97	0.24	2.47
	Training			**Testing**		

5.4.1.1 Performance

We experiment on four datasets; MNIST, Fashion-MNIST, CIFAR-10 and MSTAR. MNIST [84], Fashion-MNIST [100] and CIFAR-10 [105] are standard benchmarks. The MSTAR dataset [106] is a collection of SAR (Synthetic Aperture Radar). The subset we use is the MSTAR/IU Mixed Targets with 10 classes of vehicles. The images are target chips taken from scenes of SAR images, where each chip is 128 by 128 which we center-crop 64x64 and resize it to 32x32. For each dataset, we convert the pixels into rate coded spike trains of duration $T/2$. The choice of phase length $T/2$ as 64 is chosen based on the trade-off shown in Figure 5.3. The Loihi learning engine allows for the maximum phase length of 64. In Figure 5.3, at $T/2 = 64$, we achieve the best accuracy at a FPS suitable for real-time application. Lowering $T/2$ increases the FPS but also degrades the accuracy dramatically.

Table 5.1 shows the testing results on MNIST, Fashion-MNIST, CIFAR-10 and MSTAR dataset for EMSTDP (both FA and DFA). We see that even with the constraints in Loihi we achieve respectable performance as compared to the full precision implementations. The drop in performance on Loihi can be attributed to the quanti-

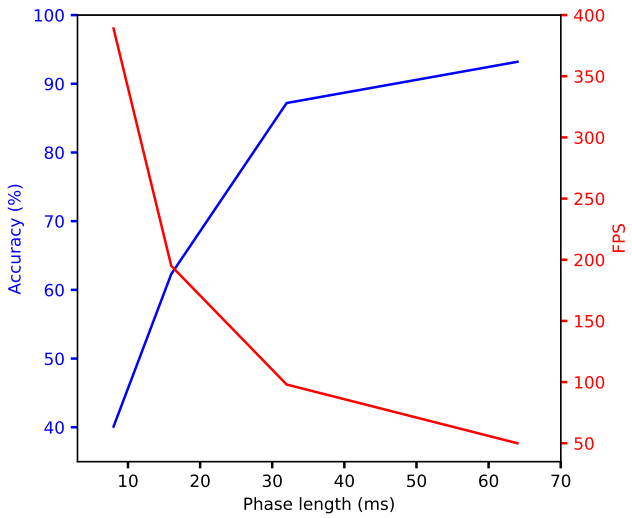


Figure 5.3: Trade-off between FPS and Accuracy with phase length on MNIST for EMSTDP on Loihi with FA while training with 10000 samples

zation error due to the limitation of 8 bit weights and computation in Loihi. We can also see that the DFA based system has slightly better accuracy. This is probably because the DFA skips the hidden layers in the backward path and hence has less accumulated quantization errors.

5.4.1.2 Power

Table 5.2 shows power and energy comparison with CPU (i7 8700) and GPU's (RTX 5000) versions with batch size 1. Given that Loihi's maximum operating frequency is 10KHz and the SNN requires $T/2$ time steps to operate in each phase, each sample takes a longer time than in CPUs and GPUs. The throughput is sufficient for online image processing. Reducing the duration of each phase will improve the throughput but also sacrifice the quality of learning. The active power consumed by Loihi running these networks are several orders lower than CPUs and GPUs. This can be attributed not only to the asynchronous and simplified architecture of Loihi but also the sparse nature of SNNs. As we are learning SNNs in-hardware, we compare the energy per sample for both training and testing time. During the inference mode, backward

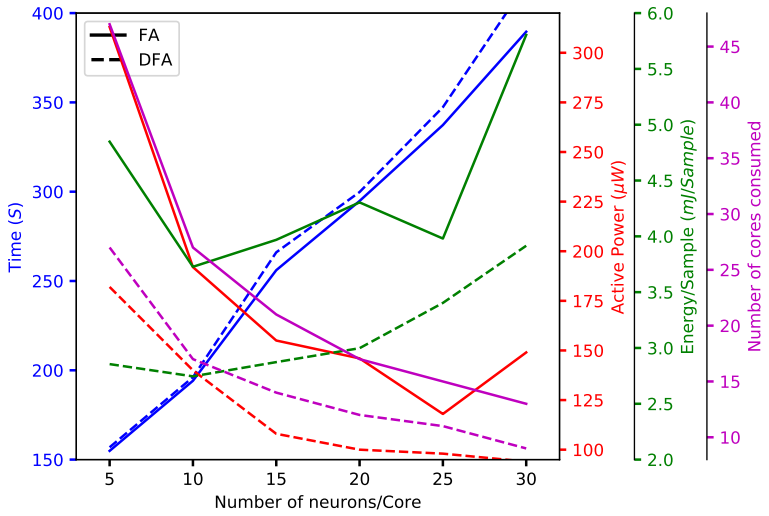


Figure 5.4: Trade-off between FPS and Active power consumption through Energy/Sample for EMSTDP on Loihi with FA and DFA while training with 10000 samples

paths are not implemented, hence it has higher throughput and lower energy per image than the training model.

In Figure 5.4, we show the effect of the iterative mapping of compartments (Section 5.3.3) in the Loihi cores during training. When we increase the number of neurons mapped to each core, the number of occupied cores is reduced and the active power decreases as the cores that are not in use are power gated. At the same time, the execution time increases as the core is shared by a higher number of neuron compartments. As a result, the energy per sample first decreases and then increase. There is a best mapping size that minimizes the energy and it is necessary to analyze the trade-offs between the execution time and the power usage. For Table 5.2, we chose 10 logical neurons in the hidden layer to be packed in a single core based on Figure5.4.

5.4.1.3 FA versus DFA

As we discussed in Section 5.3.1, utilizing DFA reduces the number of total compartments, cores, and synapses. This effect can be seen from Figure 5.4 where DFA based system consistently utilizes less cores, thus consuming less active power. Because

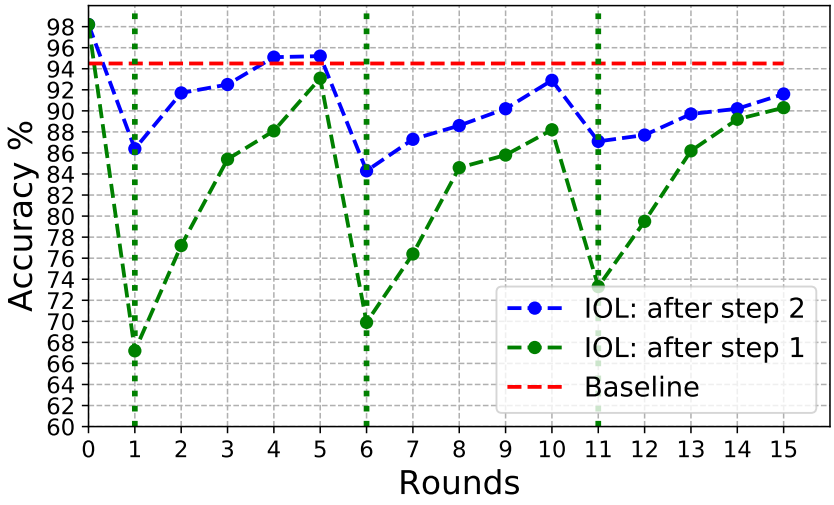


Figure 5.5: Incremental Online Learning with MNIST: 2 new classes are introduced at rounds marked by the green dotted line (Baseline: result from the same network trained with the entire dataset, IOL: Incremental Online Learning)

the execution time is determined by the number of neurons per core, DFA and FA have similar throughput at the same neurons per core level. However, at the same performance level, DFA consumes lower power and dissipates less energy per image.

5.4.2 Incremental Online Learning

The in-hardware online learning provides the flexibility for the system to learn new classes after deployment or compensate possible model errors due to a poorly chosen training set or potential sensor hardware drifting. Such adaptability cannot be found in a system running models trained offline. In the second experiment, we demonstrate such flexibility using incremental online learning.

We adopt a method similar to [107]. [107] utilizes an alternating two-step learning technique; (1) learn new classes, (2) retrain with new and old classes. A modified cross-distillation loss is used in step (1) to enable the model to minimize catastrophic forgetting which is a phenomenon where the performance on the old classes degrades dramatically as new classes are added. A regular cross-entropy loss is used in step (2) in which the retraining is done with the recently observed new classes and an equal

size sample of old classes from a set with new observations of old classes as well. These new observations may have a different distribution to that of old observations or could simply be noise or variations caused by the input device/sensor. This two step method helps to minimize concept drift [107] caused by these new observations in a standard online learning set up. Here, we set up our experiment to follow the same steps.

For this experiment, we pretrain a model for 4 randomly selected classes in the MNIST dataset. Then we have three incremental training iterations, each time 2 new classes were added. We also divide 6000 samples of each class into 5 chunks such that we can introduce the new classes over 5 rounds of learning new classes and retraining for each incremental training iteration. To approximate the effect of cross-distillation loss, we disable the classifier neurons of the old class and reduce the learning rate during step (1). In each round, the sampling of old classes for step (2) is done from a set with both old and new observations of the old classes. Figure 5.5 shows the accuracy of the observed classes for all the incremental training steps at each round at the end of step (1) and (2) for the same network used for MNIST in Section. 5.4.1. As our method is only approximating cross-distillation, catastrophic forgetting can have a large effect on our model's performance seen from the green dotted line in the beginning of each incremental training iteration. However, spreading the introduction of new classes over 5 rounds helps to alleviate that and recover. Thus, we see a big drop in accuracy in the first round of new classes and then recovery over the next rounds in each incremental learning step.

5.5 Conclusion

In this work, we adapted a spike-based backpropagation algorithm with biological plausible local update rules to fit the constraints of a neuromorphic hardware

and implemented it on Intel's Loihi chip. This resulting system enables low power in-hardware online supervised learning of multilayered SNNs, which was tested on MNIST, Fashion-MNIST, CIFAR-10 and MSTAR datasets with promising performance and energy efficiency. We also demonstrate a possibility of incremental online learning with the implementation.

Chapter 6

Conclusion

6.1 Summary

In this thesis, we first reviewed the literature surrounding SNNs including neuron and synapse models along with spike encoding and learning with spikes. Then we introduced neuromorphic hardware along with briefly reviewing two popular neuromorphic systems; TrueNorth and Loihi.

In Chapter 2, we presented a design flow that maps a special case of recurrent networks called Long Short-Term Memory (LSTM) onto a spike-based platform. The framework utilized various approximation techniques; activation discretization, weight quantization, scaling, and rounding; spiking neural circuits that implement complex gating mechanisms, and a store-and-release technique to enable neuron synchronization and faithful storage. While the presented techniques can be applied to map LSTM to any Spiking Neural Network (SNN) simulator/emulator, here we choose the TrueNorth chip as the target platform by adhering to its hardware constraints. Three LSTM applications, parity check, Extended Reber Grammar and Question classification, are evaluated. The tradeoffs among accuracy, performance, and energy tradeoffs achieved on TrueNorth are demonstrated. This is compared to

the performance on an SNN platform without hardware constraints, which represents the upper bound of the achievable accuracy. This is the first realization of LSTMs on a neuromorphic hardware and allows for the use of a neuromorphic hardware for applications requiring LSTMs to process temporal patterns.

In the next chapters, we discussed learning with SNNs. In Chapter 3, we presented a low-cost, simplified, yet stable STDP rule for layer-wise unsupervised and supervised training of a multilayer feed-forward SNN. We proposed to approximate Bayesian neurons using Stochastic Integrate and Fire (SIF) neuron model and introduce a supervised learning approach using teacher neurons to train the classification layer with one neuron per class. A SNN is trained for classification of handwritten digits with multiple layers of spiking neurons, including both the feature extraction and classification layer, using the proposed STDP rule. Our method achieves comparable to better accuracy on MNIST dataset than manually labelled two-layer networks for the same sized hidden layer. We also analyzed the parameter space to provide rationale for parameter fine-tuning and provide additional methods to improve noise resilience and input intensity variations. We further propose a Quantized 2-Power Shift (Q2PS) STDP rule, which reduces the implementation cost of digital hardware while achieves comparable performance.

In Chapter 4, we presented an approximation of the backpropagation algorithm completely with spiking neurons and extend it to a local weight update rule which resembles a biologically plausible learning rule spike-timing-dependent plasticity (STDP). This enabled error propagation through spiking neurons for a more biologically plausible and neuromorphic implementation friendly backpropagation algorithm for SNNs. We tested the proposed algorithm on various traditional and nontraditional benchmarks with competitive results.

In Chapter 5, we adapted a spike-based backpropagation algorithm with biological plausible local update rules to fit the constraints of a neuromorphic hardware

and implemented it on Intel's Loihi chip. This resulting system enabled low power in-hardware online supervised learning of multilayered SNNs which was tested on MNIST, Fashion-MNIST, CIFAR-10 and MSTAR datasets with promising performance and energy efficiency. We also demonstrated a possibility of incremental online learning with the implementation. This is the first work that realized online in-hardware supervised learning of deep SNN in a neuromorphic hardware. This online in-hardware learning is the first step towards building an autonomous machine that learns continuously from its environment and experiences.

6.2 Future Research Direction

In Chapter 4, we presented an approximation of the backpropagation algorithm called Error-Modulated STDP (EMSTDP) with approximation with considerations towards implementation in neuromorphic hardware. The algorithm is only dependent on spiking neurons and is extended to a local weight update rule which resembles a biologically plausible learning rule spike-timing-dependent plasticity (STDP). In Chapter 5, we adapted the EMSTDP algorithm with biological plausible local update rules to fit the constraints of a neuromorphic hardware and implemented it on Intel's Loihi chip. This resulting system enables low power in-hardware online supervised learning of multilayered SNNs. We also demonstrated a possibility of incremental online learning with the implementation.

Incremental online learning is a form of continual learning that attempts to solve two major obstacles that are challenges to implement for real life applications: (1) Learning new classes makes the trained model quickly forget old classes knowledge, which is referred to as catastrophic forgetting. (2) As new observations of old classes come sequentially over time, the distribution may change in an unforeseen way, making the performance degrade dramatically on future data, which is referred to as

concept drift. However, the incremental online learning framework utilized in Chapter 5 involved an alternating step of retraining with samples from already observed classes. For real life applications on devices with limited memory such as autonomous drones and other mobile devices, storing samples from already observed classes could be prohibitive. Thus, to truly enable in-hardware learning for real life applications, it is essential to have continual learning without the need of storing a large amount of retraining data. Thus, in the next step, we need to investigate continual learning approaches in machine learning and adapt them to an online setting in SNNs to enable fully in-hardware learning in neuromorphic hardware.

Bibliography

[1] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, pp. 1659–1671, 1997.

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

[3] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch *et al.*, "Convolutional networks for fast, energy-efficient neuromorphic computing," *Proceedings of the National Academy of Sciences*, p. 201604850, 2016.

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

[5] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, pp. 82–99, 2018.

[6] W. Maass, "Lower bounds for the computational power of networks of spiking neurons," *Neural Computation*, vol. 8, pp. 1–40, 1996.

[7] G. qiang Bi and M. ming Poo, "Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type," *Journal of neuroscience*, vol. 18, pp. 10 464–10 472, 1998.

[8] S. M. Schuetze, "The discovery of the action potential," *Trends in Neurosciences*, vol. 6, pp. 164–168, 1983.

[9] E. R. Kandel, J. H. Schwartz, T. M. Jessell, D. of Biochemistry, M. B. T. Jessell, S. Siegelbaum, and A. J. Hudspeth, *Principles of neural science*. McGraw-hill New York, 2000, vol. 4.

[10] W. Gerstner and W. M. Kistler, *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[11] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of physiology*, vol. 117, pp. 500–544, 1952.

[12] C. S. Sherrington, "The central nervous system," *A text book of Physiology*, vol. 929, 1897.

[13] M. R. Bennett, "The early history of the synapse: from plato to sherrington," *Brain research bulletin*, vol. 50, pp. 95–118, 1999.

[14] S. Carrillo, J. Harkin, L. McDaid, S. Pande, S. Cawley, B. McGinley, and F. Morgan, "Advancing interconnect density for spiking neural network hardware implementations using traffic-aware adaptive network-on-chip routers," *Neural networks*, vol. 33, pp. 42–57, 2012.

[15] G. Liu, P. Camilleri, S. Furber, and J. Garside, "Network traffic exploration on a many-core computing platform: Spinnaker real-time traffic visualiser," in *2015 11th Conference on Ph. D. Research in Microelectronics and Electronics*

(PRIME). 2015 11th Conference on Ph. D. Research in Microelectronics and Electronics (PRIME), 2015, pp. 228–231.

[16] J. Navaridas, L. A. Plana, J. Miguel-Alonso, M. Luján, and S. B. Furber, "Spinnaker: impact of traffic locality, causality and burstiness on the performance of the interconnection network," in *Proceedings of the 7th ACM international conference on Computing frontiers*. Proceedings of the 7th ACM international conference on Computing frontiers, 2010, pp. 11–20.

[17] P. Reinagel and R. C. Reid, "Temporal coding of visual information in the thalamus," *Journal of Neuroscience*, vol. 20, pp. 5392–5400, 2000.

[18] F. Theunissen and J. P. Miller, "Temporal encoding in nervous systems: a rigorous definition," *Journal of computational neuroscience*, vol. 2, pp. 149–162, 1995.

[19] R. C. Decharms and A. Zador, "Neural representation and the cortical code," *Annual review of neuroscience*, vol. 23, pp. 613–647, 2000.

[20] M. Baudry, "Synaptic plasticity and learning and memory: 15 years of progress," *Neurobiology of learning and memory*, vol. 70, pp. 113–118, 1998.

[21] A. Citri and R. C. Malenka, "Synaptic plasticity: multiple forms, functions, and mechanisms," *Neuropsychopharmacology*, vol. 33, pp. 18–41, 2008.

[22] P. J. Sjöström, G. G. Turrigiano, and S. B. Nelson, "Neocortical ltd via coincident activation of presynaptic nmda and cannabinoid receptors," *Neuron*, vol. 39, pp. 641–654, 2003.

[23] R. A. Nicoll and D. Schmitz, "Synaptic plasticity at hippocampal mossy fibre synapses," *Nature Reviews Neuroscience*, vol. 6, pp. 863–876, 2005.

[24] H. Markram, J. Lübke, M. Frotscher, and B. Sakmann, "Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps," *Science*, vol. 275, pp. 213–215, 1997.

[25] S. Song, K. D. Miller, and L. F. Abbott, "Competitive hebbian learning through spike-timing-dependent synaptic plasticity," *Nature neuroscience*, vol. 3, pp. 919–926, 2000.

[26] Q. Yu, H. Li, and K. C. Tan, "Spike timing or rate? neurons learn to make decisions for both through threshold-driven plasticity," *IEEE transactions on cybernetics*, vol. 49, pp. 2178–2189, 2018.

[27] R. V. Florian, "Tempotron-like learning with resume," in *International Conference on Artificial Neural Networks*. International Conference on Artificial Neural Networks, 2008, pp. 368–375.

[28] R. Gütig and H. Sompolinsky, "The tempotron: a neuron that learns spike timing–based decisions," *Nature neuroscience*, vol. 9, pp. 420–428, 2006.

[29] A. Mohemmed, S. Schliebs, S. Matsuda, and N. Kasabov, "Span: Spike pattern association neuron for learning spatio-temporal spike patterns," *International journal of neural systems*, vol. 22, p. 1250012, 2012.

[30] A. Shrestha, K. Ahmed, Y. Wang, and Q. Qiu, "Stable spike-timing dependent plasticity rule for multilayer unsupervised and supervised learning," in *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017 International Joint Conference on Neural Networks (IJCNN), 2017, pp. 1999–2006.

[31] R. Echeveste and C. Gros, "Two-trace model for spike-timing-dependent synaptic plasticity," *Neural computation*, vol. 27, pp. 672–698, 2015.

[32] F. Ponulak and A. Kasiski, "Supervised learning in spiking neural networks with resume: sequence learning, classification, and spike shifting," *Neural computation*, vol. 22, pp. 467–510, 2010.

[33] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in neuroscience*, vol. 10, p. 508, 2016.

[34] E. O. Neftci, C. Augustine, S. Paul, and G. Detorakis, "Event-driven random back-propagation: Enabling neuromorphic deep learning machines," *Frontiers in neuroscience*, vol. 11, p. 324, 2017.

[35] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. S. Maida, "Deep learning in spiking neural networks," *Neural Networks*, vol. 111, pp. 47–63, 2019.

[36] H. Fang, A. Shrestha, Z. Zhao, and Q. Qiu, "Exploiting neuron and synapse filter dynamics in spatial temporal learning of deep spiking neural network," *arXiv preprint arXiv:2003.02944*, 2020.

[37] A. Shrestha, H. Fang, Q. Wu, and Q. Qiu, "Approximating back-propagation for a biologically plausible local learning rule in spiking neural networks," in *Proceedings of the International Conference on Neuromorphic Systems*. Proceedings of the International Conference on Neuromorphic Systems, 2019, pp. 1–8.

[38] C. A. Mead and M. A. Mahowald, "A silicon model of early visual processing," *Neural networks*, vol. 1, pp. 91–97, 1988.

[39] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, pp. 1629–1636, 1990.

[40] C. Mead and M. Ismail, *Analog VLSI implementation of neural systems*. Springer Science & Business Media, 2012, vol. 80.

[41] R. Douglas, M. Mahowald, and C. Mead, "Neuromorphic analogue vlsi," *Annual review of neuroscience*, vol. 18, pp. 255–281, 1995.

[42] C. Mead, *Analog VLSI and Neural Systems*. Addison-Wesley Longman Publishing Co., Inc., 1989.

[43] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, pp. 668–673, 2014.

[44] A. S. Cassidy, P. Merolla, J. V. Arthur, S. K. Esser, B. Jackson, R. Alvarez-Icaza, P. Datta, J. Sawada, T. M. Wong, V. Feldman *et al.*, "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*. The 2013 International Joint Conference on Neural Networks (IJCNN), 2013, pp. 1–10.

[45] A. Amir, P. Datta, W. P. Risk, A. S. Cassidy, J. A. Kusnitz, S. K. Esser, A. Andreopoulos, T. M. Wong, M. Flickner, R. Alvarez-Icaza *et al.*, "Cognitive computing programming paradigm: a corelet language for composing networks of neurosynaptic cores," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*. Neural Networks (IJCNN), The 2013 International Joint Conference on, 2013, pp. 1–10.

[46] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, pp. 157–166, 1994.

[47] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–1780, 1997.

[48] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*. Advances in neural information processing systems, 2014, pp. 3104–3112.

[49] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on, 2015, pp. 3156–3164.

[50] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra, "Draw: A recurrent neural network for image generation," *arXiv preprint arXiv:1502.04623*, 2015.

[51] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko, "Sequence to sequence-video to text," in *Proceedings of the IEEE international conference on computer vision*. Proceedings of the IEEE international conference on computer vision, 2015, pp. 4534–4542.

[52] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in computational neuroscience*, vol. 9, 2015.

[53] P. U. Diehl, B. U. Pedroni, A. Cassidy, P. Merolla, E. Neftci, and G. Zarrella, "Truehappiness: Neuromorphic emotion recognition on truenorth," in *Neural Networks (IJCNN), 2016 International Joint Conference on*. Neural Networks (IJCNN), 2016 International Joint Conference on, 2016, pp. 4278–4285.

[54] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen,

"Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, pp. 699–716, 2014.

[55] M. M. Khan, D. R. Lester, L. A. Plana, A. Rast, X. Jin, E. Painkras, and S. B. Furber, "Spinnaker: mapping neural networks onto a massively-parallel chip multiprocessor," in *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on.* Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on, 2008, pp. 2849–2856.

[56] S. S. Talathi and A. Vartak, "Improving performance of recurrent neural network with relu nonlinearity," *arXiv preprint arXiv:1511.03771*, 2015.

[57] T. M. Breuel, "Benchmarking of lstm networks," *arXiv preprint arXiv:1508.02774*, 2015.

[58] Q. Chen and Q. Qiu, "Real-time anomaly detection for streaming data using burst code on a neurosynaptic processor," in *Proc. Conf. Design, Autom. Test Eur.(DATE).* Proc. Conf. Design, Autom. Test Eur.(DATE), 2017, pp. 1–6.

[59] A. Laudani, G. M. Lozito, F. R. Fulginei, and A. Salvini, "On training efficiency and computational costs of a feed forward neural network: a review," *Computational intelligence and neuroscience*, vol. 2015, p. 83, 2015.

[60] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[61] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

[62] J. Ott, Z. Lin, Y. Zhang, S.-C. Liu, and Y. Bengio, "Recurrent neural networks with limited numerical precision," *arXiv preprint arXiv:1608.06902*, 2016.

[63] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *arXiv preprint arXiv:1609.07061*, 2016.

[64] Q. He, H. Wen, S. Zhou, Y. Wu, C. Yao, X. Zhou, and Y. Zou, "Effective quantization methods for recurrent neural networks," *arXiv preprint arXiv:1611.10176*, 2016.

[65] S.-C. Zhou, Y.-Z. Wang, H. Wen, Q.-Y. He, and Y.-H. Zou, "Balanced quantization: An effective and efficient approach to quantized neural networks," *Journal of Computer Science and Technology*, vol. 32, pp. 667–682, 2017.

[66] G. Hinton, "Neural networks for machine learning, coursera," *URL: http://coursera. org/course/neuralnets*, 2012.

[67] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[68] K. Ahmed, A. Shrestha, Y. Wang, and Q. Qiu, "System design for in-hardware stdp learning and spiking based probablistic inference," in *VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on*. VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on, 2016, pp. 272–277.

[69] X. Li and D. Roth, "Learning question classifiers," in *Proceedings of the 19th international conference on Computational linguistics-Volume 1*. Proceedings of the 19th international conference on Computational linguistics-Volume 1, 2002, pp. 1–7.

[70] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), 2014, pp. 1532–1543.

[71] R. Kempter, W. Gerstner, Van, and J. H. Leo, "Hebbian learning and spiking neurons," *Physical Review E*, vol. 59, p. 4498, 1999.

[72] S. Sengupta, K. S. Gurumoorthy, and A. Banerjee, "Sensitivity analysis for additive stdp rule," *arXiv preprint arXiv:1503.07490*, 2015.

[73] M. Gilson and T. Fukai, "Stability versus neuronal specialization for stdp: long-tail weight distributions solve the dilemma," *PloS one*, vol. 6, p. e25339, 2011.

[74] Van, C. W. R. Mark, G. Q. Bi, and G. G. Turrigiano, "Stable hebbian learning from spike timing-dependent plasticity," *The Journal of Neuroscience*, vol. 20, pp. 8812–8821, 2000.

[75] T. Masquelier and S. J. Thorpe, "Unsupervised learning of visual features through spike timing dependent plasticity," *PLoS Comput Biol*, vol. 3, p. e31, 2007.

[76] B. Nessler, M. Pfeiffer, L. Buesing, and W. Maass, "Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity," *PLoS Comput Biol*, vol. 9, p. e1003037, 2013.

[77] L. F. Abbott and S. B. Nelson, "Synaptic plasticity: taming the beast," *Nature neuroscience*, vol. 3, pp. 1178–1183, 2000.

[78] J. Lisman and N. Spruston, "Questions about stdp as a general model of synaptic plasticity," *Spike-timing dependent plasticity*, vol. 26, p. 53, 2010.

[79] N. Levy, D. Horn, I. Meilijson, and E. Ruppin, "Distributed synchrony in a cell assembly of spiking neurons," *Neural networks*, vol. 14, pp. 815–824, 2001.

[80] F. S. Matias, P. V. Carelli, C. R. Mirasso, and M. Copelli, "Self-organized near-zero-lag synchronization induced by spike-timing dependent plasticity in cortical populations," *PloS one*, vol. 10, p. e0140504, 2015.

[81] K. S. Burbank and G. Kreiman, "Depression-biased reverse plasticity rule is required for stable learning at top-down connections," *PLOS Comput Biol*, vol. 8, p. e1002393, 2012.

[82] J. T. A. Kepecs, "Why neuronal dynamics should control synaptic learning rules," in *Advances in Neural Information Processing Systems 14: Proceedings of the 2001 Neural Information Processing Systems (NIPS) Conference*, vol. 1. Advances in Neural Information Processing Systems 14: Proceedings of the 2001 Neural Information Processing Systems (NIPS) Conference, 2002, p. 285.

[83] J. nosuke Teramae and T. Fukai, "Computational implications of lognormally distributed synaptic weights," *Proceedings of the IEEE*, vol. 102, pp. 500–512, 2014.

[84] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[85] N. A. Lesica, J. Jin, C. Weng, C.-I. Yeh, D. A. Butts, G. B. Stanley, and J.-M. Alonso, "Adaptation to stimulus contrast and correlations during natural visual stimulation," *Neuron*, vol. 55, pp. 479–491, 2007.

[86] S. Afshar, L. George, C. S. Thakur, J. Tapson, A. van Schaik, P. de Chazal, and T. J. Hamilton, "Turn down that noise: Synaptic encoding of afferent snr in a single spiking neuron," *IEEE transactions on biomedical circuits and systems*, vol. 9, pp. 188–196, 2015.

[87] J. C. R. Whittington, T. H. Muller, S. Mark, C. Barry, and T. E. J. Behrens, "Generalisation of structural knowledge in the hippocampal-entorhinal system," *neural information processing systems*, pp. 8493–8504, 2018.

[88] F. Crick, "The recent excitement about neural networks," *Nature*, vol. 337, pp. 129–132, 1989.

[89] S. M. Bohte, J. N. Kok, and J. A. L. Poutré, "Spikeprop: backpropagation for networks of spiking neurons." in *ESANN*. ESANN, 2000, pp. 419–424.

[90] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers in Neuroscience*, vol. 12, 2018.

[91] W. Severa, C. M. Vineyard, R. Dellana, S. J. Verzi, and J. B. Aimone, "Whetstone: A method for training deep artificial neural networks for binary communication," *arXiv preprint arXiv:1810.11521*, 2018.

[92] D. Rasmussen, "Nengodl: Combining deep learning and neuromorphic modelling methods," *arXiv*, vol. 1805.11144, pp. 1–22, 2018. [Online]. Available: http://arxiv.org/abs/1805.11144

[93] D.-H. Lee, S. Zhang, A. Fischer, and Y. Bengio, "Difference target propagation," *european conference on machine learning*, pp. 498–515, 2015.

[94] Y. Bengio, T. Mesnard, A. Fischer, S. Zhang, and Y. Wu, "Stdp as presynaptic activity times rate of change of postsynaptic activity," *arXiv preprint arXiv:1509.05936*, 2015.

[95] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nature Communications*, vol. 7, p. 13276, 2016.

[96] A. Nøkland, "Direct feedback alignment provides learning in deep neural networks," *neural information processing systems*, pp. 1037–1045, 2016.

[97] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1026–1034.

[98] P. OĆonnor and M. Welling, "Deep spiking networks," *arXiv preprint arXiv:1602.08323*, 2016.

[99] Y. Jin, W. Zhang, and P. Li, "Hybrid macro/micro level backpropagation for training deep spiking neural networks," in *Advances in Neural Information Processing Systems*. Advances in Neural Information Processing Systems, 2018, pp. 7005–7015.

[100] H. Xiao, K. Rasul, and R. Vollgraf. (2017, aug) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[101] M. W. Kadous *et al.*, *Temporal classification: Extending the classification paradigm to multivariate time series*. University of New South Wales, 2002.

[102] B. Son, Y. Suh, S. Kim, H. Jung, J.-S. Kim, C. Shin, K. Park, K. Lee, J. Park, J. Woo *et al.*, "4.1 a 640× 480 dynamic vision sensor with a 9μm pixel and 300meps address-event representation," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2017, pp. 66–67.

[103] G. Haessig, F. Galluppi, X. Lagorce, and R. Benosman, "Neuromorphic networks on the spinnaker platform," in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2019, pp. 86–91.

[104] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in *Advances in neural information processing systems*, 2015, pp. 1117–1125.

[105] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[106] DARPA and AFRL. (1995) Sensor data management system website, mstar database. https://www.sdms.afrl.af.mil/index.php?collection=mstar.

[107] J. He, R. Mao, Z. Shao, and F. Zhu, "Incremental learning in online scenario," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 926–13 935.

Vita

Amar Shrestha received the B.S. degree in electrical and electronics engineering from the Department of Electrical and Electronics Engineering in Kathmandu University, Dhulikhel, Nepal, in 2013. He has completed the Ph.D. degree from the Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY, USA. His research interests include neuromorphic computing, deep learning, computer vision and natural language processing.