University of California
Santa Barbara

High-Performance Training Algorithms and Architectural Optimization of Spiking Neural Networks

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Electrical and Computer Engineering

by

Wenrui Zhang

Committee in charge:

Professor Peng Li, Chair
Professor Michael Beyeler
Professor Forrest Brewer
Professor Zheng Zhang

September 2021

The Dissertation of Wenrui Zhang is approved.

Professor Michael Beyeler

Professor Forrest Brewer

Professor Zheng Zhang

Professor Peng Li, Committee Chair

August 2021

High-Performance Training Algorithms and Architectural Optimization of Spiking Neural Networks

by

Wenrui Zhang

In memory of my father
To my parents and my wife for their great support

Acknowledgements

First of all, I would like to thank my advisor, Prof. Peng Li, for his dedicated support and guidance for my Ph.D. research over the past five years. I want to express my greatest gratitude for his tremendous help and tutoring in my research and life. Not only does his comprehensive expertise and insightful suggestions inspire and guide me crucially, but also his everlasting enthusiasm for this area motivates me to challenge myself for broader scopes and greater goals. I truly appreciate him for being his student in this great research group.

Besides, I would also like to thank Prof. Forrest Brewer, Prof. Michael Beyeler, and Prof. Zheng Zhang for their willingness to serve on my committee and for offering insightful suggestions and comments for my work. Their valuable feedback on my qualifying exam greatly benefits my major work for the defense.

It has been a great experience to work with all my colleagues in our Group. Especially, I would like to thank Dr. Yingyezhe Jin for all his help and advice at the beginning of my research on neuromorphic computing. I am also sincerely thankful for Dr. Ya Wang, Dr. Yu Liu, Dr. Changqing Xu, Jeongjun Lee, Richard Boone, Myung Seok Shim, and Renqian Zhang who collaborated with me on one or a few research topics. I would love to greatly thank Dr. Xin Zhan, Hanbin Hu, Dr. Yanxiang Yang, Xiaopei Xu, Tianjun Wu, and Jing Li for their support in my study and life.

Finally, I would like to greatly thank my family who always love me deeply. I'd like to present my sincere thankfulness to my dear parents who teach me and support me through my life. I'd like to express my deepest gratitude to my wife, Nan Du, for her patience, tolerance, and support over these years. Without her help, it would be much harder for my life and study.

Funding Sources and Disclaimer

This dissertation is based upon work supported the National Science Foundation under Award Number 1911067 and 1940761. Any opinions, findings, conclusions or recommendations expressed in this dissertation are those of the author and do not necessarily reflect the views of NSF and their contractors.

This dissertation is also supported by the UCSB ECE Dissertation Fellowship for Summer 2021.

Curriculum Vitæ

Wenrui Zhang

Education

Aug 2021	Ph.D. in Electrical and Computer Engineering (Expected), University of California, Santa Barbara.
June 2015	B.S in Electronic Information Science and Technology, University of Science and Technology of China.

Publications

Zhang, W., & Li, P. (2021). Composing Recurrent Spiking Neural Networks using Locally-Recurrent Motifs and Risk-Mitigating Architectural Optimization. arXiv preprint arXiv:2108.01793

Zhang, W., & Li, P. (2021). Skip-Connected Self-Recurrent Spiking Neural Networks with Joint Intrinsic Parameter and Synaptic Weight Training. Neural Computation, 33(7), 1886-1913.

Zhang, W., & Li, P. (2021). Spiking Neural Networks with Laterally-Inhibited Self-Recurrent Units. In 2021 International Joint Conference on Neural Networks (IJCNN). IEEE.

Zhang, W., & Li, P. (2020). Temporal Spike Sequence Learning via Backpropagation for Deep Spiking Neural Networks. Advances in Neural Information Processing Systems (NeurIPS), 33.

Zhang, W., & Li, P. (2019). Spike-train level backpropagation for training deep recurrent spiking neural networks. Advances in Neural Information Processing Systems (NeurIPS), 32.

Zhang, W., & Li, P. (2019). Information-theoretic intrinsic plasticity for online unsupervised learning in spiking neural networks. Frontiers in neuroscience, 13, 31.

Lee, J., Zhang, W., & Li, P. (2021). Parallel Time Batching: Systolic-Array Acceleration of Sparse Spiking Neural Computation. Under review.

Boone, R., Zhang, W., & Li, P. (2021). Efficient Biologically-Plausible Training of Spiking Neural Networks with Precise Timing. In Proceedings of the International Conference on Neuromorphic Systems.

Yang, Y., Zhang, W., & Li, P. (2021). Backpropagated Neighborhood Aggregation for Accurate Training of Spiking Neural Networks. In International Conference on Machine Learning (ICML),pp. 11852-11862. PMLR.

Lee, J., Chen, J., Zhang, W., & Li, P. (2021). Systolic-Array Spiking Neural Accelerators with Dynamic Heterogeneous Voltage Regulation. In 2021 International Joint Conference on Neural Networks (IJCNN). IEEE.

Lee, J., Zhang, R., Zhang, W., Liu, Y., & Li, P. (2020). Spike-train level direct feedback alignment: sidestepping backpropagation for on-chip training of spiking neural nets. Frontiers in neuroscience, 14, 143.

Xu, C., Zhang, W., Liu, Y., & Li, P. (2020). Boosting Throughput and Efficiency of Hardware Spiking Neural Accelerators Using Time Compression Supporting Multiple Spike Codes. Frontiers in Neuroscience, 14, 104.

Liu, Y., Zhang, W., & Li, P. (2019). Enabling Non-Hebbian Learning in Recurrent Spiking Neural Processors With Hardware-Friendly On-Chip Intrinsic Plasticity. IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 9(3), 465-474.

Jin, Y., Zhang, W., & Li, P. (2018). Hybrid macro/micro level backpropagation for training deep spiking neural networks. Advances in Neural Information Processing Systems (NeurIPS), 31.

Wang, Y., Zhang, W., Li, P., & Gong, J. (2017, June). Convergence-boosted graph partitioning using maximum spanning trees for iterative solution of large linear circuits. In 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC) (pp. 1-6). IEEE.

Shim, M. S., Zhao, C., Li, Y., Zhang, X., Zhang, W., & Li, P. (2019). Robust Deep Multi-Modal Sensor Fusion using Fusion Weight Regularization and Target Learning. arXiv e-prints, arXiv-1901.

Geng, Y., Lu, M., Ding, B., Zhang, W., & Hua, Y. (2019). Improved Stereo Matching based on Convolutional Neural Network. In Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV) (pp. 19-22).

Li, B., Zhang, W., & Lu, M. (2018, December). Multiple Linear Regression Haze-removal Model Based on Dark Channel Prior. In 2018 International Conference on Computational Science and Computational Intelligence (CSCI) (pp. 307-312). IEEE.

Abstract

High-Performance Training Algorithms and Architectural Optimization of Spiking Neural Networks

by

Wenrui Zhang

The spiking neural network (SNN) is an emerging brain-inspired computing paradigm with the more biologically realistic spiking neuron model. As the third generation of artificial neural networks (ANNs), SNNs are theoretically shown to possess greater computational power than the conventional non-spiking ANNs and are well suited for spatio-temporal information processing and implementation on ultra-low power event-driven neuromorphic hardware. This dissertation aims to usher SNNs into the mainstream practice by addressing two key roadblocks: lack of high-performance training algorithms and lack of systematic exploration of computationally-powerful recurrent SNNs.

First, existing SNNs training algorithms suffer from major limitations in terms of learning performance and efficiency. To handle these challenges, we proposed a comprehensive set of solutions including synaptic plasticity (SP) and intrinsic plasticity (IP) to embrace energy-efficient SNNs with high performance. To enable SP based training algorithms, we developed two innovative backpropagation (BP) methods to boost the performance of SNNs. We proposed a Spike-Train level RSNNs Backpropagation (ST-RSBP) algorithm for training deep recurrent SNNs (RSNNs) while addressing the training difficulty introduced by non-differentiability of spiking activation function and improving training efficiency at the spike-train level. To allow for learning temporal sequences with precise timing, we propose a BP method called Temporal Spike Sequence Learning Backpropagation (TSSL-BP), breaking down error backpropagation across two types of

inter/intra-neuron dependencies and precisely capturing the temporal dependencies with ultra-low latency. To train a given SNN using IP, we proposed a method called SpiKL-IP based on a rigorous information-theoretic approach for maintaining homeostasis and shaping the dynamics of neural circuits.

While recurrence is prevalent in the brain, designing practical recurrent spiking neural networks (RSNNs) is challenging due to the intracity introduced by recurrent connections both in time and space. RSNNs are often randomly generated without optimization in the current practice, which however fails to fully exploit the computational potential of RSNNs. We explored and proposed a family of RSNN architectures aiming at building scalable large-scale RSNNs with high performance We first demonstrated a new type of RSNNs called Skip-Connected Self-Recurrent SNN (ScSr-SNN) which contain self-recurrent connections in each recurrent layer and skip connections across non-adjacent layers. It achieves improved performance over existing randomly generated RSNNs. Inspired by the potential of self-recurrent connectivity, we proposed another novel structure called the Laterally-Inhibited Self-Recurrent Unit (LISR), which consists of one excitatory neuron with a self-recurrent connection wired together with an inhibitory neuron through excitatory and inhibitory synapses. SNNs leveraging the LISR as a basic building block significantly improve performance over feedforward SNNs trained by the BP method with similar computational costs. Finally, we developed a systematic optimization-based neural architecture search framework to synthesize high-performance globally-feedforward and locally-recurrent multi-layer RSNNs.

The proposed work achieves the state-of-the-art performances on various image and speech datasets such as MNIST, FashionMNIST, CIFAR10, TI46 and common neuromorphic datasets including NMNIST, NTIDIGITS, DVS-Gesture.

Contents

Chapter 1

Introduction

The biological neocortex is remarkable in perceiving, learning, and adapting to the changing environment. The brain provides an extraordinary reference for building new computing systems with powerful performance and energy-efficient computation. As a brain-inspired computational model, spiking neural networks (SNNs) are considered as the third generation of artificial neural networks (ANNs) with the more biologically realistic spiking neuron model. There is theoretical evidence supporting that SNNs possess greater computational power over traditional ANNs [1]. In ANNs, neurons process continuous-valued inputs with continuous outputs generated through an activation function. However, the spiking neurons mimic the biological neurons' behavior and explicitly model the all-or-none firing spikes across both spatial and temporal domains. The spike-based operational principles of SNNs not only allow information coding based on efficient temporal codes and give rise to promising spatiotemporal computing power but also render energy-efficient VLSI neuromorphic chips such as Spinnaker [2], Neurogrid [3], IBM's TrueNorth [4], Intel's Loihi [5] and so on.

In this chapter, we introduce the two aspects of SNNs, the learning algorithms and network architectures. Most learning algorithms of SNNs can be categorized as either

synaptic plasticity or intrinsic plasticity. We first introduce synaptic plasticity which trains the synaptic weight and is the major workhorse of network training. Then, we introduce intrinsic plasticity which maintains homeostasis and regulates network dynamics by adjusting neuronal parameters. Finally, for the network architecture, we discuss the existing recurrent spiking neural networks (RSNNs) and their limitations.

1.1 Synaptic Plasticity

Inspired by the biological process in the brain [6, 7], spike-timing-dependent plasticity (STDP) is one of the most prevalent synaptic plasticity rules adopted in SNNs. STDP explores the correlation between the firing activities of a pair of presynaptic and postsynaptic neurons and tunes the synaptic weight locally in an unsupervised manner [8]. More specifically, synapses through which a presynaptic spike arrived before (respectively after) a postsynaptic one are reinforced (respectively depressed). Based on the idea of correlating presynaptic spikes and postsynaptic spikes, various supervised/unsupervised methods are proposed by extending the standard STDP rule [9, 10, 11]. The advantages of STDP-based methods are mainly the biological plausibility, hardware efficiency, and easy implementation. However, due to its local and unsupervised nature, the learning performance of STDP is low and cannot fully exploit the powerful computational ability of SNNs.

Backpropagation (BP) is the workhorse for training deep ANNs [12]. Its success in the ANN world has made BP a target of intensive research for SNNs. Various SNNs BP methods have emerged, aiming at attaining the same level of performance [13, 14, 15, 16, 17, 18, 19, 20, 21].

It is possible to train an ANN and then convert it to an SNN [22, 23, 24]. However, this suffers from conversion approximations and gives up the opportunity in exploring

SNNs' temporal learning capability. One of the earliest attempts to bridge the gap between discontinuity of SNNs and BP is the SpikeProp algorithm [13]. However, SpikeProp is restricted to single-spike learning and has not yet been successful in solving real-world tasks.

Recently, training SNNs using BP under a firing rate (or activity level) coded loss function has been shown to deliver competitive performances [14, 15, 20, 17]. Nevertheless, [14] does not consider the temporal correlations of neural activities and deals with spiking discontinuities by treating them as noise. [17] gets around the non-differentiability of spike events by approximating the spiking process via a probability density function of spike state change. [15], [20], and [19] capture the temporal effects by performing backpropagation through time (BPTT) [25]. Among these, [19] adopts a smoothed spiking threshold and a continuous differentiable synaptic model for gradient computation, which is not applicable to widely used spiking neuron models such as the leaky integrate-and-fire (LIF) model. Similar to [14], [15] and [20] compute the error gradient based on the continuous membrane waveforms resulted from smoothing out all spikes. In these approaches, computing the error gradient by smoothing the microscopic membrane waveforms may lose sight of the all-or-none firing characteristics of the SNN that defines the higher-level loss function and lead to inconsistency between the computed gradient and target loss, potentially degrading training performance [16].

Although many appealing results are achieved by these methods, developing SNNs BP training methods that are on a par with the mature BP tools widely available for training ANNs today is a nontrivial problem [26]. Training of SNNs via BP is challenged by two fundamental issues. First, from an algorithmic perspective, the complex neural dynamics in both spatial and temporal domains make the BP process obscure. Moreover, the errors are hard to be precisely backpropagated due to the non-differentiability of discrete spike events. Second, a large number of time steps and step-by-step backprop-

agation are typically required for emulating and training SNNs in time to achieve decent performance, leading to high latency and rendering spike-based computation unscalable to deep architectures.

In this dissertation, we propose two types of BP methods to alleviate the difficulties of training SNNs. First, motivated by the lack of powerful supervised training of general RSNNs and the fact that existing SNN research has limited scope in exploring sophisticated learning architectures like deep RSNNs with multiple feedforward and recurrent layers hybridized together, we proposed a novel Spike-Train level RSNNs Backpropagation (ST-RSBP) algorithm which is applicable to RSNNs with an arbitrary network topology. ST-RSBP is rigorously derived and can handle arbitrary recurrent connections in various RSNNs. While capturing the temporal behavior of the RSNN at the spike-train level, ST-RSBP directly computes the gradient of a rate-coded loss function w.r.t tunable parameters without incurring approximations resulted from altering and smoothing the underlying spiking behaviors.

On the other hand, we propose a new SNNs BP method, called temporal spike sequence learning via BP (TSSL-BP), to learn any target output temporal spiking sequences. TSSL-BP acts as a universal training method for any employed spike codes (rate, temporal, and combinations thereof). To tackle the above difficulties, TSSL-BP breaks down error backpropagation across two types of inter-neuron and intra-neuron dependencies, leading to improved temporal learning precision. It captures the inter-neuron dependencies within an SNN by considering the all-or-none characteristics of firing activities through presynaptic firing times. It also considers the internal evolution of each neuronal state through time, capturing how the intra-neuron dependencies between different firing times of the same presynaptic neuron impact its postsynaptic neuron.

Both of these two methods achieve highly efficient and precise learning on feedforward and recurrent SNNs. For ST-RSBP, it can train RSNNs without costly unfolding

the network through time and performing BP time point by time point, offering faster training and avoiding vanishing/exploding gradients for general RSNNs. For TSSL-BP, it can not only precisely capture the temporal dependencies but also allow ultra-low latency inference and training over only five time steps while achieving excellent accuracy.

1.2 Intrinsic Plasticity

Intrinsic plasticity (IP) is a widely used self-adaptive mechanism that maintains homeostasis and shapes the dynamics of neural circuits. Behaviors of IP have been discovered in brain areas of many species, and IP has been shown to be crucial in shaping the dynamics of neural circuits [27]. In particular, [28] observed the exponentially distributed neuron responses in visual cortical neurons. Such responses may aim at allowing neurons to transmit the maximum amount of information, e.g., measured by the highest entropy, to their outputs with a constrained level of firing activity. Discovered in individual biological neurons, IP changes the excitability of neurons through modification of voltage-gated channels [29].

One of the early biological IP models was explored on the Hodgkin-Huxley type neurons where a number of voltage-gated conductances were considered [30]. Since then, many IP mechanism researches have been conducted for different kinds of artificial neurons. It was shown that an improvement in performance could be obtained when the reservoir of an echo state network (ESN) is adapted using IP such that the neurons in the network can autonomously tune themselves to the desired output distribution [31]. [32] first proposed a mathematical approach to derive an IP rule based on the sigmoid neuron model. This work used the Kullback–Leibler (KL) divergence from an exponential distribution to the actual output firing rate distribution to derive an adaptation rule for the neuron to generate responses following the exponential distribution. Based on

the same principle, an IP rule for hyperbolic tangent neurons was also proposed [31].

Developing effective IP mechanisms for SNNs is a challenging problem. Several empirical IP rules were proposed for SNNs, however, without a rigorous theoretical foundation. The most straightforward idea of IP, also known as dynamic thresholding, is boosting the neuron's activity if the neuron rarely fires and depressing the neuron if it fires too frequently. [33] presented an IP rule by which a spiking neuron's firing threshold voltage changes by a fixed value per update based on whether the neuron fired or not. However, this method cannot precisely determine when and how much the firing threshold voltage should be changed in different situations, and there is no clear understanding of the optimality of the resulting IP behavior. In [34], IP was proposed to control average firing activity and aid synapses to undergo Hebbian modification via STDP depending upon their history of use. [35] presented an approach in which the parameters of the IP rule derived for sigmoid neurons in [36] were empirically mapped to ones for spiking neurons. Since this rule is derived based on the sigmoid neuron model which is significantly different from the spiking neuron model, the property of this IP rule remains elusive when it is applied to adapt the output firing activity of spiking neurons. [37] proposed an IP rule according to the inter-spike-interval (ISI). However, similar to [33], this method only constraints the ISI into a certain range but does not have a rigorous target for adapting the output response. Moreover, [38] proposed another homeostasis mechanism called Non-Hebbian Plasticity which decays synaptic weights based on the activity of postsynaptic neurons. It can control the reservoir neurons' responses to match the firing rate profile of the input and also avoid weight crowding caused by STDP. This non-Hebbian plasticity is based on synaptic plasticity which is different from IP, the intrinsic neuronal plasticity. As discussed in [34], both intrinsic plasticity and non-Hebbian plasticity are homeostatic plasticity mechanisms and observed in biological neurons. They can work together for homeostatic regulation. The mechanism of the dynamic threshold method

is the same as the adaptive LIF (ALIF) neuron model in [20], which adapts the firing rate by adjusting the threshold.

While IP based learning has been attempted for spiking neuron models, the existing IP rules are either ad hoc in nature or not applied particularly towards enabling real-life learning tasks. In this dissertation, we proposed a novel IP method named SpiKL-IP. First, we derive a differentiable transfer function bridging the input current strength and output firing rate when the input level is fixed based on the leaky integrate-and-fire(LIF) model. This transfer function is referred to as the firing-rate transfer function (FR-TF). It shall be noted that FR-TF can correlate the dynamic evolution of the output firing activity measured as averaged firing rate as a function of a received input over a sufficiently long timescale. Next, with this transfer function, we derive an information-theoretical intrinsic plasticity rule for spiking neurons, dubbed *SpiKL-IP*, to minimize the KL-divergence from the exponential distribution to the output firing rate distribution. We further present an online version of the SpiKL-IP rule for minimizing our KL-divergence based loss function in a way analogous to the stochastic gradient descent (SGD) method, which is widely adopted for training deep learning neural networks. Finally, we address two practical issues to ensure the proper operation and robustness of the proposed online IP rule. Among the two issues, it is desirable to apply the proposed IP tuning using the instantaneous input current and the measured output firing rate, allowing seamless consideration of the potentially dynamically changing current input. However, this creates a mismatch to the underlying FR-TF transfer function, which is addressed by making the online IP rule dependent only on the output firing rate such that the LIF model parameters are tuned based on the input/output activities of long timescales. Under various settings, the outputs of targeted spiking neurons converge robustly to the desirable exponential distribution under the proposed SpiKL-IP rule.

1.3 Spiking Neural Networks Architecture

The architectures of spiking neural networks (SNNs) can be categorized as feedforward networks and recurrent networks. Fig. 1.1A shows two SNN architectures often explored in neuroscience: single layer (top) and liquid state machine (bottom) networks. Fig. 1.1B demonstrates the multi-layer feedforward network. Each hidden layer in the feedforward network can be either a dense layer or a convolutional layer. The inter-layer connections can be sparse or fully connected. Several BP methods have demonstrated promising results on feedforward SNNs [14, 15, 16, 17]. Fig. 1.1C presents a more complicated deep recurrent SNNs (RSNNs) with multiple recurrent hidden layers. Unlike the feedforward SNNs which are well studied with much recent effort aiming at improving its performance, the exploration of RSNNs architectures and learning methods is still limited.

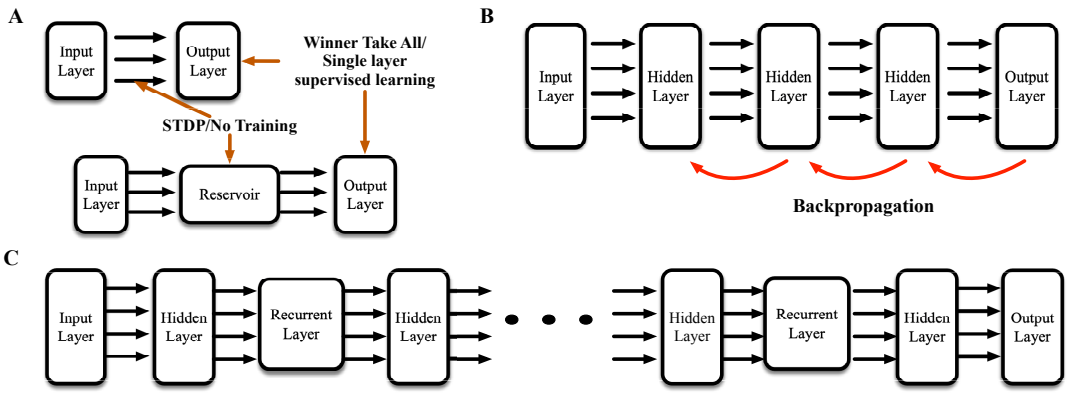

Figure 1.1: Various SNN networks: (A) one layer SNNs and liquid state machine; (B) multi-layer feedforward SNNs; (C) deep hybrid feedforward/recurrent SNNs.

In the brain, recurrent connectivity is indispensable and maintains dynamics, functions, and oscillations of the network [39]. As a brain-inspired computational model, SNNs are well suited for processing spatiotemporal information [1]. In particular, RSNNs can well mimic microcircuits in the biological brain and induce rich behaviors that are critical for memory formation and learning. Recurrence has been explored in conventional non-spiking artificial neural networks (ANNs) in terms of Long Short Term Mem-

ory (LSTM) [40], Echo State Networks (ESN) [41], Deep RNNs [42], Gated Recurrent Units (GRU) [43], and Legendre Memory Units (LMU) [44]. While recurrence presents unique challenges and opportunities under the context of spiking neural networks, RSNNs are yet to be well explored.

Most existing works in RSNNs adopt recurrent layers or reservoirs with randomly generated connections. The Liquid State Machine (LSM) [45] is one of the most widely used RSNN architectures with one or multiple recurrent reservoirs and an output readout layer wired up using feedforward synapses [46, 47, 48]. However, there lack principled approaches for setting up the recurrent connections in a reservoir for which ad-hoc randomly generated wiring patterns are adopted, e.g. a synapse between two neurons is established probabilistically based on a preset connectivity probability. The recurrent weights in the reservoirs are typically either fixed or trained by unsupervised learning methods like spike-timing-dependent plasticity (STDP) [49] while the readout layer is trained by supervision [50, 46, 9]. [20] proposed an architecture called long short-term memory SNNs (LSNNs). The recurrent layer contains a regular spiking portion with both inhibitory and excitatory spiking neurons and an adaptive neural population. [51] demonstrated a new reservoir with multiple groups of excitatory neurons and a central group of inhibitory neurons. However, the recurrent connections in all of these works are randomly generated with certain probabilities. Randomly generated connections without any optimization may hinder learning performance and cannot fully exploit the power of RSNNs.

Systemic RSNN architecture design and optimization remain as an open problem. Current studies suffer from two major limitations: **(1)** lack of powerful supervised training of general RSNNs and limited scope in exploring sophisticated learning architectures like deep RSNNs with multiple feedforward and recurrent layers hybridized together; **(2)** randomly generated recurrent connections without connectivity optimization or specific

design.

In this dissertation, from the algorithmic point of view, we proposed two BP methods, ST-RSBP and TSSL-BP, which are applicable to RSNNs with an arbitrary network topology. From an architectural perspective, we first propose a new RSNN structure called Skip-Connected Self-Recurrent SNN (ScSr-SNN) to offer a simple and structured approach for designing high-performance RSNNs and mitigating the training challenges resulted from random recurrent connections as in the prior works. Recurrence in ScSr-SNN is introduced in a stereotyped manner by adding self-recurrent connections to spiking neurons. The SNNs with self-recurrent connections can realize recurrent behaviors similar to those of more complex RSNNs while the error gradients can be more straightforwardly calculated due to the mostly feedforward nature of the network. The network dynamics is enriched by skip connections between nonadjacent layers.

Then, we proposed a novel recurrent structure for SNNs called Laterally-Inhibited Self-Recurrent Unit (LISR) which consists of one excitatory neuron with a self-recurrent connection wired together with an inhibitory neuron through excitatory and inhibitory synapses. The self-recurrent connection of the excitatory neuron mitigates the information loss caused by the firing-and-resetting mechanism and maintains the long-term neuronal memory. The lateral inhibition from the inhibitory neuron to the corresponding excitatory neuron, on the one hand, adjusts the firing activity of the latter. On the other hand, it plays as a forget gate to clear the memory of the excitatory neuron. This proposed connectivity not only offers a structured approach for designing high-performance RSNNs but also mitigates the training challenges resulting from random recurrent connections as in the existing works.

Finally, to enable the systemic design of large RSNNs, we proposed a novel layer architecture and an alternating two-step architectural optimization method. Firstly, we compose RSNNs based on a layer architecture called Sparsely-Connected Recurrent Mo-

tif Layer (SC-ML) that consists of multiple small recurrent motifs wired together by sparse lateral connections. The small size of the motifs and sparse inter-motif connectivity leads to an RSNN architecture scalable to large network sizes. We argue that composing RSNNs based on well-optimized building blocks small in size, or recurrent motifs, can lead to an architectural solution scalable to large networks while achieving high performance. Secondly, we propose a method called Hybrid Risk-Mitigating Architectural Search (HRMAS) to systematically optimize the topology of the proposed recurrent motifs and SC-ML layer architecture. HRMAS is an alternating two-step optimization process by which we mitigate the risk of network instability and performance degradation caused by architectural change by introducing a novel biologically-inspired "self-repairing" mechanism through intrinsic plasticity. The intrinsic plasticity is introduced to the second step of each HRMAS iteration and acts as unsupervised fast self-adaption to structural and synaptic weight modifications introduced by the first step during the RSNN architectural "evolution". To the best of the our knowledge, this is the first work that performs systematic architectural optimization of RSNNs.

We also demonstrate that both the proposed architectural optimization method and our manually design RSNN architectures can significantly outperform the best-reported performance obtained from the existing works over various speech and neuromorphic datasets.

1.4 Outline

The rest of this dissertation is organized as follows: In Chapter 2, we introduce the background knowledge such as the spiking neuron model, datasets, and so on. In Chapter 3, we propose the new IP rule named SpiKL-IP to address the theoretical and practical limitations of the existing works. In Chapter 4, we present a novel Spike-Train level

RSNNs Backpropagation (ST-RSBP) algorithm for training deep RSNNs. In Chapter 5, we demonstrate a novel Temporal Spike Sequence Learning Backpropagation (TSSL-BP) method for training deep SNNs, which breaks down error backpropagation across two types of inter-neuron and intra-neuron dependencies and leads to improved temporal learning precision. In Chapter 6, we propose a new type of RSNNs called Skip-Connected Self-Recurrent SNN (ScSr-SNN). Then, we further demonstrate a novel recurrent structure called the Laterally-Inhibited Self-Recurrent Unit (LISR), which consists of one excitatory neuron with a self-recurrent connection wired together with an inhibitory neuron through excitatory and inhibitory synapses. In Chapter 7, we present a novel layer architecture called Sparsely-Connected Recurrent Motif Layer (SC-ML) that consists of multiple small recurrent motifs wired together by sparse lateral connections. Then, we present a method called Hybrid Risk-Mitigating Architectural Search (HRMAS) to systematically optimize the topology of the proposed recurrent motifs and SC-ML layer architecture. Chapter 8 concludes this dissertation and provides discussions of potential future work.

Chapter 2

Background

2.1 Spiking Neuron Model

Spiking neural networks (SNNs) employ more biologically plausible spiking neuron models compared to traditional artificial neural networks (ANNs). In this dissertation, we adopt the leaky integrate-and-fire (LIF) neuron model and first order synaptic model [52].

Consider the input spike train from presynaptic neuron j to neuron i: $s_j(t) = \sum_{t_j^{(f)}} \delta(t - t_j^{(f)})$, where $t_j^{(f)}$ denotes a particular firing time of presynaptic neuron j. The incoming spikes are converted into an (unweighted) postsynaptic current (PSC) $a_j(t)$ through a synaptic model. The neuronal membrane voltage $u_i(t)$ at time t for neuron i is given by

$$\tau_m \frac{du_i(t)}{dt} = -u_i(t) + R \sum_j w_{ij} a_j(t) + \eta_i(t), \tag{2.1}$$

where R and τ_m are the effective leaky resistance and time constant of the membrane, w_{ij} is the synaptic weight from presynaptic neuron j to neuron i, $a_j(t)$ is the PSC induced by the spikes from presynaptic neuron j, and $\eta_i(t)$ denotes the reset function.

The PSC and the reset function can be written as

$$a_j(t) = (\epsilon * s_j)(t), \qquad \eta_i(t) = (\nu * s_i)(t), \tag{2.2}$$

where $\epsilon(\cdot)$ and $\nu(\cdot)$ are the spike response and reset kernel, respectively. We adopt a first order synaptic model as the spike response kernel which is expressed as:

$$\tau_s \frac{a_j(t)}{dt} = -a_j(t) + s_j(t), \tag{2.3}$$

where τ_s is the synaptic time constant. The reset kernel reduces the membrane potential by a certain amount Δ_R, where Δ_R is equal to the firing threshold right after the neuron fires. Considering the discrete time steps simulation, we use the fixed-step first-order Euler method to discretize (2.1) to

$$u_i[t] = (1 - \frac{1}{\tau_m})u_i[t-1] + \sum_j w_{ij} a_j[t] + \eta_i[t]. \tag{2.4}$$

The ratio of R and τ_m is absorbed into the synaptic weight. The reset function $\eta_i[t]$ represents the firing-and-resetting mechanism of the neuron model. Moreover, the firing output of the neuron is expressed as

$$s_i[t] = H\left(u_i[t] - V_{th}\right), \tag{2.5}$$

where V_{th} is the firing threshold and $H(\cdot)$ is the Heaviside step function.

2.2 Datasets

To demonstrate the effectiveness of the proposed methods, we adopt various types of datasets for the experiments including images, speeches, neuromorphic images, neuromorphic speeches, and neuromorphic videos. In this section, we introduce all the datasets and their preprocessing steps applied in this dissertation.

2.2.1 MNIST

The MNIST [53] digit dataset consists of $60,000$ samples for training and $10,000$ for testing, each of which is a 28×28 grayscale image. For the fully-connected feedforward networks, the inputs are encoded from each $28 \times 28 \times N_t$ image into a 2D $784 \times N_t$ matrix where N_t is the simulation time steps. Each input sample is normalized to the same mean and standard deviation. In the experiments of the ST-RSBP, we convert each pixel value of an MNIST image into a spike train using Poisson sampling based on which the probability of spike generation is proportional to the pixel intensity. For all other experiments, each pixel value of an MNIST image is converted into a real-valued input current.

2.2.2 Fashion-MNIST

The Fashion-MNIST dataset [54] contains 28x28 grey-scale images of clothing items, meant to serve as a much more difficult drop-in replacement for the well-known MNIST dataset. It contains 60,000 training examples and 10,000 testing examples with each image falling under one of the 10 classes. Similar to MNIST, for the ST-RSBP, using Poisson sampling, we encode each 28×28 image into a 2D $784 \times L$ binary matrix, where $L = 400$ represents the duration of each spike sequence in ms, and a 1 in the matrix represents a spike. However, for all other methods, the pixel values are directly used as

the real-valued input current at each time step.

2.2.3 CIFAR10

The CIFAR-10 [55] dataset contains $60,000$ 32×32 color images in 10 different types of objects. There are $50,000$ training images and $10,000$ testing images. The pixel intensity of each channel is converted into a real-valued input. Similar to what is commonly adopted for preprocessing, the dataset is normalized, and random cropping and horizontal flipping are applied for data augmentation. In addition, dropout layers with a rate of 0.2 are also applied during the training of CIFAR10.

2.2.4 TI46-Alpha

TI46 speech corpus [56] contains spoken English alphabets and digits audios from 16 speakers. In our experiments, the full alphabets subset of the TI46 is used. We name this subset TI46-Alpha in the rest of the paper. The TI46-Alpha has 4142 and 6628 spoken English examples in 26 classes for training and testing, respectively. The continuous temporal speech waveforms are preprocessed by Lyon's ear model [57]. The sample rate of this dataset is 12.5 kHz. The decimation factor of Lyon's ear model is 125. The Matlab implementation of Lyon's ear model is available online [58]. Each sample is encoded into 78 channels. In our experiments, the preprocessed real-value intensities are directly applied as the inputs.

2.2.5 N-MNIST

The N-MNIST dataset [59] is a neuromorphic version of the MNIST dataset generated by tilting a Dynamic Version Sensor (DVS) [60] in front of static digit images on a computer monitor. The movement induced pixel intensity changes at each location are

encoded as spike trains. Since the intensity can either increase or decrease, two kinds of ON and OFF spike events are recorded. Due to the relative shifts of each image, an image size of 34×34 is produced. Each sample of the N-MNIST is a spatio-temporal pattern with $34 \times 34 \times 2$ spike sequences lasting for $300ms$ with the resolution of $1us$. It means there are 300000 time steps in the original N-MNIST dataset. In our experiments, we reduce the time resolution of the N-MNIST samples by 3000 times to speed up the simulation. Therefore, the preprocessed samples only have about 100 time steps. We determine that a channel has a spike at a certain time step of the preprocessed sample if there's at least one spike among the corresponding 3000 time steps of the original sample.

2.2.6 N-TIDIGITS

The N-TIDIGITS [61] is the neuromorphic version of the speech dataset Tidigits [62]. The original audios are processed by a 64-channel CochleaAMS1b sensor and recorded as the spike responses. The dataset contains both single-digit samples and connected-digit sequences with a vocabulary consisting of 11 digits including "oh," "zero" and the digits "1" to "9". In the experiments, only the single-digit samples are used. In total, there are 55 male and 56 female speakers with 2475 single-digit samples for training and the same number of samples for testing. In the original dataset, each sample has 64 input channels and takes about 0.9 s. To speed up the simulation, each sample is reduced to 300 time steps by compressing the time resolution from 1 us to 3 ms. During the compression, a channel has a spike at a certain time step in the preprocessed sample if it contains at least one spike in the corresponding time window of the original sample.

2.2.7 DVS-Gesture

The DVS-Gesture dataset [63] consists of recordings of 29 different individuals (subjects) performing hand and arm gestures. The spikes are generated from natural motion. There are 122 trials in total. Each trial contains the recording for one subject by a dynamic vision sensor (DVS) camera under one of the three different lighting conditions. In each trial, 11 hand and arm gestures of the subject are recorded. We follow the same preprocessing procedures in [17]. Samples from the first 23 subjects are used for training and the other 6 subjects for testing. During preprocessing, the trials are separated into individual actions (gestures). The task is to classify the action sequence video into an action label. Each action (sample) lasts for about 6 s. In addition, two channels with 128×128 pixels in each channel are recorded. In the experiments, only the first 1.5 s of action video for each sample are used. We compress the temporal resolution to 5 ms which means it takes 300 time steps for each sample. Similar to the preprocessing of N-TIDIGITS, the input pixel has a spike at a certain time step in the preprocessed sample if it contains at least one spike in the corresponding 5 ms time window of the original sample.

Chapter 3

Information-Theoretic Intrinsic Plasticity

As a self-adaptive mechanism, intrinsic plasticity (IP) plays an important role in maintaining homeostasis and shaping the dynamics of neural circuits. From a computational point of view, IP has the potential in enabling promising non-Hebbian learning in artificial neural networks. While IP-based learning has been attempted for spiking neuron models, the existing IP rules are ad hoc in nature and the practical success of their application has not been demonstrated particularly towards enabling real-life learning tasks. In this chapter, we aim to address the theoretical and practical limitations of the existing works by proposing a new IP rule named SpiKL-IP. SpiKL-IP is developed based on a rigorous information-theoretic approach where the target of IP tuning is to maximize the entropy of the output firing rate distribution of each spiking neuron. This goal is achieved by tuning the output firing rate distribution towards a targeted optimal exponential distribution. Operated on a proposed firing-rate transfer function, SpiKL-IP adapts the intrinsic parameters of a spiking neuron while minimizing the KL-divergence from the targeted exponential distribution to the actual output firing rate distribution.

In addition, SpiKL-IP can robustly operate in an online manner under complex input and network settings.

3.1 Firing-Rate Transfer Function

From the LIF model in (2.1), we denote the total input to the neuron as x. Then, the LIF neuron model can be expressed as

$$\tau_m \frac{du(t)}{dt} = -u(t) + Rx. \tag{3.1}$$

Once the membrane potential $u(t)$ exceeds the firing threshold V_{th}, the neuron generates a spike and the membrane potential is reset to the resting potential, which is 0 V in our case. In this chapter, a refractory period of duration t_r is also considered after a spike is generated during which $u(t)$ is maintained at 0 V.

Before presenting the proposed SpiKL-IP rule for spiking neurons, we shall first establish the relationship between the input current and the resulting output firing rate. This relationship is not obvious since the response is in the form of spikes and it depends on the cumulative effects of all the past input. As a result, it is difficult to evaluate the output firing rate of spiking neurons at each time point under a varying input. We deal with this difficulty by deriving the proposed firing-rate transfer function (FR-TF) where the input is assumed to be constant. In other words, FR-TF correlates the dynamic evolution of the output firing activity measured as averaged firing rate as a function of a received input over a sufficiently long time scale.

Now, we assume that the input current x_0 is constant and an output spike has occurred at time $t = t^{(1)}$. By integrating (3.1) with the initial condition that $u\left(t^{(1)}\right) = 0$, it gives

the expression of the membrane potential before the next output spike

$$u(t) = Rx_0 \left[1 - exp\left(-\frac{t - t^{(1)} - t_r}{\tau_m}\right)\right], \qquad t > t^{(1)} + t_r. \tag{3.2}$$

Assume that the next output spike occurs at $t = t^{(2)}$, i.e. $u(t^{(2)}) = V_{th}$. Let the interspike interval be $T_{isi} = t^{(2)} - t^{(1)}$, we get

$$V_{th} = Rx_0 \left[1 - exp\left(-\frac{T_{isi} - t_r}{\tau_m}\right)\right] \tag{3.3}$$

which leads to

$$T_{isi} = t_r + \tau_m ln \frac{Rx_0}{Rx_0 - V_{th}}, \qquad Rx_0 > V_{th}. \tag{3.4}$$

where the constraint of $Rx_0 > V_{th}$ comes from the fact that only when the constant input current is sufficient large, the neuron can generate spikes. Since both the input x_0 and T_{isi} are constant, the mean output firing rate of the spiking neuron is given by

$$y = \frac{1}{T_{isi}} = \frac{1}{t_r + \tau_m ln \frac{Rx_0}{Rx_0 - V_{th}}}, \qquad Rx_0 > V_{th}. \tag{3.5}$$

In this way, we obtain the transfer function of spiking neurons under the condition that it has constant input so that this relation between input and output can be used in the deriving process. Since this function can only represent spiking neurons with a fixed input, in order to distinguish the spiking neurons and this transfer function, when referring to firing-rate model neurons, it means the neurons with this firing-rate transfer function (3.5).

Figure 3.1 shows two tuning curves of the firing-rate transfer function where the input current level is swept while either the leaky resistance R or the membrane time constant τ_m is held at a specific value. As shown in Figure 3.1A, changing R while fixing τ_m

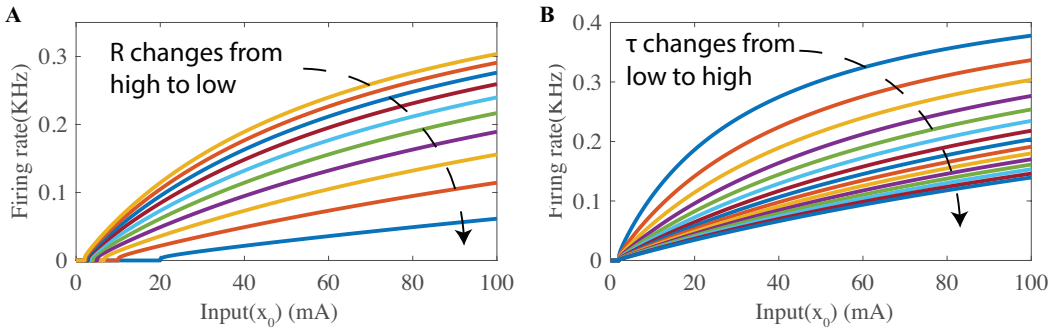

Figure 3.1: The firing-rate transfer function (FR-TF): **(A)** as a function of the leaky resistance R, and **(B)** as a function of the membrane time constant τ_m.

modifies both the bias and curvature of the tuning curve. Figure 3.1B illustrates that τ_m controls the curvature of the tuning curve when R is fixed. In the following part, the proposed SpiKL-IP Rule is based on tuning R and τ_m.

3.2 The Basic SpiKL-IP

Based on the presented firing-rate transfer function (3.5), we now take an information-theoretical approach to derive the SpiKL-IP rule to minimize the KL-divergence from the exponential distribution to the output firing rate distribution.

We consider the information processing of a given spiking neuron as it receives stimuli from external inputs or other neurons in the same network over a dataset, mimicking part of the lifespan of the biological counterpart. We define the input and output firing rate probability distributions for each spiking neuron in the following way. As shown in Figure 3.2, the input current level X varies across different time points, it forms an input probability distribution over the course of the entire process denoted by $f_x(x)$. Accordingly, the output firing rate Y varies over time and forms an output probability distribution denoted by $f_y(y)$.

The goal of the SpiKL-IP rule is to obtain an approximately exponential distribution

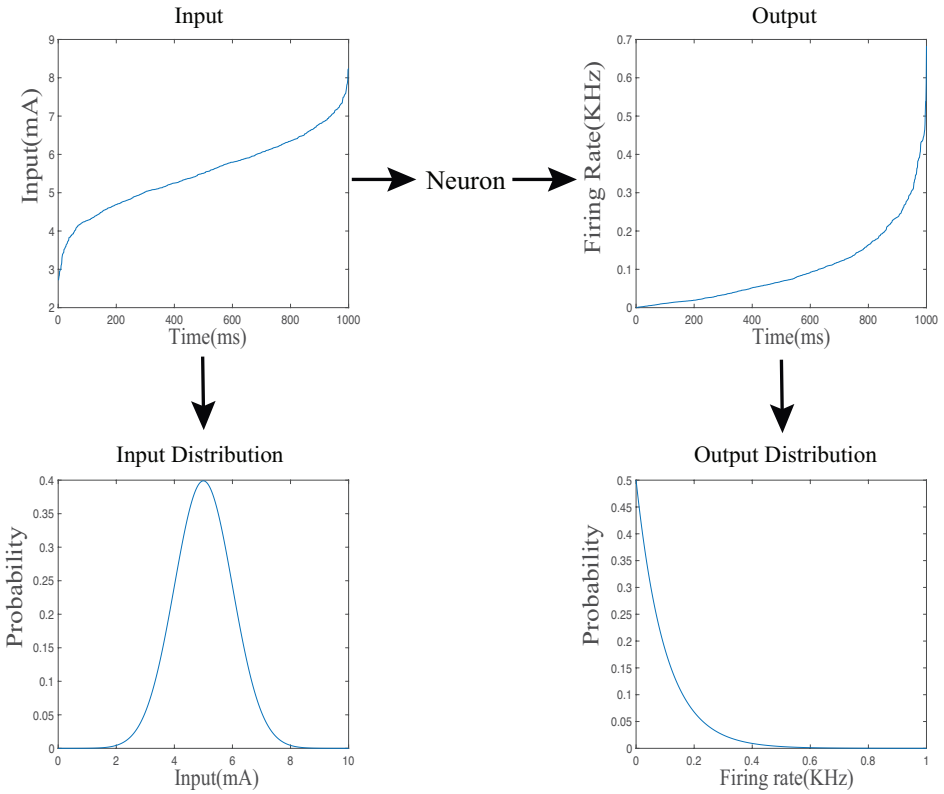

Figure 3.2: The mapping from the input current distribution to the output firing rate distributing of a neuron.

of the output firing rate at a fixed level of metabolic costs. In a biological perspective, exponential distributions of the output firing rate have been observed in mammalian visual cortical neurons responding to natural scenes and allow the neuron to transmit the maximum amount of information given a fixed level of metabolic costs [28].

From an information-theoretic point of view, [64] argued that a neuron may self-adapt to maximize the mutual information of the input X and the output Y, a measure for the amount of information about the input obtained from the output, or vice versa

$$I(Y, X) = H(Y) - H(Y|X), \tag{3.6}$$

where $H(Y)$ is the entropy of the output while $H(Y|X)$ represents the amount of entropy

(uncertainty) of output that does not come from the input. Since $H(Y|X)$ is the intrinsic uncertainty of Y which does not depend on neural parameters and the input, maximizing $I(Y, X)$ is equivalent to maximizing $H(Y)$. To this end, it is instrumental to note when the mean of the distribution is kept constant, the exponential distribution corresponds to the largest entropy among all probability distributions of a non-negative random variable. This leads to the conclusion that the exponential distribution with a targeted mean shall be the optimal distribution for the output firing rate, where the mean specifies the practical constraint on energy expenditure. The exponential distribution is given by

$$f(x) = \mu exp(-\mu x), \qquad x >= 0, \tag{3.7}$$

where μ is the mean of the distribution.

Inspired by the IP rule for sigmoid neurons in [32], we derive the SpiKL-IP rule for spiking neurons while minimizing the KL-divergence from a targeted exponential distribution to the actual output firing rate distribution, where Kullback–Leibler divergence (KL-divergence) is used as a difference measure as follows

$$\begin{aligned} D = d\left(f_y(y)||f_{exp}\right) &= \int f_y(y) log\left(\frac{f_y(y)}{\frac{1}{\mu}exp(\frac{-y}{\mu})}\right) dy \\ &= \int f_y(y) log(f_y(y)) dy + \int f_y(y)\left(\frac{y}{\mu}\right) dy + \int f_y(y) log\mu dy, \end{aligned} \tag{3.8}$$

where y and $f_y(y)$ denote the output and the output firing rate distribution, respectively, and μ is the mean value of the targeted exponential distribution. The smaller the KL-divergence D is, the closer the exponential distribution is to the output distribution. In (3.8), since $\int f_y(y)dy = 1$ the third integral evaluates to a fixed value of $log\mu$. Minimizing KL-Divergence D by adapting R and τ_m reduces to minimize the first two integrals, giving

rise to the following loss function

$$\begin{aligned} L &= \int f_y(y) log(f_y(y)) dy + \int f_y(y) \left(\frac{y}{\mu}\right) dy \\ &= E\left[log(f_y(Y)) + \frac{Y}{\mu}\right]. \end{aligned} \tag{3.9}$$

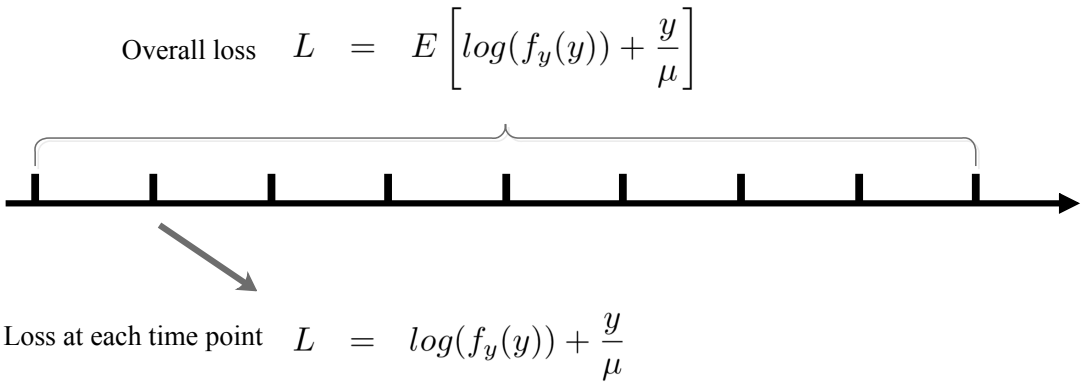

Figure 3.3: Online SpiKL-IP learning: minimization of the KL divergence at each time point during the training process.

Note that (3.9) is in terms of an expectation over the entire output distribution. Now, we convert (3.9) into an online form that is analogous to the stochastic gradient descent method with a batch size of one. To make SpiKL-IP amenable for online training, using a proper stepsize we discretize the entire training process into multiple small time intervals each in between two adjacent time points as shown in Fig 3.3. The input level to the spiking neuron at each time point is considered as an individual observation or training example. In this way, the adjustment of the tunable parameters is not delayed until the output firing rate distribution is collected after the entire dataset is applied to the neuron (or neural network). Instead, these parameters are adjusted as the neuron experiences a given input example at each time point in an online manner. To do this, the following loss function that corresponds to the received input example is minimized at each time

point t

$$L(t) = log(f_y(y(t))) + \frac{y(t)}{\mu}, \tag{3.10}$$

where $y(t)$ denotes the output firing rate Y observed at time t. From now on, we drop the explicit dependency of $y(t)$ and $x(t)$ (observed input level at t) on t for notational simplicity. Recognizing that the output probability distribution relates to the input counterpart by [65]

$$f_y(y) = \frac{f_x(x)}{\frac{\partial y}{\partial x}} \tag{3.11}$$

and substituting it into (3.10) leads to

$$L(t) = log(f_x(x)) - log\left(\frac{\partial y}{\partial x}\right) + \frac{y}{\mu}, \tag{3.12}$$

which can be further simplified to

$$\hat{L}(t) = -log\left(\frac{\partial y}{\partial x}\right) + \frac{y}{\mu}, \tag{3.13}$$

as $log(f_x(x))$ is a function of the input probability distribution and does not depend on R and τ_m.

The online SpiKL-PI rule is based upon the partial derivatives of (3.10) with respect to x, R and τ_m. We first shall compute the derivatives of the output firing rate $y(t)$ with respect to x, R, τ_m. We make use of the firing rate transfer function (3.5) whose application at each time point t is justified if the input $x(t)$ changes slowly with respect

to the chosen stepsize and the averaged output firing rate measure is used, and obtain

$$\begin{aligned}\frac{\partial y}{\partial x} &= \frac{y^2 \tau_m V_{th}}{x(Rx - V_{th})} \\ \frac{\partial y}{\partial R} &= \frac{y^2 \tau_m V_{th}}{R(Rx - V_{th})} \\ \frac{\partial y}{\partial \tau_m} &= \frac{t_r y^2 - y}{\tau_m}.\end{aligned} \tag{3.14}$$

Taking (3.14) into account, the partial derivatives of the loss function (3.10) with respect to R and τ_m are found to be

$$\begin{aligned}\frac{\partial L}{\partial R} &= \frac{\partial}{\partial R}\left(-log\left(\frac{\partial y}{\partial x}\right) + \frac{y}{\mu}\right) = \frac{\partial}{\partial R}\left(-(2log(y) - log(Rx - V_{th})) + \frac{y}{\mu}\right) \\ &= \frac{\left(\frac{y^2}{\mu} - 2y\right)\tau_m V_{th} + Rx}{R(Rx - V_{th})}\end{aligned} \tag{3.15}$$

and

$$\begin{aligned}\frac{\partial L}{\partial \tau_m} &= \frac{\partial}{\partial \tau_m}\left(-log\left(\frac{\partial y}{\partial x}\right) + \frac{y}{\mu}\right) = \frac{\partial}{\partial \tau_m}\left(-(2log(y) + log\tau_m) + \frac{y}{\mu}\right) \\ &= \frac{1 + \frac{1}{\mu}(t_r y^2 - y) - 2t_r y}{\tau_m},\end{aligned} \tag{3.16}$$

respectively, which gives the following online IP rule

$$R = R - \eta_1 \frac{\partial L}{\partial R} = R + \eta_1 \frac{\left(2y - \frac{y^2}{\mu}\right)\tau_m V_{th} - Rx}{R(Rx - V_{th})}, \qquad Rx > V_{th} \tag{3.17}$$

and

$$\tau_m = \tau_m - \eta_2 \frac{\partial L}{\partial \tau_m} = \tau_m + \eta_2 \frac{2t_r y - 1 - \frac{1}{\mu}(t_r y^2 - y)}{\tau_m}, \qquad Rx > V_{th}. \tag{3.18}$$

where η_1 and η_2 are learning rates, μ the constant value depending on the desired mean of the output firing rate. The condition $Rx > V_{th}$ comes from the transfer function (3.5).

3.3 Practical Considerations and Final SpiKL-IP

While (3.17, 3.18) has the key elements of the proposed online IP rule, its direct implementation, however, has been experimentally shown to be unsuccessful, i.e. it can neither train spiking neurons to generate output firing rates following the exponential distribution nor improve SNN learning performance for real-world classification tasks. The problem has to do with the fact that one underlying assumption behind the firing rate transfer function (FR-TF) (3.5) and hence the IP rule (3.17, 3.18) is that the input current is constant or changes over a sufficiently slow timescale. However, in a practical setting, the total postsynaptic input received by a spiking neuron does vary in time and the rate of change depends on the frequency of firing activities of its presynaptic neurons. With the internal dynamics, the output firing level of a spiking neuron cannot immediately follow the instantaneous current input, e.g. it is possible that the output firing rate is still low while the input current has increased to a high level. As a result, the assumption on the input current is rather constraining and its violation leads to ineffectiveness of IP tuning.

On the other hand, it is worth noting that the FR-TF captures the correlation between the average input current and output firing rate over a long time scale. In the meantime, the proposed IP rule aims to adapt spiking neurons to produce a desirable probability distribution of the output firing rate. In other words, the objective is not to tune each instance of the output firing rate. Instead, it is to achieve desirable collective characteristics of the output firing rate measured by an exponential distribution. In some sense, the FR-TF correlates the input and output correspondence in a way that is

meaningful for the objective of online IP tuning.

To find a solution to the above difficulty, we remove the dependency on the instantaneous input current from the IP rule of (3.17, 3.18) by substituting the input x using the output firing rate y with the transfer function (3.5). More specifically, a new variable W is defined by $W = Rx - V_{th}$, which can be expressed using y based on (3.5) as

$$W = \frac{V_{th}}{e^{\left(\frac{1}{\tau_m}\left(\frac{1}{y} - t_r\right)\right)} - 1}. \tag{3.19}$$

Making use of (3.19), (3.17, 3.18) is converted to a form which only depends on y

$$\begin{aligned} R &= R + \eta_1 \frac{2y\tau_m V_{th} - W - V_{th} - \frac{1}{\mu}\tau_m V_{th} y^2}{RW}, \qquad y > 0. \\ \tau_m &= \tau_m + \eta_2 \frac{2t_r y - 1 - \frac{1}{\mu}(t_r y^2 - y)}{\tau_m}. \end{aligned} \tag{3.20}$$

As can be seen, the rule in (3.20) adjusts the two parameters only based on the output firing rate y. Substituting the instantaneous value of x by the firing rate y based on the firing rate transfer function effectively operates the IP rule based on the averaged input/output characteristics over a longer time scale.

Note that the condition that $Rx > V_{th}$ in (3.17, 3.18) is changed to an equivalent form of $y > 0$ in (3.20). A closer examination of Figure 3.1 shows that the firing rate transfer functions are not differentiable around $y = 0$ ($Rx = V_{th}$). Interpreting differently, the proposed IP tuning can operate only when the output firing rate is nonzero. To further improve the robustness of the proposed IP rule, the tuning in (3.20) is only activated when $y > \delta$ with δ being small such as 1 Hz. When $y \leq \delta$, R and τ_m are decreased and increased, respectively, to bring up the output firing activity. Putting everything

together, the final SpiKL-IP rule is

$$
\begin{aligned}
R = \{ & \begin{array}{ll} R + \eta_1 \frac{2y\tau_m V_{th} - W - V_{th} - \frac{1}{\mu}\tau_m V_{th} y^2}{RW}, & y > \delta \\ R + \eta_1 \alpha_1, & y \leq \delta \end{array} \\
\tau_m = \{ & \begin{array}{ll} \tau_m + \eta_2 \frac{2t_r y - 1 - \frac{1}{\mu}(t_r y^2 - y)}{\tau_m}, & y > \delta \\ \tau_m - \eta_2 \alpha_2, & y \leq \delta \end{array}
\end{aligned}
\tag{3.21}
$$

where α_1 and α_2 are chosen to be small.

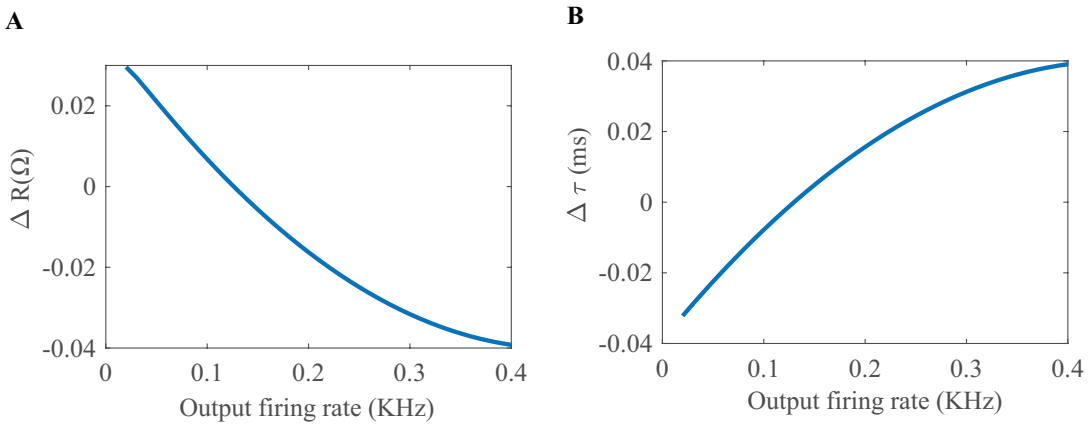

Figure 3.4: Tuning characteristics of one-time application of SpiKL-IP at different output firing rate levels starting from a chosen combination of R and τ_m values R and τ_m: (A) Tuning of the leaky resistance R, and (B) tuning of the membrane time constant τ_m.

To provide an intuitive understanding of the proposed SpiKL-IP rule, Figure 3.4 shows how R and τ_m are altered by the one-time application of SpiKL-IP at different output firing rate levels starting from a chosen combination of R and τ_m values.

3.4 Experiments and Results

To demonstrate the mechanisms and performances of the proposed SpiKL-IP rule, we conduct three types of experiments by applying SpiKL-IP to a single neuron as well as a

group of spiking neurons as part of a neural network. First, we show that when applied to a single neuron whose behavior is governed by the firing-rate transfer function (3.5) the proposed rule can tune the neuron to produce the targeted exponential distribution of the output firing rate even under a time-varying input. Then, we apply SpiKL-IP to a single spiking neuron as well as a group of spiking neurons to demonstrate that our rule can robustly produce the desired output firing distribution in all tested situations even although it is derived from the FR-TF which is based on assumption that the input is constant. Finally, we demonstrate the significant performance boosts achieved by SpiKL-IP when applied to real-world speech and image classification tasks. Furthermore, we compare SpiKL-IP with two existing IP rules for spiking neurons [33, 35]. In this article, we name the IP rule in [33] as the Voltage-Threshold IP rule and one in [35] as the RC IP rule.

The following experimental setups are adopted in the experiments of this section. We simulate the continuous-time LIF model (2.1) using a fixed discretization time step of $1ms$ according to which all neuronal activities are evaluated in lockstep. To measure the firing rate of each spiking neuron as a continuous-valued quantity over time under a constant of varying input, we use the intracellular calcium concentration $C_{cal}(t)$ as a good indicator of the averaged firing activity over a chosen time scale

$$\frac{dC_{cal}(t)}{dt} = -\frac{C_{cal}(t)}{\tau_{cal}} + \sum_i \delta(t - t_i), \tag{3.22}$$

where τ_{cal} is the time constant and the output firing spikes are presented by a series of Dirac delta functions. According to (3.22), the calcium concentration increases by one unit when an output spike is generated, and decays with a time constant τ_{cal} [66]. The

time-varying output firing rate is measured using the normalized calcium concentration

$$y(t) = \frac{C_{cal}(t)}{\tau_{cal}}. \tag{3.23}$$

3.4.1 Single Neurons Modeled by FR-TF

We apply the proposed SpiKL-IP rule to a single neuron modeled based on the firing-rate transfer function (3.5). The parameters of the neuron and SpiKL-IP are are set as follows: $V_{th} = 20mV$, $t_r = 2ms$, and $\mu = 0.2KHz$. In addition, the tuning ranges for R and τ_m are set to $[1\Omega, 1024\Omega]$ and $[1ms, 1,024ms]$ with R and τ_m initialized to 64Ω and $64ms$, respectively. The input current level at each time point is randomly generated according to a Gaussian distribution with the mean of $7mA$ and variance of $1mA$ as well as a uniform distribution between $[0.5mA, 5.5mA]$ in a way that is similar to the setups in [32, 35]. For both cases, a total of $10,000$ time points are considered.

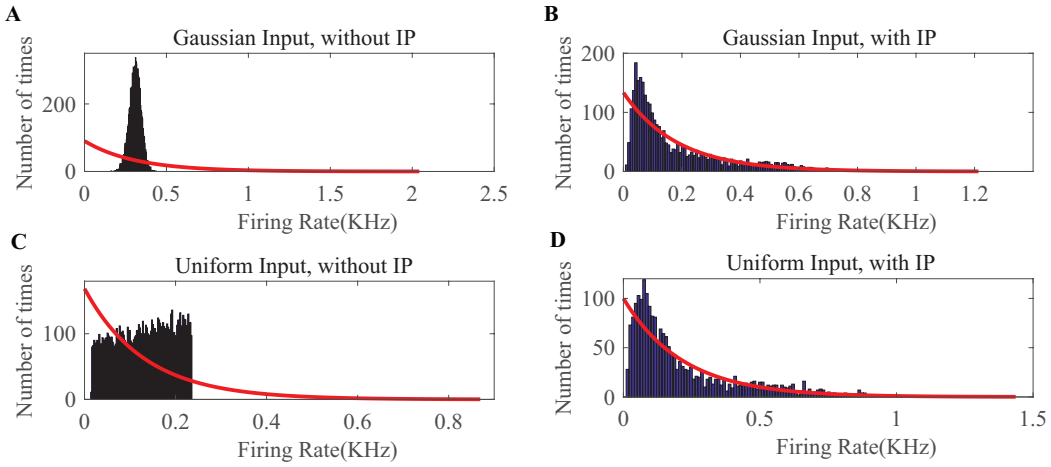

Figure 3.5: The output firing-rate distributions of a single neuron characterized using the firing-rate transfer function and driven by randomly generated current input following a Gaussian or Uniform distribution: (A) Gaussian input without IP tuning, (B) Gaussian input with the SpiKL-IP rule, (C) uniform input without IP tuning, and (D) uniform input with the SpiKL-IP rule. The red curve in each plot represents the exponential distribution that best fits the actual output firing rate data.

In Figure 3.5, we compare the recorded output firing rate distribution when no IP tuning is used with the one that is produced by the proposed SpiKL-IP rule under two random input distributions. In each plot of Figure 3.5, we fit the actual firing histogram with to the closest exponential distribution (red curve). It is evident from 3.5A and 3.5C that without IP tuning the output firing distribution is far from the targeted optimal exponential distribution with the maximum entropy. With the application of SpiKL-IP, however, the output distribution can be trained to almost converge to the desirable exponential distribution under two dramatically different input distributions. Note that since the simulation time step size is $1ms$, the output firing rate is bound by $1KHz$. This creates a subtle difference between the actual and the exponential distribution at the tails of the two distributions, which is negligible in practice. These results indicate that the proposed IP rule can robustly maximize the information contained in the output firing rate distribution by tuning it towards the exponential distribution regardless of the input distribution.

3.4.2 Leaky Integrate-and-Fire Spiking Neurons

Since SpiKL-IP is based on the firing-rate transfer function which only characterizes the behavior of LIF neurons over a large timescale, it is interesting to test SpiKL-IP using LIF neurons. The parameters for the spiking neurons and SpiKL-IP are set as follow: $V_{th} = 20mV$, $t_r = 2ms$, $\mu = 0.2KHz$, $\tau_c = 64ms$ with R and τ_m initialized to 64Ω and $64ms$, respectively. The tuning ranges for R and τ_m are again set to $[1\Omega, 1024\Omega]$ and $[1ms, 1,024ms]$, respectively.

First, we apply SpiKL-IP to a single LIF neuron whose input is a spike (Dirac delta) train randomly generated according to a Poisson process with a mean firing rate of 160 Hz for 1000 ms. The details of input generation are described in [67]. The output firing

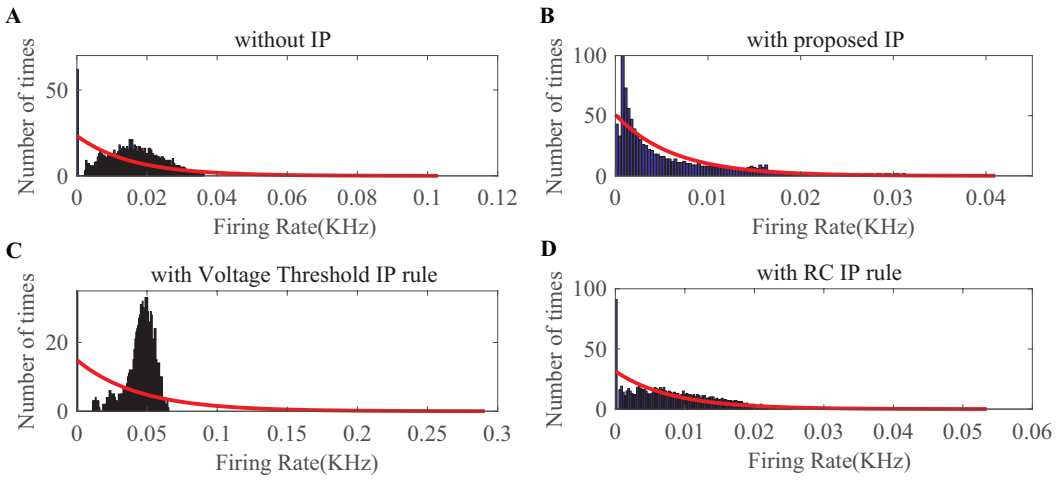

Figure 3.6: Output firing rate distributions of a single spiking neuron: (A) without IP tuning, (B) with proposed SpiKL-IP rule, (C) with the Voltage Threshold IP rule, and (D) with the RC IP rule. The red curve in each plot represents the exponential distribution that best fits the actual output firing rate data.

rate is evaluated by the normalized intracellular calcium concentration in (3.23). Figure 3.6 compares the output firing distributions generated with no IP and with the three IP rules. Clearly, the proposed rule produces an output distribution close to the desired exponential distribution while without IP tuning the neuron is unable to generate an exponentially distributed output. As shown in Figure 3.6C, the Voltage Threshold IP rule [33] can only alter the average output firing rate rather than tuning the shape of the output firing rate distribution towards that of an exponential distribution. Figure 3.6D shows that it is also tricky for the RC IP rule [35] to train the neuron to generate an output whose distribution is close to the exponential distribution.

Next, more interestingly, we examine the behavior of IP tuning in a spiking neural network. In this case, we set up a fully connected recurrent network of 100 LIF neurons. There are 30 external inputs with each being a Poisson spike train with a mean rate of $80Hz$ and a duration of $1,000ms$. Each input is connected to 30 neurons through synaptic whose weights are set to -8 or 8 with equal probability. The synaptic weights

between the reservoir neurons in the network are uniformly distributed between -1 and 1. This neural network is similar to the reservoir network used in [31].

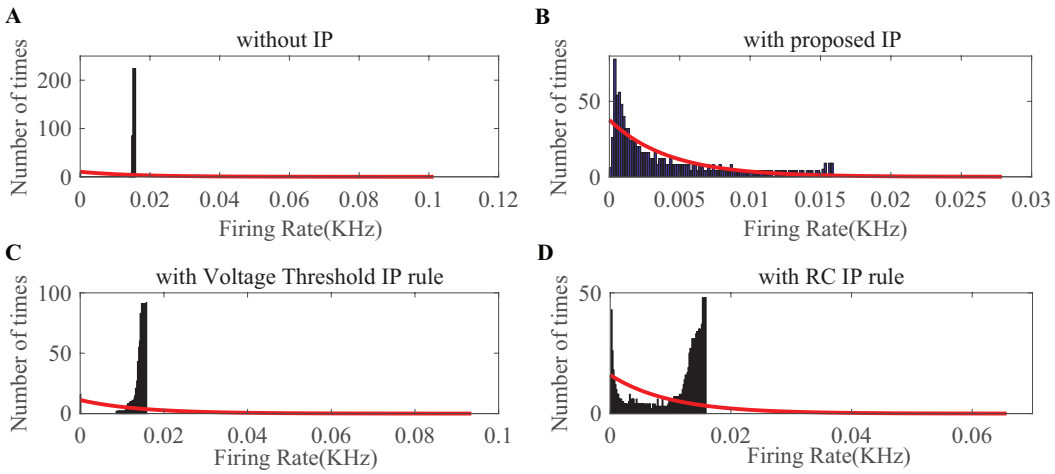

Figure 3.7: Output firing rate distributions of one spiking neuron in a fully connected network: (A) without IP tuning, (B) with proposed SpiKL-IP rule, (C) with the Voltage Threshold IP rule, and (D) with the RC IP rule. The red curve in each plot represents the exponential distribution that best fits the actual output firing rate data.

We randomly choose one neuron and record its output firing rate for a demonstration. As can be seen in Figure 3.7A, without IP tuning the output distribution is quite different from any exponential distributions. As shown in 3.7C and 3.7D, neither the Voltage Threshold IP rule nor the RC IP rule can produce an output distribution that is reasonably close to an exponential distribution. In contrast, the proposed SpiKL-IP rule leads to excellent results, generating an output distribution that is very close to an exponential distribution. These experiments demonstrate that SpiKL-IP maintains its effectiveness in the more complex network setting where spiking neurons interact with each other while receiving external spike inputs.

3.4.3 Real World Classification Tasks on Liquid State Machine

Although intrinsic plasticity has been studied for a very long time with many different IP rules proposed, rarely any rule is tested on real-world learning tasks. As a result, it is not clear whether IP tuning is capable of improving the performance of more meaningful tasks. In this paper, we realize several spiking neural networks based on the bio-inspired Liquid State Machine (LSM) network model and evaluate the performance of IP tuning using realistic speech recognition datasets TI46-Alpha [56].

LSM is a biologically plausible spiking neural network model with embedded recurrent connections [45]. The LSM has an input layer, a recurrent reservoir, and a readout layer. The reservoir has a recurrent structure with a group of excitatory and inhibitory spiking neurons randomly connected in a way approximating the spatial distribution of biological neurons [45]. Typically, the synaptic weights between the reservoir neurons are fixed. The input spike trains generate spatiotemporal firing patterns in the reservoir, which are projected onto the readout layer through full connectivity. In this paper, the feedforward plastic synapses between the reservoir neurons and readout are adjusted according to a bio-inspired spike-based online learning rule [46]. Several LSMs with different sizes are set up to evaluate the potential impact of an IP rule on classification performance.

For the networks evaluated using TI46-Alpha, the input layer has 78 neurons. These networks have 135(3*3*5), 270(3*3*30), 540(6*6*15) reservoir neurons, respectively, where each input neuron is randomly connected to 16, 24, 32 reservoir neurons with the weights set to 2 or -2 with equal probability, respectively. Among the reservoir neurons, 80% are excitatory, and 20% are inhibitory. The reservoir is composed of all types of synaptic connections depending on the pre-neuron and post-neuron types including EE, EI, IE, II, where the first letter indicates the type of the presynaptic neuron, and the second letter the type of the postsynaptic neuron, and E and I mean excitatory and inhibitory

neurons, respectively. The probability of a synaptic connection from neuron a to neuron b in the reservoir is defined as $C \cdot e^{-(D(a,b)/\lambda)^2}$, where λ is 3, C is 0.3(EE), 0.2(EI), 0.4(IE), 0.1(II) and D(a, b) is the Euclidean distance between neurons a and b [45]. The synaptic weights in the reservoir are fixed to 1(EE, EI) or -1(IE, II). For the readout layer, the reservoir neurons are fully connected to 26 readout neurons with the weights randomly generated from -8 to 8 following the Gaussian distribution. All the readout synapses are plastic and trained according to [46]. When testing an IP rule, it is only applied to the reservoir neurons. The parameters of each neuron are: $V_{th} = 20mV$, $t_r = 2ms$, $\mu = 0.2KHz$, $\tau_c = 64ms$, $\eta_1 = \eta_2 = 5$, and $\alpha_1 = \alpha_2 = 0.1$. R and τ_m are initialized to 64Ω and $64ms$, respectively. The tuning ranges for R and τ_m are again set to $[32\Omega, 512\Omega]$ and $[32ms, 512ms]$, respectively. A five-fold cross-validation scheme is adopted to obtain classification performances. Five hundred epochs are simulated, and the best results are reported.

Table 3.1: The performances of LSM-based speech recognition with and without the proposed SpiKL-IP rule evaluated using the single and multi-speaker subsets of the TI46 Speech Corpus.

Dataset Size	Reservoir Size	Without IP	With IP
260(1 Speaker)	90	88.46%	97.31%
	135	92.30%	98.46%
520(2 Speakers)	135	86.15%	92.31%
	270	89.04%	95.58%
1040(4 Speakers)	135	79.04%	87.69%
	270	84.62%	93.37%
2080(8 Speakers)	270	72.69%	86.95%
	540	76.59%	91.96%
3120(12 Speakers)	270	72.17%	84.25%
	540	77.49%	90.64%
4160(16 Speakers)	270	70.76%	83.98%
	540	76.19%	88.58%

Table 3.1 demonstrates the classification accuracy for a number of LSMs of different amounts of reservoir neurons with and without the proposed SpiKL-IP rule based on different subsets of the TI46 speech corpus. The 260-samples subset is a single speaker subset while the ones with 520, 1,040, 2,080, 3,120, 4,160 samples contain 2, 4, 8, 12, and 16 speakers, respectively. It shall be noted that as the number of speakers increases, the recognition task becomes increasingly challenging. To the best knowledge of the authors, there exists no prior reported success in recognizing multiple-speaker subsets using spiking neural networks. As shown in Table 3.1, the recognition performs drops rapidly as the number of speakers increases without SpiKL-IP. In comparison, the use of SpiKL-IP can significantly boost the recognition accuracy by up to more than 16%. Moreover, SpiKL-IP leads to higher performance boosts as it is applied to smaller networks or more challenging subsets of greater numbers of speakers and samples.

From the LSM with 135 reservoir neurons, we randomly choose six neurons and record their firing responses on one of the speech samples after a few initial training iterations. Figure 3.8 shows that most neurons' responses can follow the exponential distribution, demonstrating that the proposed SpiKL-IP rule can tune neurons to generate outputs with a distribution close to the exponential distribution in a complicated network.

Figure 3.9 compares the recognition performances of several LSMs all with 135 reservoir neurons reported in related works. The performances are evaluated based upon the single-speaker subset with 260 samples. We adopt the LSM in [46] which makes use of a spike-based supervised learning rule for training the readout synapses and has no IP tuning as a baseline. The LSM in [9] adds the spike-timing-dependent-plasticity (STDP) rule to the baseline to train the synaptic weights between reservoir neurons. On top of the baseline, we further implement the Voltage Threshold IP rule [33], the RC IP rule [35], or the SpiKL-IP rule to tune the reservoir neurons. The proposed rule produces the highest recognition accuracy improvement of more than 6% over the baseline LSM.

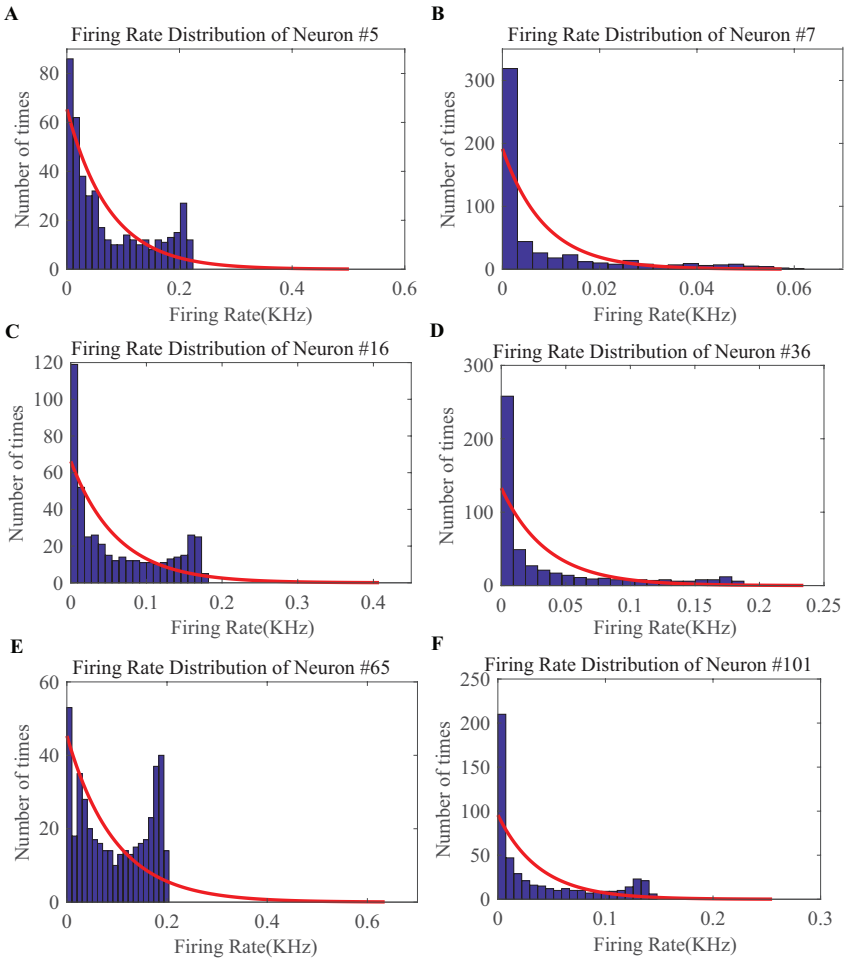

Figure 3.8: The output firing distributions of six reservoir neurons in an LSM after the reservoir is trained. The red curve in each plot represents the exponential distribution that best fits the actual output firing rate data.

3.5 Summary and Discussions

While intrinsic plasticity (IP) was attempted for spiking neurons in the past, the prior IP rules lacked a rigorous treatment in their development, and the efficacy of these rules was not verified using practical learning tasks. In this chapter, we address the theoretical and practical limitations of the existing works by proposing the SpiKL-IP rule. SpiKL-IP is based upon a rigorous information-theoretic perspective where the target of IP tuning is to produce the maximum entropy in the resulting output firing rate distribution of each

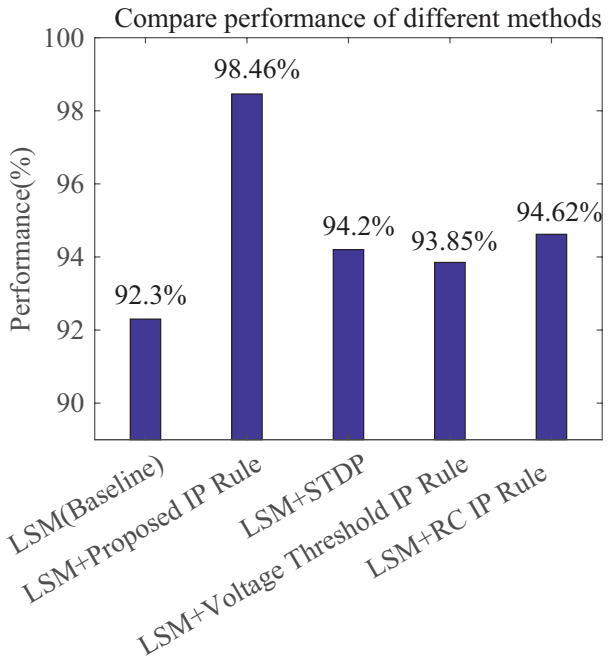

Figure 3.9: Speech recognition performances of various learning rules when applied to a LSM with 135 reservoir neurons. The performance evaluation is based on the single-speaker subset of the TI46 Speech Corpus. (1) LSM (Baseline): with the settings and supervised readout learning rule in [46] and no reservoir tuning. All other compared networks add additional mechanisms to the baseline. (2) LSM+Proposed IP Rule: with additional reservoir neurons tuning using SpiKL-IP. (3) LSM+STDP: with additional reservoir neurons tuning using the STDP rule in [9]; (4) LSM+Voltage Threshold IP Rule: with additional reservoir neurons tuning using the IP rule in [33]. (5) LSM+RC IP Rule: with additional reservoir neurons tuning using the IP rule in [35].

spiking neuron. The maximization of output entropy, or information transfer from the input to the output, is realized by producing a targeted optimal exponential distribution of the output firing rate.

More specifically, SpiKL-IP aims to tune the intrinsic parameters of a spiking neuron while minimizing the KL-divergence from the targeted exponential distribution to the actual output firing rate distribution. However, several challenges must be addressed as we work toward achieving the above goal. First, we rigorously relate the output firing rate with the static input current by deriving the firing-rate transfer function (FR-TF).

FR-TF provides a basis for allowing the derivation of the SpiKL-IP rule that minimizes the KL-divergence. Furthermore, we cast SpiKL-IP in a suitable form to enable the online application of IP tuning. Finally, we address one major challenge associated with applying SpiKL-IP under realistic contexts where the input current to each spiking neuron may be time-varying, which leads to the final IP rule that has no dependency on the instantaneous input level and effectively tuning the neural model parameters based upon averaged firing activities.

Chapter 4

Spike-Train Level Backpropagation for Recurrent Spiking Neural Networks

In Chapter 3, we demonstrate an unsupervised learning method SpiKL-IP. However, biologically inspired unsupervised learning has limited capability in boosting the performance of spiking neural networks (SNNs). SNNs well support spatio-temporal learning and energy efficient event-driven hardware neuromorphic processors. As an important class of SNNs, recurrent spiking neural networks (RSNNs) possess great computational power. However, the practical application of RSNNs is severely limited by challenges in training. Inspired by the success of error backpropagation (BP) and its variants in training conventional deep neural networks (DNNs), various SNNs BP methods have emerged, aiming at attaining the same level of performance [13, 14, 15, 16, 17, 18, 19, 20, 21]. Although many appealing results are achieved by these methods, developing SNNs BP training methods that are on a par with the mature BP tools widely available for training ANNs today is a nontrivial problem [26]. Existing backpropagation (BP) methods

suffer from high complexity of unfolding in time, vanishing and exploding gradients, and approximate differentiation of discontinuous spiking activities when applied to RSNNs. For example, backpropagation through time (BPTT) in principle may be applied to training RSNNs [20], but bottlenecked with several challenges in: (1) unfolding the recurrent connections through time, (2) backpropagating errors over both time and space, and (3) backpropagating errors over non-differentiable spike events.

This chapter is motivated by: **1)** lack of powerful supervised training of general RSNNs, and **2)** an immediate outcome of 1), i.e. the existing SNN research has limited scope in exploring sophisticated learning architectures like deep RSNNs with multiple feedforward and recurrent layers hybridized together. As a first step towards addressing these challenges, we propose a novel biologically non-plausible Spike-Train level RSNNs Backpropagation (ST-RSBP) algorithm which is applicable to RSNNs with arbitrary network topology and achieves state-of-the-art performances on several widely used datasets. The proposed ST-RSBP employs spike-train level computation similar to what is adopted in the recent hybrid macro/micro level BP (HM2-BP) method for feedforward SNNs [16], which demonstrates encouraging performances and outperforms BPTT such as the one implemented in [15].

Fig. 4.1 compares BPTT and ST-RSBP, where we focus on a recurrent layer since the feedforward layer can be viewed as a simplified recurrent layer. To apply BPTT, one shall first unfold an RSNN in time to convert it into a larger feedforward network without recurrent connections. The total number of layers in the feedforward network is increased by a factor equal to the number of times the RSNN is unfolded, and hence can be very large. Then, this unfolded network is integrated in time with a sufficiently small time step to capture the dynamics of the spiking behavior. BP is then performed spatio-temporally layer-by-layer across the unfolded network based on the same time step size used for integration as shown in Fig. 4.1. In contrast, ST-RSBP is rigorously derived and can

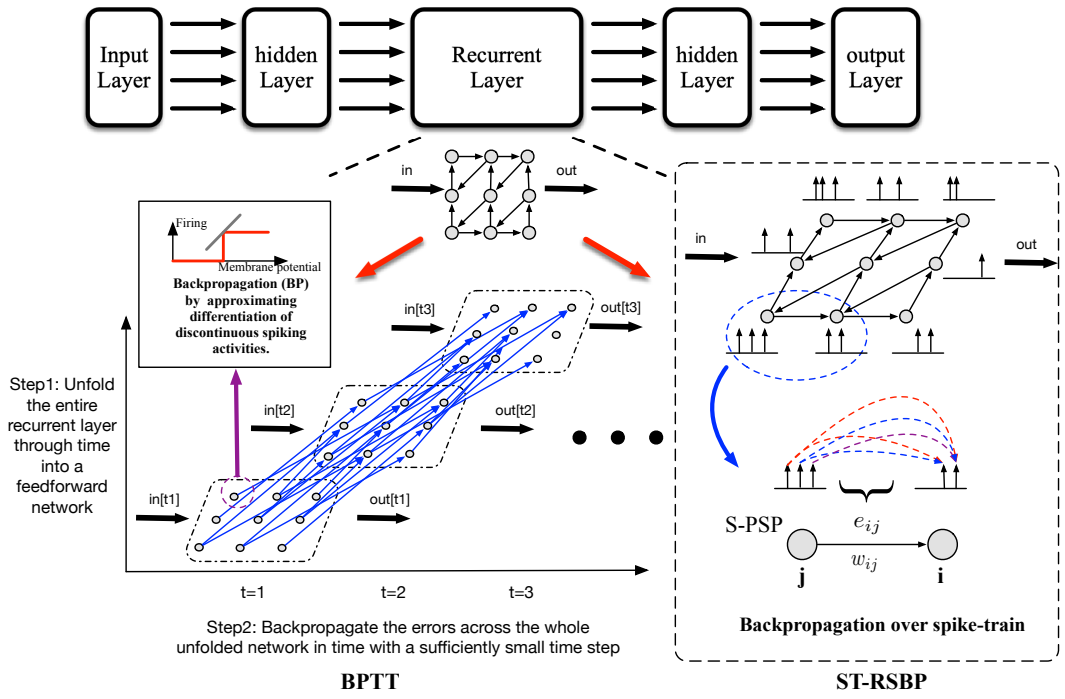

Figure 4.1: Backpropagation in recurrent SNNs: BPTT vs. ST-RSBP.

handle arbitrary recurrent connections in various RSNNs. While capturing the temporal behavior of the RSNN at the spike-train level, ST-RSBP directly computes the gradient of a rate-coded loss function w.r.t tunable parameters without incurring approximations resulted from altering and smoothing the underlying spiking behaviors. ST-RSBP is able to train RSNNs without costly unfolding the network through time and performing BP time point by time point, offering faster training and avoiding vanishing/exploding gradients for general RSNNs. Moreover, since ST-RSBP more precisely computes error gradients than HM2-BP [16], it can achieve better results than HM2-BP even on the feedforward SNNs.

4.1 Spike-train Level postsynaptic Potential

Spike-train Level postsynaptic Potential (S-PSP) captures the spike-train level interactions between a pair of pre/postsynaptic neurons and can be defined for any neural models with all-or-none spiking characteristics and any synaptic models [16]. Without loss of generality, we describe S-PSP using the LIF model and the first order synaptic model introduced in (2.1) and (2.3), respectively.

The integration of (2.1) and (2.3) leads to the spike response model (SRM) [52]:

$$u_i(t) = \sum_j w_{ij} \sum_{t_j^{(f)}} \epsilon\left(t - \hat{t}_i^{(f)}, t - t_j^{(f)}\right), \tag{4.1}$$

where $\hat{t}_i^{(f)}$ denotes the last firing time of the neuron i and $t_j^{(f)}$ denotes a particular firing time of the neuron j. $\epsilon(s,t)$ specifies the normalized time course of the *postsynaptic potential* evoked by a single firing spike of the presynaptic neuron:

$$\epsilon(s,t) = \frac{1}{C}\int_0^s \exp\left(-\frac{t'}{\tau_m}\right)\ \alpha_i\left(t - t'\right)\ \mathrm{d}t'. \tag{4.2}$$

Through integration, (4.2) can be re-written as:

$$\epsilon(s,t) = \frac{e^{(-\max(t-s,0)/\tau_s)}}{1-\frac{\tau_s}{\tau_m}} \quad \left[e^{\left(-\frac{\min(s,t)}{\tau_m}\right)} - e^{\left(-\frac{\min(s,t)}{\tau_s}\right)}\right] H(s)H(t), \tag{4.3}$$

where $H(t)$ is the Heaviside step function.

Note that each neuron fires whenever its postsynaptic potential reaches the firing threshold. We now sum up the contributions of the presynaptic neuron j's spike train to the (normalized) postsynaptic potential of the neuron i right before all the neuron i's

firing times as illustrated in Fig. 4.2:

$$e_{ij} = \sum_{t_i^{(f)}} \sum_{t_j^{(f)}} \epsilon(t_i^{(f)} - \hat{t}_i^{(f)}, t_i^{(f)} - t_j^{(f)}), \tag{4.4}$$

defining the (normalized) **spike-train level postsynaptic potential** (S-PSP) from the neuron j to the neuron i.

The significance of S-PSPs lies in that it characterizes the aggregated effect of the spike train of the presynaptic neuron j on the membrane potential of the postsynaptic neuron i and its firing activities. Employing S-PSPs in the proposed ST-RSBP algorithm is beneficial; it allows efficient consideration of the temporal dynamics and recurrent connections of an RSNN across all firing events at the spike-train level without expensive unfolding in time and backpropagation time point by time point, which are required by BPTT.

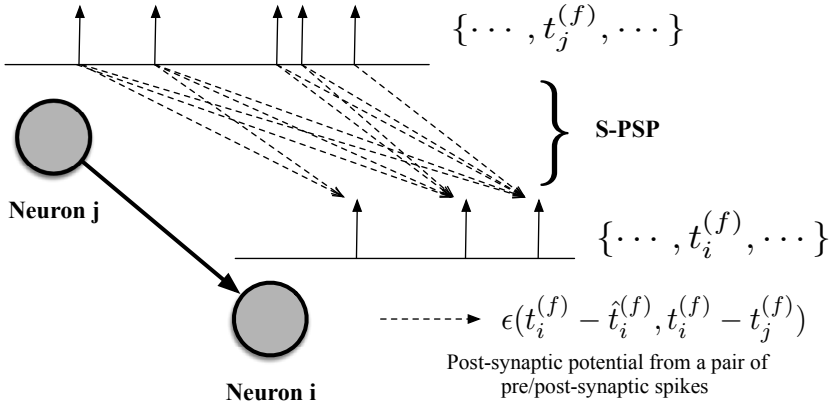

Figure 4.2: The computation of the S-PSP.

The sum of the weighted S-PSPs from all presynaptic neurons of the neuron i is defined as the **total postsynaptic potential** (**T-PSP**) α_i, relating to the neuron i's

firing count o_i via the firing threshold ν:

$$a_i = \sum_j w_{ij}\ e_{ij}, \qquad o_i = g(\alpha_i) \approx \frac{a_i}{\nu}. \tag{4.5}$$

a_i and o_i are analogous to the pre-activation and activation in the traditional ANNs, respectively, and $g(\cdot)$ can be considered as an activation function converting the T-PSP to the output firing count.

Note that, in this chapter, we use a to denote T-PSP while in other chapters a represents the postsynaptic current (PSC) defined in (2.2) which is induced by the spikes from the presynaptic neuron through the synaptic kernel.

4.2 Proposed ST-RSBP

We use the generic recurrent spiking neural network with a combination of feedforward and recurrent layers to derive ST-RSBP. For the spike-train level activation of each neuron l in the layer $k+1$, (4.5) is modified to include the recurrent connections explicitly if necessary:

$$a_l^{k+1} = \sum_{j=1}^{N_k} w_{lj}^{k+1}\ e_{lj}^{k+1} + \sum_{p=1}^{N_{k+1}} w_{lp}^{k+1}\ e_{lp}^{k+1}, \qquad o_l^{k+1} = g(a_l^{k+1}) \approx \frac{a_l^{k+1}}{\nu^{k+1}}. \tag{4.6}$$

N_{k+1} and N_k are the number of neurons in the layers $k+1$ and k, w_{lj}^{k+1} is the feedforward weight from the neuron j in the layer k to the neuron l in the layer $k+1$, w_{lp}^{k+1} is the recurrent weight from the neuron p to the neuron l in the layer $k+1$, which is non-existent if the layer $k+1$ is feedforward, e_{lj}^{k+1} and e_{lp}^{k+1} are the corresponding S-PSPs, ν^{k+1} is the firing threshold at the layer $k+1$, o_l^{k+1} and a_l^{k+1} are the firing count and pre-activation (T-PSP) of the neuron l at the layer $k+1$, respectively.

The rate-coded loss is defined at the output layer as:

$$E = \frac{1}{2}||\boldsymbol{o} - \boldsymbol{y}||_2^2 = \frac{1}{2}||\frac{\boldsymbol{a}}{\nu} - \boldsymbol{y}||_2^2, \tag{4.7}$$

where $\boldsymbol{y}$, $\boldsymbol{o}$ and $\boldsymbol{a}$ are vectors of the desired output neuron firing counts (labels) and actual firing counts, and the T-PSPs of the output neurons, respectively. Differentiating (4.7) with respect to each trainable weight w_{ij}^k incident upon the layer k leads to:

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial a_i^k}\frac{\partial a_i^k}{\partial w_{ij}^k} = \delta_i^k \frac{\partial a_i^k}{\partial w_{ij}^k} \qquad \text{with} \quad \delta_i^k = \frac{\partial E}{\partial a_i^k}, \tag{4.8}$$

where δ_i^k and $\frac{\partial a_i^k}{\partial w_{ij}^k}$ are referred to as the ***back propagated error*** and ***differentiation of activation***, respectively, for the neuron i. ST-RSBP updates w_{ij}^k by $\Delta w_{ij}^k = \eta \frac{\partial E}{\partial w_{ij}^k}$, where η is a learning rate.

4.2.1 Back Propagated Errors

When the layer k is the output layer, the back propagated error at the i_{th} neuron of the layer is given by differentiating the loss defined in (4.7):

$$\delta_i^k = \frac{\partial E}{\partial a_i^k} = \frac{(o_i^k - y_i^k)}{\nu^k}, \tag{4.9}$$

where o_i^k is the actual firing count, y_i^k the desired firing count (label), and a_i^k the corresponding T-PSP.

For the hidden layer case, at each hidden layer k, by applying the chain rule, the back propagated error δ_i^k for the neuron i can be expressed as:

$$\delta_i^k = \frac{\partial E}{\partial a_i^k} = \sum_{l=1}^{N_{k+1}} \frac{\partial E}{\partial a_l^{k+1}} \frac{\partial a_l^{k+1}}{\partial a_i^k} = \sum_{l=1}^{N_{k+1}} \delta_l^{k+1} \frac{\partial a_l^{k+1}}{\partial a_i^k}. \tag{4.10}$$

N_{k+1} is the number of neurons in the layer $k+1$. Define two error vectors $\boldsymbol{\delta}^{k+1}$ and $\boldsymbol{\delta}^{k}$ for the two layers: $\boldsymbol{\delta}^{k+1} = [\delta_1^{k+1}, \cdots, \delta_{N_{k+1}}^{k+1}]$, and $\boldsymbol{\delta}^{k} = [\delta_1^{k}, \cdots, \delta_{N_k}^{k}]$, respectively for the layers $k+1$ and k, where N_k is the number of the neurons in the layer k. Assuming $\boldsymbol{\delta}^{k+1}$ is given, which is the case for the output layer based on (4.9), the goal is to back propagate from $\boldsymbol{\delta}^{k+1}$ to $\boldsymbol{\delta}^{k}$. Clearly, this entails to compute $\frac{\partial a_l^{k+1}}{\partial a_i^k}$ in (4.10).

[**Backpropagation from a Hidden Recurrent Layer**] Now consider that case that the errors are back propagated from a recurrent layer $k+1$ to its preceding layer k. Note that the S-PSP e_{lj} from any presynaptic neuron j to a post-synpatic neuron l is a function of both the rate and temporal information of the pre/post spike trains, which can be made explicitly via some function f:

$$e_{lj} = f(o_j, o_l, \boldsymbol{t}_j^{(f)}, \boldsymbol{t}_l^{(f)}), \tag{4.11}$$

where o_j, o_l, $\boldsymbol{t}_j^{(f)}$, $\boldsymbol{t}_l^{(f)}$ are the presynaptic/postsynaptic firing counts and firing times, respectively.

Now based on (4.6), $\frac{\partial a_l^{k+1}}{\partial a_i^k}$ is split also into two summations:

$$\frac{\partial a_l^{k+1}}{\partial a_i^k} = \sum_j^{N_k} w_{lj}^{k+1} \frac{de_{lj}^{k+1}}{da_i^k} + \sum_p^{N_{k+1}} w_{lp}^{k+1} \frac{de_{lp}^{k+1}}{da_i^k}, \tag{4.12}$$

where the first summation sums over all presynaptic neurons in the previous layer k while the second sums over the presynaptic neurons in the current recurrent layer as illustrated in Fig. 4.3.

On the right side of (4.12), $\frac{de_{lj}^{k+1}}{da_i^k}$ is given by:

$$\frac{de_{lj}^{k+1}}{da_i^k} = \begin{cases} \frac{1}{\nu^k}\frac{\partial e_{li}^{k+1}}{\partial o_i^k} + \frac{1}{\nu^{k+1}}\frac{\partial e_{lj}^{k+1}}{\partial o_l^{k+1}}\frac{\partial a_l^{k+1}}{\partial a_i^k} & j = i \\ \frac{1}{\nu^{k+1}}\frac{\partial e_{lj}^{k+1}}{\partial o_l^{k+1}}\frac{\partial a_l^{k+1}}{\partial a_i^k} & j \neq i, \end{cases} \tag{4.13}$$

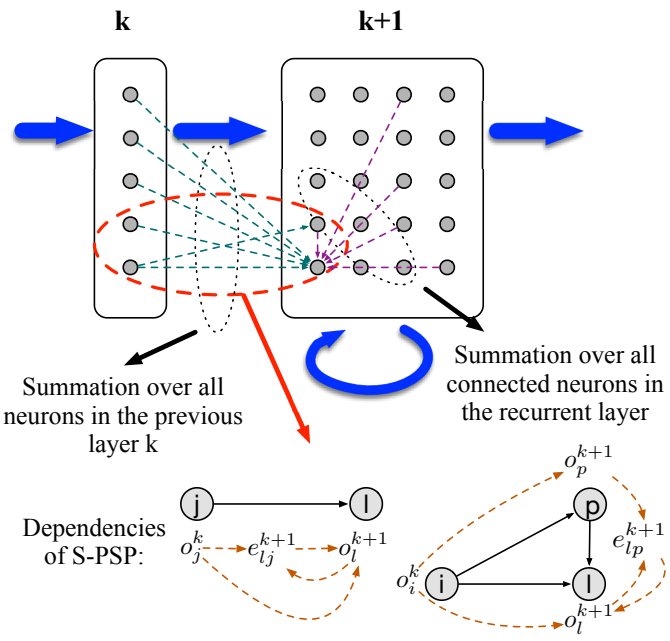

Figure 4.3: Connections for a recurrent layer neuron and the dependencies among its S-PSPs.

where ν^k and ν^{k+1} are the firing threshold voltages for the layers k and $k+1$, respectively, and we have used that $o_i^k \approx a_i^k/\nu^k$ and $o_l^{k+1} \approx a_l^{k+1}/\nu^{k+1}$ from (4.5). Importantly, the last term on the right side of (4.13) exists due to e_{lj}^{k+1}'s dependency on the postsynaptic firing rate o_l^{k+1} per (4.11) and o_l^{k+1}'s further dependency on the presynaptic activation o_i^k (hence pre-activation a_i^k), as shown in Fig. 4.3.

On the right side of (4.12), $\frac{de_{lp}^{k+1}}{da_i^k}$ is due to the recurrent connections within the layer $k+1$ and is given by:

$$\frac{de_{lp}^{k+1}}{da_i^k} = \frac{1}{\nu^{k+1}} \frac{\partial e_{lp}^{k+1}}{\partial o_l^{k+1}} \frac{\partial a_l^{k+1}}{\partial a_i^k} + \frac{1}{\nu^{k+1}} \frac{\partial e_{lp}^{k+1}}{\partial o_p^{k+1}} \frac{\partial a_p^{k+1}}{\partial a_i^k}. \tag{4.14}$$

The first term on the right side of (4.14) is due to e_{lp}^{k+1}'s dependency on the postsynaptic firing rate o_l^{k+1} per (4.11) and o_l^{k+1}'s further dependence on the presynaptic activation o_i^k (hence pre-activation a_i^k). Per (4.11), it is important to note that the second term exists because e_{lp}^{k+1}'s dependency on the presynaptic firing rate o_p^{k+1} , which

further depends on o_i^k (hence pre-activation a_i^k), as shown in Fig. 4.3.

Putting (4.12), (4.13), and (4.14) together leads to:

$$\begin{aligned}\frac{\partial a_l^{k+1}}{\partial a_i^k} =& w_{li}^{k+1}\ \frac{1}{\nu^k}\frac{\partial e_{li}^{k+1}}{\partial o_i^k} + \frac{1}{\nu^{k+1}}\frac{\partial a_l^{k+1}}{\partial a_i^k}\left(\sum_j^{N_k} w_{lj}^{k+1}\ \frac{\partial e_{lj}^{k+1}}{\partial o_l^{k+1}} + \sum_p^{N_{k+1}} w_{lp}^{k+1}\ \frac{\partial e_{lp}^{k+1}}{\partial o_l^{k+1}}\right) \\ &+ \sum_p^{N_{k+1}} w_{lp}^{k+1}\frac{1}{\nu^{k+1}}\ \frac{\partial e_{lp}^{k+1}}{\partial o_p^{k+1}}\frac{\partial a_p^{k+1}}{\partial a_i^k}.\end{aligned} \tag{4.15}$$

Now, (4.15) is rearranged to:

$$\begin{aligned}&\left(1 - \frac{1}{\nu^{k+1}}\left(\sum_j^{N_k} w_{lj}^{k+1}\ \frac{\partial e_{lj}^{k+1}}{\partial o_l^{k+1}} + \sum_p^{N_{k+1}} w_{lp}^{k+1}\ \frac{\partial e_{lp}^{k+1}}{\partial o_l^{k+1}}\right)\right)\frac{\partial a_l^{k+1}}{\partial a_i^k} = \\ &w_{li}^{k+1}\ \frac{1}{\nu^k}\frac{\partial e_{li}^{k+1}}{\partial o_i^k} + \sum_p^{N_{k+1}} w_{lp}^{k+1}\frac{1}{\nu^{k+1}}\ \frac{\partial e_{lp}^{k+1}}{\partial o_p^{k+1}}\frac{\partial a_p^{k+1}}{\partial a_i^k}.\end{aligned} \tag{4.16}$$

It is evident that all $N_{k+1} \times N_k$ partial derivatives involving the recurrent layer $k+1$ and its preceding layer k, i.e. $\frac{\partial a_l^{k+1}}{\partial a_i^k}, l = [1, N_{k+1}], i = [1, N_k]$, form a coupled linear system via (4.16), which is written in a matrix form as:

$$\boldsymbol{\Omega}^{k+1,k} \cdot \boldsymbol{P}^{k+1,k} = \boldsymbol{\Phi}^{k+1,k} + \boldsymbol{\Theta}^{k+1,k} \cdot \boldsymbol{P}^{k+1,k}, \tag{4.17}$$

where $\boldsymbol{P}^{k+1,k} \in \mathbb{R}^{N_{k+1} \times N_k}$ contains all the desired partial derivatives, $\boldsymbol{\Omega}^{k+1,k} \in \mathbb{R}^{N_{k+1} \times N_{k+1}}$ is diagonal, $\boldsymbol{\Theta}^{k+1,k} \in \mathbb{R}^{N_{k+1} \times N_{k+1}}$, $\boldsymbol{\Phi}^{k+1,k} \in \mathbb{R}^{N_{k+1} \times N_k}$, and

$$\begin{aligned}&\Omega_{ij}^{k+1,k} = \begin{cases} 1 - \frac{1}{\nu^{k+1}}\left(\sum_m^{N_k} w_{lm}^{k+1}\ \frac{\partial e_{lm}^{k+1}}{\partial o_l^{k+1}} + \sum_p^{N_{k+1}} w_{lp}^{k+1}\ \frac{\partial e_{lp}^{k+1}}{\partial o_l^{k+1}}\right) & i = j \\ 0 & i \neq j \end{cases} \\ &P_{ij}^{k+1,k} = \frac{\partial a_i^{k+1}}{\partial a_j^k} \qquad \Phi_{ij}^{k+1,k} = w_{ij}^{k+1}\frac{1}{\nu^k}\ \frac{\partial e_{ij}^{k+1}}{\partial o_j^k} \qquad \Theta_{ij}^{k+1,k} = w_{ij}^{k+1}\ \frac{1}{\nu^{k+1}}\frac{\partial e_{ij}^{k+1}}{\partial o_j^{k+1}}.\end{aligned} \tag{4.18}$$

The partial derivatives of the S-PSP with respect to the presynaptic and postsynaptic firing counts, i.e. $\frac{\partial e_{ij}^{k+1}}{\partial o_j^k}$ and $\frac{\partial e_{ij}^{k+1}}{\partial o_i^{k+1}}$ as needed in (4.18) will be determined in Section 4.2.3. Solving the linear system in (4.17) gives all $\frac{\partial a_i^{k+1}}{\partial a_j^k}$:

$$\boldsymbol{P}^{k+1,k} = (\boldsymbol{\Omega}^{k+1,k} - \boldsymbol{\Theta}^{k+1,k})^{-1} \cdot \boldsymbol{\Phi}^{k+1,k}. \tag{4.19}$$

Note that since $\boldsymbol{\Omega}$ is a diagonal matrix, the cost in factoring the above linear system can be reduced by approximating the matrix inversion using a first-order Taylor's expansion without performing any matrix factorization.

All N_k errors at the layer k back propagated from the layer $k+1$ per (4.10) is put into a vector form: $\boldsymbol{\delta^k} = [\delta_1^k, \cdots, \delta_{N_k}^k]$, and is given by:

$$\boldsymbol{\delta}^k = (\boldsymbol{P}^{k+1,k})^T \cdot \boldsymbol{\delta}^{k+1}, \tag{4.20}$$

where $\boldsymbol{\delta}^{k+1}$ is the error vector at the layer $k+1$.

[**Backpropagation from a Hidden Feedforward Layer**] Consider the much simpler case of backpropagating errors from a feedforward layer $k+1$ to its preceding layer k. Due to non-existence of recurrent connections in the layer $k+1$, (4.19) is simplified to:

$$\boldsymbol{P}^{k+1,k} = (\boldsymbol{\Omega}^{k+1,k})^{-1} \cdot \boldsymbol{\Phi}^{k+1,k}. \tag{4.21}$$

Since $\boldsymbol{\Omega}^{k+1,k}$ is diagonal, each $\frac{\partial a_l^{k+1}}{\partial a_i^k}$ can be directly computed:

$$\frac{\partial a_l^{k+1}}{\partial a_i^k} = \frac{\frac{1}{\nu^k} w_{li}^{k+1} \frac{\partial e_{li}^{k+1}}{\partial o_i^k}}{1 - \frac{1}{\nu^{k+1}} \sum_{p=1}^{N_k} w_{lp}^{k+1} \frac{\partial e_{lp}^{k+1}}{\partial o_l^{k+1}}}. \tag{4.22}$$

4.2.2 Differentiation of Activation

Per (4.8), we derive the differentiation of activation $\frac{\partial a_i^k}{\partial w_{ij}}$ under two cases.

[**Backpropagation from a Hidden Feedforward Layer**] For a feedforward layer k and based on (4.6), differentiation of each activation is given by:

$$\frac{\partial a_i^k}{\partial w_{ij}^k} = \frac{\partial}{\partial w_{ij}^k}\left(\sum_l^{N_{k-1}} w_{il}^k\ e_{il}^k\right) = e_{ij}^k + \frac{1}{\nu^k}\frac{\partial a_i^k}{\partial w_{ij}^k}\sum_l^{N_{k-1}} w_{il}^k \frac{\partial e_{il}^k}{\partial o_i^k}. \tag{4.23}$$

The first term on the right side of (4.23) reflects the direct dependency of a_i^k on w_{ij}^k while the second term captures the dependency of each S-PSP e_{il}^k on the post-synaptic firing count o_i^k, which further depends on w_{ij} according to (4.11). The derivative $\frac{\partial a_i^k}{\partial w_{ij}^k}$ on the right side of (4.23) is precisely considered in ST-RSBP. However, HM2-BP [16] does not consider the hidden dependency of e_{ij}^k on w_{ij}^k when deriving (4.23). As a result, the $\frac{\partial a_i^k}{\partial w_{ij}^k}$ term on the right side of (4.23) is approximated to e_{ij}^k.

(4.23) gives the desired differentiation of activation as:

$$\frac{\partial a_i^k}{\partial w_{ij}^k} = \frac{e_{ij}^k}{1 - \frac{1}{\nu^k}\sum_l^{N_{k-1}} w_{il}^k \frac{\partial e_{il}^k}{\partial o_i^k}}. \tag{4.24}$$

[**Recurrent Layers**] For the activation a_i^k of the neuron i at the recurrent layer k, we further consider the recurrent connections and get

$$\frac{\partial a_i^k}{\partial w_{ij}^k} = \frac{\partial}{\partial w_{ij}^k}\left(\sum_l^{N_{k-1}} w_{il}^k\ e_{il}^k + \sum_p^{N_k} w_{ip}^k\ e_{ip}^k\right) = e_{ij}^k + \frac{\partial a_i^k}{\partial w_{ij}^k}\frac{1}{\nu^k}\left(\sum_l^{N_{k-1}} w_{il}^k \frac{\partial e_{il}^k}{\partial o_i^k} + \sum_p^{N_k} w_{ip}^k \frac{\partial e_{ip}^k}{\partial o_i^k}\right),$$

leading to:

$$\frac{\partial a_i^k}{\partial w_{ij}} = \frac{e_{ij}^k}{1 - \frac{1}{\nu^k}\left(\sum_l^{N_{k-1}} w_{il}^k \frac{\partial e_{il}^k}{\partial o_i^k} + \sum_p^{N_k} w_{ip}^k \frac{\partial e_{ip}^k}{\partial o_i^k}\right)}. \tag{4.25}$$

4.2.3 Differentiation of S-PSP w.r.t Pre/PostSynaptic Firing Counts

Before presenting the final ST-RSBP algorithm, we shall determine the partial derivatives $\frac{\partial e_{ij}}{\partial o_j}$ and $\frac{\partial e_{ij}}{\partial o_i}$ of an S-PSP e_{ij} with respect to the firing counts of the pre-synaptic neuron j and post-synaptic neuron i, respectively, as needed in (4.18), (4.22), (4.24), and (4.25). As discussed in Section 4.1, S-PSPs serve as a bridge between neuron-level firing timings and spike-train level firing count and allow backpropagating errors defined for a rate-coded loss at the spike-train level.

In [16], the HM2-BP computes the two partial derivatives by assuming that each S-PSP e_{ij} is approximately linear in both o_j and o_i.

To examine this assumption, we evaluate the S-PSP from the neuron j to neuron i via a synapse. The LIF neuron model of (2.1) and the synaptic model of (2.3) with $\tau_m = 64ms$, $\tau_s = 8ms$ are adopted in this analysis. The simulation duration is set to $600ms$ and the first-order Euler method with a fixed stepsize of $1ms$ is used for simulation. To cover a wide range of interactions between the two neurons, we consider all combinations of the firing rates of two neurons o_i and o_j when they are swept widely from 1 to 50. For each combination of o_i and o_j values, we generate the spike trains of the two neurons by randomly choosing o_i and o_j numbers of random spiking times, respectively, and compute the S-PSP e_{ij} according to (4.4). We repeat this process 500 times and take the average value of e_{ij}.

We plot the relation between the pre/post-synaptic firing counts o_j and o_i and the average e_{ij} in Fig. 4.4A. Fig. 4.4B shows that with o_i fixed e_{ij} increases rather linearly

in o_j, consistent with [16], and hence we have:

$$\frac{\partial e_{ij}}{\partial o_j} \approx \frac{e_{ij}}{o_j}. \tag{4.26}$$

However, Fig. 4.4C shows that with o_j fixed, e_{ij} is not linear in a wide range of o_i, suggesting that the assumption made in [16] can lead to errors when the postsynaptic firing rates vary a lot. Based on the data collected for Fig. 4.4A, for each fixed o_j, we instead fit e_{ij} as a third-order polynomial in o_i to obtain the corresponding values for the derivative $\frac{\partial e_{ij}}{\partial o_i}$. The characterization of $\frac{\partial e_{ij}}{\partial o_i}$ occurs offline prior to the training process. In this approach, ST-RSBP can more precisely measure the differentiation of S-PSP w.r.t firing counts than HM2-BP [16]. Therefore, ST-RSBP may achieve better results even on feedforward networks like spiking CNNs.

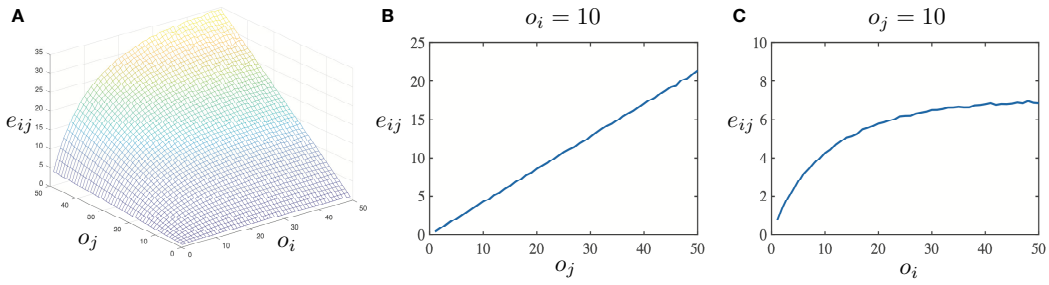

Figure 4.4: (A) The average S-PSP value vs. pre and post-synaptic firing counts; (B) The average e_{ij} vs. o_j when the post-synaptic firing count is fixed ($o_i = 10$); (C) The average e_{ij} vs. o_i when the pre-synaptic firing count is fixed ($o_j = 10$).

4.2.4 The Final Proposed ST-RSBP Algorithm

For each layer k, denote the error vector by $\boldsymbol{\delta}^k \in \mathbb{R}^{N_k}$, the matrix of differentiation of activation by $\boldsymbol{F}^{k,k-1} \in \mathbb{R}^{N_k \times N_{k-1}}$, and the weight matrix from the layer $k-1$ to layer k by $\boldsymbol{W}^{k,k-1} \in \mathbb{R}^{N_k \times N_{k-1}}$, respectively. $\boldsymbol{P}^{k+1,k} \in \mathbb{R}^{N_{k+1} \times N_k}$ contains all derivatives of $\frac{\partial a_l^{k+1}}{\partial a_i^k}$ obtained from (4.19) or (4.22). If the layer k is recurrent layer, we additionally use

$\boldsymbol{F}^{k,k} \in \mathbb{R}^{N_k \times N_k}$ and $\boldsymbol{W}^{k,k} \in \mathbb{R}^{N_k \times N_k}$ to denote the matrix of differentiation of activation and the weight matrix of recurrent connections within the layer k. Putting everything together, the complete ST-RSBP algorithm with a learning rate η is as follows:

$$\begin{cases} \Delta \boldsymbol{W}^{k,k-1} = \eta \frac{\nabla E}{\nabla \boldsymbol{W}^{k,k-1}} = \eta \cdot diag(\boldsymbol{\delta}^k) \cdot \boldsymbol{F}^{k,k-1} & \text{for feedforward connections} \\ \Delta \boldsymbol{W}^{k,k} = \eta \frac{\nabla E}{\nabla \boldsymbol{W}^{k,k}} = \eta \cdot diag(\boldsymbol{\delta}^k) \cdot \boldsymbol{F}^{k,k} & \text{for recurrent connections} \end{cases}$$

$$F_{ij}^{k,k-1} = \frac{e_{ij}^k}{1 - \frac{1}{\nu^k} \sum_l^{N_{k-1}} w_{il}^k \frac{\partial e_{il}^k}{\partial o_i^k}}$$
$$F_{ij}^{k,k} = \frac{e_{ij}^k}{1 - \frac{1}{\nu^k} \left(\sum_l^{N_{k-1}} w_{il}^k \frac{\partial e_{il}^k}{\partial o_i^k} + \sum_p^{N_k} w_{ip}^k \frac{\partial e_{ip}^k}{\partial o_i^k} \right)}$$

$$\begin{cases} \delta_i^k = \frac{o_i^k - y_i^k}{\nu^k} & \text{if layer k is the output} \\ \boldsymbol{\delta}^k = (\boldsymbol{P}^{k+1,k})^T \cdot \boldsymbol{\delta}^{k+1} & \text{if layer k+1 is feedforward} \\ \boldsymbol{\delta^k} = ((\boldsymbol{\Omega}^{k+1,k} - \boldsymbol{\Theta}^{k+1,k})^{-1} \cdot \boldsymbol{\Phi}^{k+1,k})^T \cdot \boldsymbol{\delta^{k+1}} & \text{if layer k+1 is recurrent} \end{cases}$$

$$\Omega_{ij}^{k+1,k} = \begin{cases} 1 - \frac{1}{\nu^{k+1}} \left(\sum_m^{N_k} w_{lm}^{k+1} \frac{\partial e_{lm}^{k+1}}{\partial o_l^{k+1}} + \sum_p^{N_{k+1}} w_{lp}^{k+1} \frac{\partial e_{lp}^{k+1}}{\partial o_l^{k+1}} \right) & i = j \\ 0 & i \neq j \end{cases}$$
$$\Phi_{ij}^{k+1,k} = w_{ij}^{k+1} \frac{1}{\nu^k} \frac{\partial e_{ij}^{k+1}}{\partial o_j^k} \qquad \Theta_{ij}^{k+1,k} = w_{ij}^{k+1} \frac{1}{\nu^{k+1}} \frac{\partial e_{ij}^{k+1}}{\partial o_j^{k+1}}$$
$$P_{ij}^{k+1,k} = \frac{\frac{1}{\nu^k} w_{ij}^{k+1} \frac{\partial e_{ij}^{k+1}}{\partial o_j^k}}{1 - \frac{1}{\nu^{k+1}} \sum_{p=1}^{N_k} w_{ip}^{k+1} \frac{\partial e_{ip}^{k+1}}{\partial o_i^{k+1}}}.$$

The application of ST-RSBP follows the typical backpropagation steps. First, the SNN is simulated layer-by-layer based on chosen synaptic/neural models such as the LIF model (2.1). Second, the firing counts of the output layer are compared with the desirable firing labels to compute the output error $\boldsymbol{\delta^k}$. After that, the error vector in the output layer is propagated backwards to determine the gradient, based on which both the recurrent synapses weights and the weights between layers are trained.

4.3 Experiments and Results

4.3.1 Experimental Settings

The experimented SNNs are based on the LIF model and weights are randomly initialized by following the uniform distribution $U[-1, 1]$. Fixed firing thresholds are used in the range of $5mV$ to $20mV$ depending on the layer. Exponential weight regularization [14], lateral inhibition in the output layer [14] and Adam [68] as the optimizer are adopted. The parameters like the desired output firing counts, thresholds and learning rates are empirically tuned. Table 4.1 lists the typical constant values adopted in the proposed ST-RSBP learning rule in our experiments. The simulation step size is set to 1 ms. The batch size is 1 which means ST-RSBP is applied after each training sample to update the weights. Each experiment reported below is repeated five times to obtain the mean and standard deviation (STD) of the accuracy.

Table 4.1: Parameters settings

Parameter	**Value**	**Parameter**	**Value**
Time Constant of Voltage τ_m	64 ms	Threshold ν	10 mV
Time Constant of Synapse τ_s	8 ms	Synaptic Time Delay	1 ms
Refractory Period	2 ms	Reset Membrane Voltage V_{reset}	0 mV
Batch Size	1	Learning Rate η	0.001

4.3.2 TI46-Alpha Speech Dataset

In this experiment, we use the full set of the TI46-ALpha which contains 4,142 and 6,628 spoken English examples in 26 classes for training and testing, respectively.

Table 4.2: Comparison of different SNN models on TI46-Alpha

Algorithm	Hidden Layers[a]	# Params	Mean	STD	Best
HM2-BP [16]	800	83,200	89.36%	0.30%	89.92%
HM2-BP [16]	400-400	201,600	89.83%	0.71%	90.60%
HM2-BP [16]	800-800	723,200	90.50%	0.45%	90.98%
Non-spiking BP[b] [69]	LSM: R2000	52,000			78%
ST-RSBP (this work)	R800	86,280	91.57%	0.20%	91.85%
ST-RSBP (this work)	400-R400-400	363,313	93.06%	0.21%	**93.35%**

[a] We show the number of neurons in each hidden layer. R represents recurrent layer.

[b] An LSM model. The state vector of the reservoir is used to train the single readout layer by BP.

Table 4.2 compares ST-RSBP with several other algorithms on TI46-Alpha. The result from [69] shows that only training the single readout layer of a recurrent LSM is inadequate for this challenging task, demonstrating the necessity of training all layers of a recurrent network using techniques such as ST-RSBP. ST-RSBP outperforms all other methods. In particular, ST-RSBP is able to train a three-hidden-layer RSNN with 363,313 weights to increase the accuracy from 90.98% to 93.35% when compared with the feedforward SNN with 723,200 weights trained by HM2-BP.

4.3.3 N-TIDIGITS Neuromorphic Speech Dataset

In this experiment, the full N-TIDIGITS dataset is applied. Table 4.3 shows that proposed ST-RSBP achieves excellent accuracies up to 93.90%, which is significantly better than that of HM2-BP and the non-spiking GRN and LSTM in [61]. With a similar/less number of tunable weights, ST-RSBP outperforms all other methods rather significantly.

Table 4.3: Comparison of different models on N-TIDIGITS

Algorithm	Hidden Layers	# Params	Mean	STD	Best
HM2-BP [16]	250-250	81,250			89.69%
GRN (NS[a]) [61]	2× G200-100[b]	109,200			90.90%
Phased-LSTM (NS) [61]	2× 250L[c]	610,500			91.25%
ST-RSBP (this work)	250-R250	82,050	92.94%	0.20%	93.13%
ST-RSBP (this work)	400-R400-400	351,241	93.63%	0.27%	**93.90%**

[a]NS represents non-spiking algorithm; [b]G represents a GRN layer; [c]L represents an LSTM layer.

4.3.4 Spiking Convolution Neural Networks for the MNIST

ST-RSBP can more precisely compute gradients error than HM2-BP even for the case of feedforward CNNs. We demonstrate the performance improvement of ST-RSBP over several other state-of-the-art SNN BP algorithms based on spiking CNNs using the MNIST dataset. The spiking CNN trained by ST-RSBP consists of two 5×5 convolutional layers with a stride of 1, each followed by a 2×2 pooling layer, one fully connected hidden layer and an output layer for classification. In the pooling layer, each neuron connects to 2×2 neurons in the preceding convolutional layer with a fixed weight of 0.25. In addition, we use elastic distortion [70] for data augmentation which is similar to [14, 15, 16]. In Table 4.4, we compare the results of the proposed ST-RSBP with other BP rules on similar network settings. It shows that ST-RSBP can achieve an accuracy of 99.62%, surpassing the best previously reported performance [16] with the same model complexity.

4.4 Summary and Discussions

In this chapter, we present the novel spike-train level backpropagation algorithm ST-RSBP, which can transparently train all types of SNNs including RSNNs without unfolding in time. The employed S-PSP model improves the training efficiency at the

Table 4.4: Performances of Spiking CNNs on MNIST

Algorithm	Hidden Layers	Mean	STD	Best
Spiking CNN [14]	20C5-P2-50C5-P2-200[a]			99.31%
STBP [15]	15C5-P2-40C5-P2-300			99.42%
SLAYER [17]	12C5-p2-64C5-p2	99.36%	0.05%	99.41%
HM2-BP [16]	15C5-P2-40C5-P2-300	99.42%	0.11%	99.49%
ST-RSBP (this work)	12C5-p2-64C5-p2	99.50%	0.03%	99.53%
ST-RSBP (this work)	15C5-P2-40C5-P2-300	99.57%	0.04%	**99.62%**

[a] 20C5: convolution layer with 20 of the 5×5 filters. P2: pooling layer with 2×2 filters.

spike-train level and also addresses key challenges of RSNNs training in handling temporal effects and gradient computation of loss functions with inherent discontinuities for accurate gradient computation. The spike-train level processing for RSNNs is the starting point for ST-RSBP. After that, we have applied the standard BP principle while dealing with specific issues of derivative computation at the spike-train level.

More specifically, in ST-RSBP, the given rate-coded errors can be efficiently computed and back-propagated through layers without costly unfolding the network in time and through expensive time point by time point computation. Moreover, ST-RSBP handles the discontinuity of spikes during BP without altering and smoothing the microscopic spiking behaviors. The problem of network unfolding is dealt with accurate spike-train level BP such that the effect of all spikes is captured and propagated in an aggregated manner to achieve accurate and fast training. As such, both rate and temporal information in the SNN are well exploited during the training process.

Using the efficient GPU implementation of ST-RSBP, we demonstrate the best performances for both feedforward SNNs, RSNNs, and spiking CNNs over the speech datasets TI46-Alpha, and N-TIDIGITS and the image dataset MNIST, outperforming the current state-of-the-art SNN training techniques. Moreover, ST-RSBP outperforms conventional ANN models like LSTM, GRN, and traditional non-spiking BP on the same datasets.

We expect this work would advance the research on spiking neural networks and neuromorphic computing.

Chapter 5

Temporal Spike Sequence Learning via Backpropagation

In Chapter 4, we propose ST-RSBP which can handle temporal effects and gradient computation of loss functions with inherent discontinuities for accurate gradient computation. However, most existing SNN error backpropagation (BP) methods lack proper handling of spiking discontinuities and suffer from low performance compared with the BP methods for traditional artificial neural networks. In addition, a large number of time steps are typically required, including the proposed ST-RSBP method, to achieve decent performance, leading to high latency and rendering spike-based computation unscalable to deep architectures. We present a novel Temporal Spike Sequence Learning Backpropagation (TSSL-BP) method for training deep SNNs, which breaks down error backpropagation across two types of inter-neuron and intra-neuron dependencies and leads to improved temporal learning precision. It captures inter-neuron dependencies through presynaptic firing times by considering the all-or-none characteristics of firing activities and captures intra-neuron dependencies by handling the internal evolution of each neuronal state in time. TSSL-BP efficiently trains deep SNNs within a much short-

ened temporal window of a few steps while improving the accuracy for various image classification datasets including CIFAR10.

5.1 Forward Pass

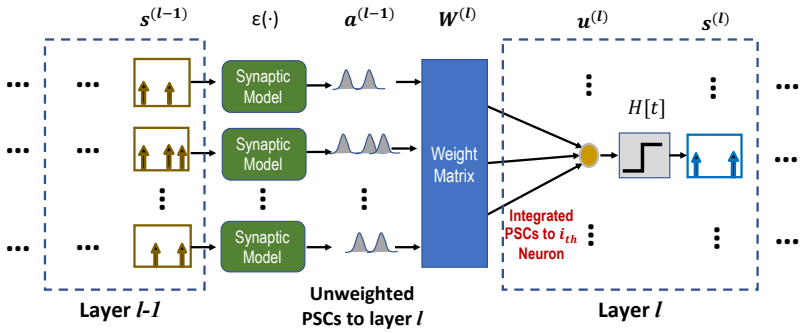

Figure 5.1: Forward evaluation pass of SNNs.

Without loss of generality, we consider performing BP across two adjacent layers $l-1$ and l with N_{l-1} and N_l neurons, respectively, in a fully-connected feedforward SNNs as shown in Figure 5.1. The procedure can be also applied to convolutional and pooling layers. Denote the presynaptic weights by $\boldsymbol{W}^{(l)} = \left[\boldsymbol{w}_1^{(l)}, \cdots, \boldsymbol{w}_{N_l}^{(l)}\right]^T$, where $\boldsymbol{w}_i^{(l)}$ is a column vector of weights from all the neurons in layer $l-1$ to the neuron i of layer l. In addition, we also denote PSCs from neurons in layer $l-1$ by $\boldsymbol{a}^{(l-1)}[t] = \left[a_1^{(l-1)}[t], \cdots, a_{N_{l-1}}^{(l-1)}[t]\right]^T$, spike trains output of the $l-1$ layer by $\boldsymbol{s}^{(l-1)}[t] = \left[s_1^{(l-1)}[t], \cdots, s_{N_{l-1}}^{(l-1)}[t]\right]^T$, membrane potentials and the corresponding output spike trains of the l layer neurons respectively by $\boldsymbol{u}^{(l)}[t] = \left[u_1^{(l)}[t], \cdots, u_{N_l}^{(l)}[t]\right]^T$ and $\boldsymbol{s}^{(l)}[t] = \left[s_1^{(l)}[t], \cdots, s_{N_l}^{(l)}[t]\right]^T$, where variables associated with neurons in the layer l have l as the superscript.

From the discretized LIF neuron model (2.4), the forward propagation between the

two layers is described as

$$\begin{aligned}
&\boldsymbol{a}^{(l-1)}[t] = (\epsilon * \boldsymbol{s}^{(l-1)})[t], \\
&\boldsymbol{u}^{(l)}[t] = (1 - \frac{1}{\tau_m})\boldsymbol{u}^{(l)}[t-1] + \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)}[t] + (\nu * \boldsymbol{s}^{(l)})[t], \\
&\boldsymbol{s}^{(l)}[t] = H\left(\boldsymbol{u}^{(l)}[t] - V_{th}\right).
\end{aligned} \tag{5.1}$$

In the forward pass, the spike trains $\boldsymbol{s}^{(l-1)}[t]$ of the $l-1$ layer generate the (unweighted) PSCs $\boldsymbol{a}^{(l-1)}[t]$ according to the synaptic model. Then, $\boldsymbol{a}^{(l-1)}[t]$ are multiplied the synaptic weights and passed onto the neurons of layer l. The integrated PSCs alter the membrane potentials and trigger the output spikes of the layer l neurons when the membrane potentials exceed the threshold.

5.2 Backward Pass

5.2.1 The Loss Function

The goal of the proposed TSSL-BP method is to train a given SNN in such a way that each output neuron learns to produce a desired firing sequence arbitrarily specified by the user according to the input class label. Denote the desired and the actual spike trains in the output layer by $\boldsymbol{d} = [\boldsymbol{d}[t_0], \cdots, \boldsymbol{d}[t_{N_t}]]$ and $\boldsymbol{s} = [\boldsymbol{s}[t_0], \cdots, \boldsymbol{s}[t_{N_t}]]$ where N_t is the number of the considered time steps, $\boldsymbol{d}[t]$ and $\boldsymbol{s}[t]$ the desired and actual firing events for all output neurons at time t, respectively.

The loss function L can be defined using any suitable distance function measuring the difference between $\boldsymbol{d}$ and $\boldsymbol{s}$ by the square error for each output neuron at each time step:

$$L = \sum_{k=0}^{N_t} E[t_k] = \frac{1}{2}\sum_{k=0}^{N_t}(\boldsymbol{d}[t_k] - \boldsymbol{s}[t_k])^2 = \frac{1}{2}||\boldsymbol{d} - \boldsymbol{s}||_2^2, \tag{5.2}$$

where $E[t]$ is the error at time t.

In this work, we use the spike response kernel, defining the error at each time step as

$$E[t] = \frac{1}{2}((\epsilon * \boldsymbol{d})[t] - (\epsilon * \boldsymbol{s})[t])^2 = \frac{1}{2}(\boldsymbol{a_d}[t] - \boldsymbol{a_s}[t])^2. \tag{5.3}$$

$\epsilon(\cdot)$ is a kernel function which measures the so-called Van Rossum distance between the actual spike train and desired spike train.

5.2.2 Backpropagation Flow

From the loss function (5.3), we define the error $E[t_k]$ at each time step. $E[t_k]$ is based on the output layer firing spikes at t_k which further depend on all neuron states $\mathbf{u}[t]$, $t \leq t_k$.

We adopt (5.3) to define the total loss

$$L = \sum_{k=0}^{N_t} E[t_k] = \frac{1}{2}\sum_{k=0}^{N_t}(\boldsymbol{a_d}[t_k] - \boldsymbol{a_s}[t_k])^2. \tag{5.4}$$

For the neurons in layer l, the error gradient with respect to the presynaptic weights matrix $\boldsymbol{W}^{(l)}$ is

$$\frac{\partial L}{\partial \boldsymbol{W}^{(l)}} = \sum_{k=0}^{N_t} \frac{\partial E[t_k]}{\partial \boldsymbol{W}^{(l)}}. \tag{5.5}$$

(5.1) reveals that the values of $\boldsymbol{u}^{(l)}$ at time t_k have contribution to all future fires and losses. Using the chain rule, we get

$$\frac{\partial L}{\partial \boldsymbol{W}^{(l)}} = \sum_{k=0}^{N_t}\sum_{m=0}^{k} \frac{\partial E[t_k]}{\partial \boldsymbol{u}^{(l)}[t_m]} \frac{\partial \boldsymbol{u}^{(l)}[t_m]}{\partial \boldsymbol{W}^{(l)}}. \tag{5.6}$$

By changing the order of summation, (5.6) can be written as

$$\frac{\partial L}{\partial \boldsymbol{W}^{(l)}} = \sum_{m=0}^{N_t} \frac{\partial \boldsymbol{u}^{(l)}[t_m]}{\partial \boldsymbol{W}^{(l)}} \sum_{k=m}^{N_t} \frac{\partial E[t_k]}{\partial \boldsymbol{u}^{(l)}[t_m]} = \sum_{m=0}^{N_t} \boldsymbol{a}^{(l-1)}[t_m] \sum_{k=m}^{N_t} \frac{\partial E[t_k]}{\partial \boldsymbol{u}^{(l)}[t_m]}. \tag{5.7}$$

We use δ to denote the back propagated error at time t_m as $\boldsymbol{\delta}^{(l)}[t_m] = \sum_{k=m}^{N_t} \frac{\partial E[t_k]}{\partial \boldsymbol{u}^{(l)}[t_m]}$. Therefore, the weights update formula (5.7) can be written as

$$\frac{\partial L}{\partial \boldsymbol{W}^{(l)}} = \sum_{m=0}^{N_t} \boldsymbol{a}^{(l-1)}[t_m] \boldsymbol{\delta}^{(l)}[t_m]. \tag{5.8}$$

where $\boldsymbol{a}^{(l-1)}[t_m]$ is analogous to the pre-activation in the traditional ANNs which can be easily obtained from (5.1) in the forward pass. $\boldsymbol{\delta}^{(l)}[t_m]$ is considered in two cases.

[l **is the output layer.**] The $\boldsymbol{\delta}^{(l)}[t_m]$ can be computed from

$$\boldsymbol{\delta}^{(l)}[t_m] = \sum_{k=m}^{N_t} \frac{\partial E[t_k]}{\partial \boldsymbol{a}^{(l)}[t_k]} \frac{\partial \boldsymbol{a}^{(l)}[t_k]}{\partial \boldsymbol{u}^{(l)}[t_m]}. \tag{5.9}$$

From (5.4), the first term of (5.9) is given by

$$\frac{\partial E[t_k]}{\partial \boldsymbol{a}^{(l)}[t_k]} = \frac{1}{2} \frac{\partial (\boldsymbol{a_d}[t_k] - \boldsymbol{a}^{(l)}[t_k])^2}{\partial \boldsymbol{a}^{(l)}[t_k]} = \boldsymbol{a}^{(l)}[t_k] - \boldsymbol{a_d}[t_k]. \tag{5.10}$$

[l **is a hidden layer.**] $\boldsymbol{\delta}^{(l)}[t_m]$ is derived using the chain rule and (5.1).

$$\boldsymbol{\delta}^{(l)}[t_m] = \sum_{j=m}^{N_t} \sum_{k=m}^{j} \frac{\partial \boldsymbol{a}^{(l)}[t_k]}{\partial \boldsymbol{u}^{(l)}[t_m]} \left(\frac{\partial \boldsymbol{u}^{(l+1)}[t_k]}{\partial \boldsymbol{a}^{(l)}[t_k]} \frac{\partial E[t_j]}{\partial \boldsymbol{u}^{(l+1)}[t_k]} \right). \tag{5.11}$$

It is obtained from the fact that membrane potentials $\boldsymbol{u}^{(l)}$ of the neurons in layer l influence their (unweighted) corresponding postsynaptic currents (PSCs) $\boldsymbol{a}^{(l)}$ through fired spikes, and $\boldsymbol{a}^{(l)}$ further affect the membrane potentials $\boldsymbol{u}^{(l+1)}$ in the next layer.By

changing the order of summation, maps the error $\boldsymbol{\delta}$ from layer $l+1$ to layer l.

$$\begin{aligned}\boldsymbol{\delta}^{(l)}[t_m] &= \sum_{k=m}^{N_t} \frac{\partial \boldsymbol{a}^{(l)}[t_k]}{\partial \boldsymbol{u}^{(l)}[t_m]} \sum_{j=k}^{N_t} \frac{\partial \boldsymbol{u}^{(l+1)}[t_k]}{\partial \boldsymbol{a}^{(l)}[t_k]} \frac{\partial E[t_j]}{\partial \boldsymbol{u}^{(l+1)}[t_k]} \\ &= \sum_{k=m}^{N_t} \frac{\partial \boldsymbol{a}^{(l)}[t_k]}{\partial \boldsymbol{u}^{(l)}[t_m]} \sum_{j=k}^{N_t} \boldsymbol{W}^{(l+1)} \frac{\partial E[t_j]}{\partial \boldsymbol{u}^{(l+1)}[t_k]} \\ &= \sum_{k=m}^{N_t} \frac{\partial \boldsymbol{a}^{(l)}[t_k]}{\partial \boldsymbol{u}^{(l)}[t_m]} (\boldsymbol{W}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)}[t_k] \\ &= (\boldsymbol{W}^{(l+1)})^T \sum_{k=m}^{N_t} \frac{\partial \boldsymbol{a}^{(l)}[t_k]}{\partial \boldsymbol{u}^{(l)}[t_m]} \boldsymbol{\delta}^{(l+1)}[t_k].\end{aligned} \tag{5.12}$$

As shown above, for both the output layer and hidden layers, once $\frac{\partial \boldsymbol{a}^{(l)}[t_k]}{\partial \boldsymbol{u}^{(l)}[t_m]}(t_k \geq t_m)$ are known, the error $\boldsymbol{\delta}$ can be back propagated and the gradient of each layer can be calculated. In the next section, we analyze the main difficulties and derive the TSSL-BP method to handle these challenges.

5.3 TSSL-BP Method

5.3.1 Key challenges in SNN BackPropagation

The dependencies of the PSCs on the corresponding membrane potentials of the presynaptic neurons reflected in $\frac{\partial \boldsymbol{a}^{(l)}[t_k]}{\partial \boldsymbol{u}^{(l)}[t_m]}(t_k \geq t_m)$ are due to the following spiking neural behaviors: a change in the membrane potential may bring it up to the firing threshold, and hence activate the corresponding neuron by generating a spike, which in turn produces a PSC. Computing $\frac{\partial \boldsymbol{a}^{(l)}[t_k]}{\partial \boldsymbol{u}^{(l)}[t_m]}(t_k \geq t_m)$ involves the activation of each neuron, i.e. firing a spike due to the membrane potential's crossing the firing threshold from below. Unfortunately, the all-or-none firing characteristics of spiking neurons makes the

activation function nondifferentiable, introducing several key challenges.

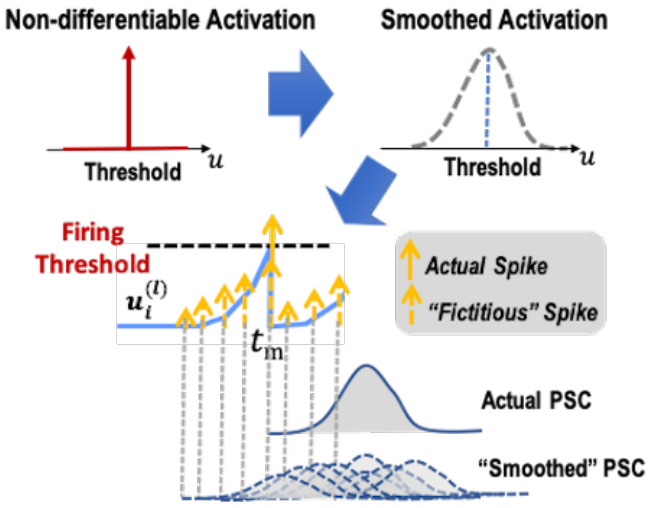

Figure 5.2: "Fictitious" smoothing of activation.

A typical strategy in dealing with the non-differentiability of the activation is to smooth the activation function by approximating it using a differentiable curve [15] as shown in the Figure 5.2, or a continuous probability density function [17], which is similar to the former approach in spirit. However, these approaches effectively spread each discrete firing spike continuously over time, converting one actual spike to multiple "fictitious" spikes and also generating multiple "fictitious" PSCs displaced at different time points. We stress that while smoothing circumvents the numerical challenges brought by non-differentiability of the spiking activation, it effectively alters the underlying spiking neuron model and firing times, and leads to degraded accuracy in the error gradient computation. It is important to reflect that spike timing is the hallmark of spiking neural computation, altering firing times in BP can hamper precise learning of the targeted firing sequences as pursued in this paper.

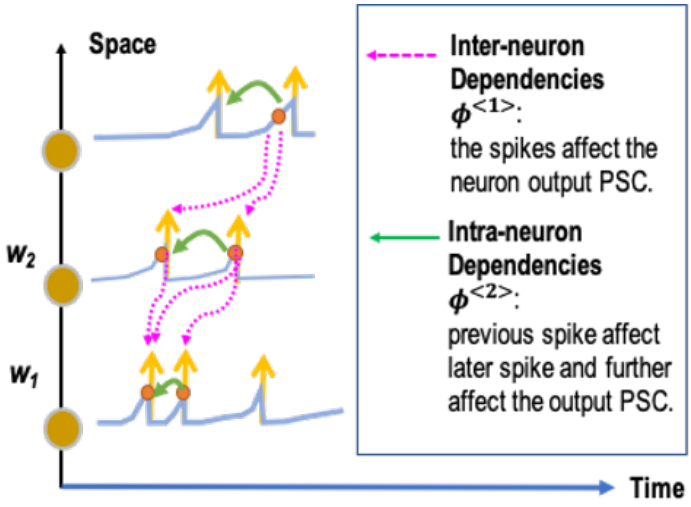

Figure 5.3: Inter/Intra neuron dependencies.

5.3.2 The Main Ideas Behind TSSL-BP

TSSL-BP addresses the two key limitations of the prior BP methods: lack proper handling of spiking discontinuities (leading to loss of temporal precision) and need for many time steps (i.e. high latency) to ensure good performance. TSSL-BP computes $\frac{\partial \boldsymbol{a}^{(l)}[t_k]}{\partial \boldsymbol{u}^{(l)}[t_m]}$ across two categories of spatio-temporal dependencies in the network: **inter-neuron** and **intra-neuron** dependencies. As shown in Figure 5.3, our key observations are: 1) temporal dependencies of a postsynaptic neuron on any of its presynaptic neurons *only* take place via the presynaptic spikes which generate PSCs to the postsynaptic neuron, and shall be considered as inter-neuron dependencies; 2) furthermore, the timing of one presynaptic spike affects the timing of the immediately succeeding spike from the same presynaptic neuron through the intra-neuron temporal dependency. The timing of the first presynaptic spike affects the PSC produced by the second spike, and has additional impact on the postynaptic neuron through this indirect mechanism. In the following, we show how to derive the $\frac{\partial a_i^{(l)}[t_k]}{\partial u_i^{(l)}[t_m]}$ for each neuron i in layer l. We denote $\phi_i^{(l)}(t_k, t_m) = \frac{\partial a_i^{(l)}[t_k]}{\partial u_i^{(l)}[t_m]} = \phi_i^{(l)<1>}(t_k, t_m) + \phi_i^{(l)<2>}(t_k, t_m)$, where $\phi_i^{(l)<1>}(t_k, t_m)$ represents

the inter-neuron dependencies and $\phi_i^{(l)<2>}(t_k, t_m)$ is the intra-neuron dependencies.

5.3.3 Inter-Neuron Backpropagation

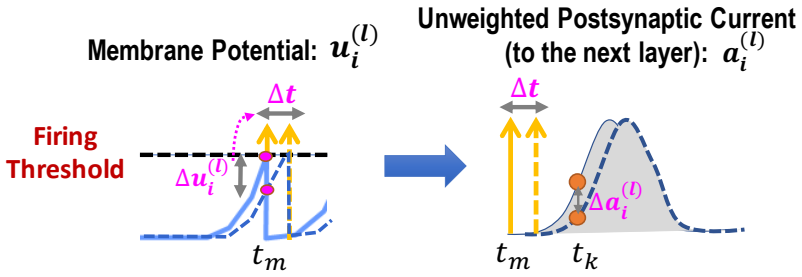

Figure 5.4: PSC dependencies on presynaptic potential.

Instead of performing the problematic activation smoothing, we critically note that the all-or-none characteristics of firing behavior is such that a PSC waveform is only triggered at a presynaptic firing time. Specially, as shown in Figure 5.4, a perturbation $\Delta u_i^{(l)}$ of $u_i^{(l)}[t_m]$, i.e, due to weight updates, may result in an incremental shift in the firing time Δt, which in turn shifts the onset of the PSC waveform corresponding to the shifted spike, leading to a perturbation $\Delta a_i^{(l)}$ of $a_i^{(l)}[t_k]$. We consider this as an inter-neuron dependency since the change in PSC ($\Delta a_i^{(l)}$) alters the membrane potential of the postsynaptic neuron in the next layer.

We make two important points: 1) we shall capture the inter-neuron dependencies via (the incremental changes of) the presynaptic firing times, which precisely corresponds to how different neurons interact with each other in an SNN; and 2) the *inter-neuron* dependency of each neuron i's PSC $a_i^{(l)}$ at t_k on its membrane potential $u_i^{(l)}$ at t_m happens only if the neuron fires at t_m. In general, $a_i^{(l)}[t_k]$'s inter-neuron dependencies on *all* preceding firing times shall be considered. Figure 5.5 shows the situation where $a_i^{(l)}[t_k]$ depends on two presynaptic firing times t_m and t_p. Conversely, the inter-neuron dependencies $\phi_i^{(l)<1>}(t_k, t_m) = 0$ if $t_k < t_m$ or there is no spike at t_m. Assuming that the presynaptic

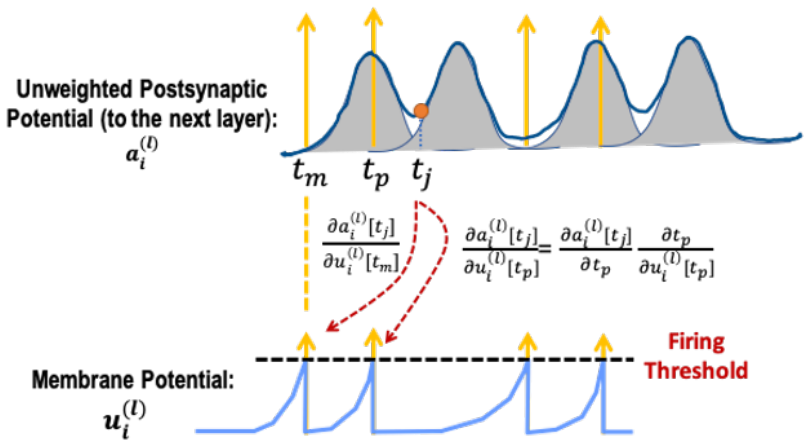

Figure 5.5: Inter-neuron dependencies.

neuron i spikes at t_m, The inter-neuron dependencies is

$$\phi_i^{(l)<1>}(t_k, t_m) = \frac{\partial a_i^{(l)}[t_k]}{\partial t_m} \frac{\partial t_m}{\partial u_i^{(l)}[t_m]}, \tag{5.13}$$

where, importantly, the chain rule is applied through the presynaptic firing time t_m.

From (2.2), the two parts part of (5.13) can be calculated as

$$\frac{\partial a_i^{(l)}[t_k]}{\partial t_m} = \frac{\partial(\epsilon * s_i^{(l)}[t_m])[t_k]}{\partial t_m}, \qquad \frac{\partial t_m}{\partial u_i^{(l)}[t_m]} = \frac{-1}{\frac{\partial u_i^{(l)}[t_m]}{\partial t_m}}, \tag{5.14}$$

where $\frac{\partial u_i^{(l)}[t_m]}{\partial t_m}$ is obtained by differentiating (2.4).

5.3.4 Intra-Neuron Backpropagation

Now we consider the intra-neuron dependency $\phi_i^{(l)<2>}(t_k, t_m)$ defined between an arbitrary time t_k and a presynaptic firing time t_m ($t_m < t_k$). From Section 5.3.3, the presynpatic firing at time t_m invokes a continuous PSC which has a direct impact on the postsynaptic potential at time t_k, which is an inter-neuron dependency. On the other hand, $\phi_i^{(l)<2>}(t_k, t_m)$ corresponds to an indirect effect on the postsynaptic potential via

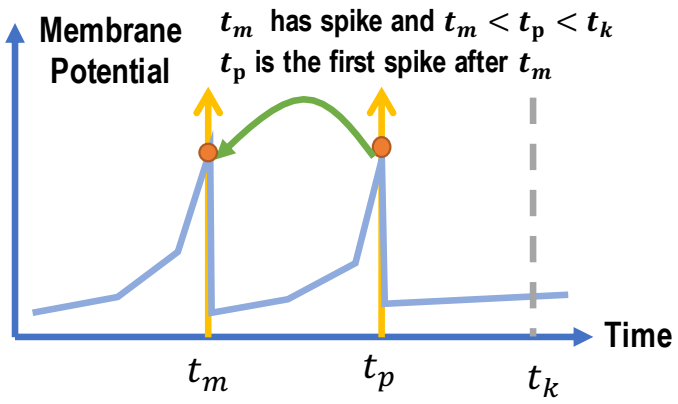

Figure 5.6: Intra-neuron dependencies.

the intra-neuron dependency.

We consider $\phi_i^{(l)<2>}(t_k, t_m)$ specifically under the context of the adopted LIF model, which may occur if the presynaptic neuron spikes at t_p immediately following t_m such that $t_m < t_p < t_k$. In this case, the presynatpic membrane potential at t_m not only contributes to a PSC due to the neuron firing, but also affects the membrane potential at the next spike time t_p resulted from the reset occurring at t_m as described in (2.4). The PSC $a_i^{(l)}[t_k]$ has an inter-neuron dependency on membrane potential $u_i^{(l)}[t_p]$ while $u_i^{(l)}[t_p]$ is further affected by the immediately preceding firing time t_m due to the reset of the presynaptic potential at t_m. Recall $s_i^{(l)}[t] = 1$ if neuron i fires at t as in (2.5). More precisely, $\phi_i^{(l)<2>}(t_k, t_m)$ takes this indirect intra-neuron effect on $a_i^{(l)}[t_k]$ into consideration if $\exists t_p \in (t_m, t_k)$ such that $s_i^{(l)}[t_p] = 1$ and $s_i^{(l)}[t] = 0\ \forall t \in (t_m, t_p)$, i.e. no other presynaptic spike exists between t_m and t_p

$$\begin{aligned}\phi_i^{(l)<2>}(t_k, t_m) &= \frac{\partial a_i^{(l)}[t_k]}{\partial u_i^{(l)}[t_p]} \frac{\partial u_i^{(l)}[t_p]}{\partial t_m} \frac{\partial t_m}{\partial u_i^{(l)}[t_m]} \\ &= \phi_i^{(l)}(t_k, t_p) \frac{\partial (\nu * s_i^{(l)}[t_m])[t_p]}{\partial t_m} \frac{\partial t_m}{\partial u_i^{(l)}[t_m]},\end{aligned} \tag{5.15}$$

where $\nu(\cdot)$ is the reset kernel and $\frac{\partial t_m}{\partial u_i^{(l)}[t_m]}$ is evaluated by (5.14). In (5.15), $\phi_i^{(l)}(t_k, t_p)$ would have been already computed during the backpropagation process since t_p is a presynaptic firing time after t_m.

5.3.5 Final TSSL-BP Rule

To sum it up, we obtain the derivative of loss with respect to weight according to TSSL-BP method as follows:

$$
\begin{aligned}
&\frac{\partial L}{\partial \boldsymbol{W}^{(l)}} = \sum_{m=0}^{N_t} \boldsymbol{a}^{(l-1)}[t_m]\boldsymbol{\delta}^{(l)}[t_m], \\
&\boldsymbol{\delta}^{(l)}[t_m] = \begin{cases} \sum_{k=m}^{N_t} (\boldsymbol{a}^{(l)}[t_k] - \boldsymbol{a_d}[t_k])\phi_i^{(l)}(t_k, t_m) & \text{for output layer,} \\ (\boldsymbol{W}^{(l+1)})^T \sum_{k=m}^{N_t} \phi_i^{(l)}(t_k, t_m)\boldsymbol{\delta}^{(l+1)}[t_k] & \text{for hidden layers,} \end{cases} \\
&\phi_i^{(l)}(t_k, t_m) = \\
&\begin{cases} 0 & s_i^{(l)}[t_m] = 0, s_i^{(l)}[t_p] = 0 \ \forall t_p \in (t_m, t_k), \\ \frac{\partial a_i^{(l)}[t_k]}{\partial t_m} \frac{\partial t_m}{\partial u_i^{(l)}[t_m]} & s_i^{(l)}[t_m] = 1, s_i^{(l)}[t_p] = 0 \ \forall t_p \in (t_m, t_k), \\ \frac{\partial a_i^{(l)}[t_k]}{\partial t_m} \frac{\partial t_m}{\partial u_i^{(l)}[t_m]} + \phi_i^{(l)<2>}(t_k, t_m) & s_i^{(l)}[t_m] = 1, \exists t_p \text{ such that} \\ & s_i^{(l)}[t_p] = 1, s_i^{(l)}[t] = 0 \ \forall t \in (t_m, t_p). \end{cases}
\end{aligned}
\tag{5.16}
$$

There are two key distinctions setting our approach apart from the aforementioned activation smoothing. First, the inter-neuron dependencies are only considered at presynaptic firing times as opposed to all prior time points, latter of which is the case when the activation smoothing is applied with BPTT. The handling adopted in TSSL-BP is a manifestation of the all-or-none firing characteristics of spiking neurons. Second, as in Figure 5.4, the key step in backpropagation is the consideration of the incremental change of spiking times, which is not considered in recent SNNs BP works.

5.4 Experiments and Results

In this section, we test the proposed TSSL-BP method on four datasets MNIST [53], N-MNIST [59], FashionMNIST [54] and CIFAR10 [55]. We first explain experimental settings and the details of practical simulation issues. Then, we compare TSSL-BP with several state-of-the-art results with the same or similar network sizes including different SNNs BP methods, converted SNNs, and traditional AfNNs.

5.4.1 Experimental Settings

The experimented SNNs are based on the LIF model described in (2.4). The simulation step size is set to 1 ms. Only a few time steps are used to demonstrate low-latency spiking neural computation. The parameters like thresholds and learning rates are empirically tuned. Table 5.1 lists the typical constant values adopted in our experiments. For the time constant, we vary the membrane time constant from 2ms to 16ms. The same performance level has been observed. This indicates that the proposed method can train SNNs with dynamical behaviors across different timescales and the empirically observed results are not very sensitive to the choice of membrane time constant. No axon and synaptic delay or refractory period is used nor is normalization. Dropout is only applied for the experiments on CIFAR10. Adam [68] is adopted as the optimizer. The network models we train or compare with are either fully connected feedforward networks or convolutional neural networks (CNNs). The mean and standard deviation (STD) of the accuracy reported are obtained by repeating the experiments five times.

5.4.2 Handling of Practical Issues

Two practical circumstances need to be taken into consideration as for other spike-time based BP methods like SpikeProp [13, 71]. First, when a spike is produced by the

Table 5.1: Parameters settings.

Parameter	**Value**	**Parameter**	**Value**
simulation step size	1 ms	Learning Rate η	0.005
Time Constant of Voltage τ_m	4 ms	Time Constant of Synapse τ_s	2 ms
Threshold V_{th}	1 mV	Batch Size	50

membrane potential $u[t]$ that barely reaches the threshold, the derivative of $u[t]$ w.r.t time is very small. Numerically, this can make (5.14) large and result in an undesirable large weight update. To mitigate, we set a bound for this derivative. Second, absence of firing activities in spiking neurons due to low initial weight values block backpropagation through these neurons. We use a warm-up mechanism to bring up the firing activity of the network before applying the BP method. In the warm-up mechanism, we detect the firing events of each neuron. If there's at least one spike within a certain time window, TSSL-BP is applied directly. Otherwise, warm-up is applied, which uses the continuous sigmoid function of membrane potential to approximate the activation function so that the error can be propagated back even when there is no spike.

The desired output spike trains (labels) for different classes are manually selected without much optimization effort. In the experiments with 5 time steps, we set two fixed sequences [0, 1, 0, 1, 1] and [0, 0, 0, 0, 0] where 1 represents a spike and 0 means no spike at a given time step. We adopted a simple scheme: the number of output neurons is same as the number of classes. For each class, the first sequence is chosen to be the target of one (distinct) neuron corresponding to the class, and the second sequence is targeted for all other output neurons.

5.4.3 MNIST

On MNIST [53], we compares the accuracies of the spiking CNNs trained by the TSSL-BP method with ones trained by other algorithms in Table 5.2. In our method, the pixel intensities of the image are converted into real-valued spike current to the inputs within a short time window. The proposed TSSL-BP delivers up to 99.53% accuracy and outperforms all other methods except for the ST-RSBP which we proposed in Section 4. However, compared to ST-RSBP, TSSL-BP can train high-performance SNNs with only 5 time steps, achieving 80× reduction of step count (latency). The accuracy of ST-RSBP drops below that of TSSL-BP noticeably under short time windows. In addition, no data augmentation is applied in this experiment, which is adopted in HM2BP [16] and ST-RSBP.

Table 5.2: Performances of Spiking CNNs on MNIST.

Methods	Network	Time steps	Mean	STD	Best
Spiking CNN [14]	20C5-P2-50C5-P2-200	> 200			99.31%
STBP [15]	15C5-P2-40C5-P2-300	> 100			99.42%
SLAYER [17]	12C5-p2-64C5-p2	300	99.36%	0.05%	99.41%
HM2BP [16]	15C5-P2-40C5-P2-300	400	99.42%	0.11%	99.49%
ST-RSBP	15C5-P2-40C5-P2-300	400	99.57%	0.04%	99.62%
This work	15C5-P2-40C5-P2-300	**5**	99.50%	0.02%	99.53%

20C5: convolution layer with 20 of the 5×5 filters. P2: pooling layer with 2×2 filters.

5.4.4 N-MNIST

We test the proposed method on N-MNIST dataset [59], the neuromorphic version of the MNIST. The inputs to the networks are spikes rather than real value currents. Table 5.3 compares the results obtained by different models on N-MNIST. The SNN trained by our proposed approach naturally processes spatio-temporal spike patterns,

achieving the start-of-the-art accuracy of 99.40%. It is important to note that our proposed method with the accuracy of 99.28% outperforms the best previously reported results in [17], obtaining 10 times fewer time steps which leads to significant latency reduction.

Table 5.3: Performances on N-MNIST.

Methods	Network	Time steps	Mean	STD	Best
HM2BP [16]	$400-400$	600	98.88%	0.02%	98.88%
SLAYER [17]	$500-500$	300	98.89%	0.06%	98.95%
SLAYER [17]	12C5-P2-64C5-P2	300	99.20%	0.02%	99.22%
This work	12C5-P2-64C5-P2	100	**99.35**%	0.03%	**99.40**%
This work	12C5-P2-64C5-P2	**30**	99.23%	0.05%	99.28%

All the experiments in this table train the N-MNIST for 100 epochs

5.4.5 FashionMNIST

We compare several trained fully-connected feedforward SNNs and spiking CNNs on FashionMNIST [54], a more challenging dataset than MNIST. In Table 5.4, TSSL-BP achieves 89.80% test accuracy on the fully-connected feedforward network of two hidden layers with each having 400 neurons, outperforming the HM2BP method of [16], which is the best previously reported algorithm for feedforward SNNs. TSSL-BP can also deliver the best test accuracy with much fewer time steps. Moreover, TSSL-BP achieves **92.83**% on the spiking CNN networks, noticeably outperforming the same size non-spiking CNN trained by a standard BP method.

5.4.6 CIFAR10

Furthermore, we apply the proposed method on the more challenging dataset of CIFAR10 [55]. As shown in Table 5.5, our TSSL-BP method achieves 89.22% accuracy with

Table 5.4: Performances on FashionMNIST.

Methods	Network	Time steps	Mean	STD	Best
ANN	$400-400$				89.01%
HM2BP [16]	$400-400$	400			88.99%
This work	$400-400$	**5**	89.75%	0.03%	**89.80%**
ANN [54]	32C5-P2-64C5-P2-1024				91.60%
This work	32C5-P2-64C5-P2-1024	5	92.69%	0.09%	**92.83%**

a mean of 88.98% and a standard deviation of 0.27% under five trials on the first CNN and achieves 91.41% accuracy on the second CNN architecture. TSSL-BP delivers the best result among a previously reported ANN, SNNs converted from pre-trained ANNs, and the spiking CNNs trained by the STBP method of [72]. CIFAR10 is a challenging dataset for most of the existing SNNs BP methods since the long latency required by those methods makes them hard to scale to deeper networks. The proposed TSSL-BP not only achieves up to 3.98% accuracy improvement over the work of [72] without the additional optimization techniques including neuron normalization and population decoding which are employed in [72], but also utilizes fewer time steps.

Table 5.5: Performances of CNNs on CIFAR10.

Methods	Network	Time steps	Epochs	Accuracy
Converted SNN [73]	CNN 1	80		83.52%
STBP [72]	CNN 1	8	150	85.24%
This work	CNN 1	**5**	150	**89.22%**
ANN [72]	CNN 2			90.49%
Converted SNN [74]	CNN 2		200	87.46%
STBP (without NeuNorm) [72]	CNN 2	8	150	89.83%
STBP (with NeuNorm) [72]	CNN 2	8	150	90.53%
This work	CNN 2	**5**	150	**91.41%**

CNN 1: 96C3-256C3-P2-384C3-P2-384C3-256C3-1024-1024

CNN 2: 128C3-256C3-P2-512C3-P2-1024C3-512C3-1024-512

5.4.7 Firing Sparsity

As presented, the proposed TSSL-BP method can train SNNs with low latency. In the meanwhile, the firing activities in well-trained networks also tend to be sparse. To demonstrate firing sparsity, we select two well-trained SNNs, one for the CIFAR10 and the other for the N-MNIST.

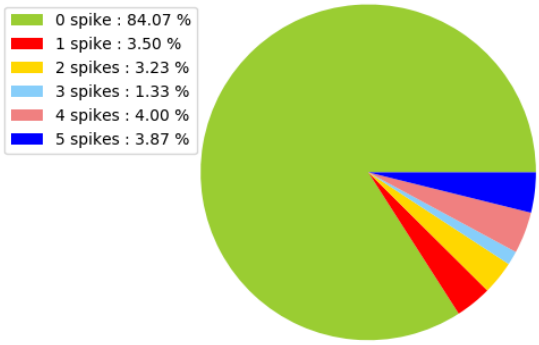

Figure 5.7: Firing activity on CIFAR10.

The CIFAR10 network is simulated over 5 time steps. We count the percentage of neurons that fire $0, 1, \ldots, 5$ times, respectively, and average the percentages over 100 testing samples. As shown in Figure 5.7, the network's firing activity is sparse. More than 84% of neurons are silent while 12% of neurons fire more than once, and about 4% of neurons fire at every time step.

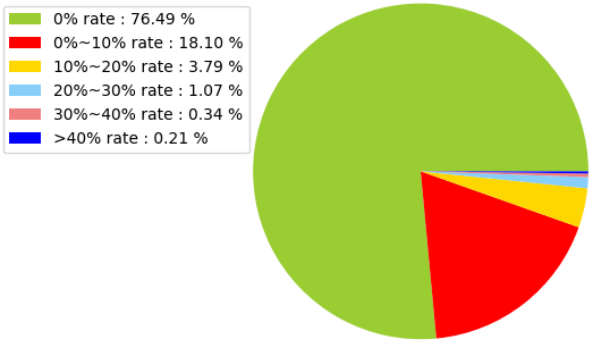

Figure 5.8: Firing activity on N-MNIST.

The N-MNIST network demonstrated here is simulated over 100 time steps. The firing rate of each neuron is logged. The number of neurons with a certain range of firing rates is counted and averaged over 100 testing samples. Similarly, as shown in Figure 5.8, the firing events of the N-MNIST network are also sparse and more than 75% of neurons keep silent. In the meanwhile, there are about 5% of neurons with a firing rate of greater than 10%.

5.5 Summary and Discussions

In this chapter, we present the novel temporal spike sequence learning via a back-propagation (TSSL-BP) method to train deep SNNs. Unlike all prior SNNs BP methods, TSSL-BP improves temporal learning precision by circumventing the non-differentiability of the spiking activation function while faithfully reflecting the all-or-none firing characteristics and the underlying structure in the dependencies of spiking neural activities in both space and time.

TSSL-BP provides a universal BP tool for learning arbitrarily specified target firing sequences with high accuracy while achieving low temporal latency. This is in contrast with most of the existing SNN BP methods which require hundreds of time steps for achieving decent accuracy. The ability in training and inference over a few time steps results in significant reductions of the computational cost required for training large/deep SNNs, and the decision time and energy dissipation of the SNN model when deployed on either a general-purpose or a dedicated neurormorphic computing platform.

Chapter 6

Optimal Structured Recurrent Spiking Neural Networks

In Chapter 4 and Chapter 5, we propose two backpropagation methods for SNNs which can precisely handle time dependency and train the network with high efficiency. However, a high-performance and complex SNN cannot be achieved by only applying powerful learning. The network architecture and synaptic connectivity also play important roles in network dynamics and performance, especially for the recurrent SNNs (RSNNs). Recurrence is ubiquitous in the brain and involved in most of the brain's dynamics. Recurrent connections between neurons play diverse functional roles for storing spatial patterns in memory [75, 76], winner-take-all decision making [77, 78], oscillations of multiple types [39], object recognition in the visual system [79, 80], and so on. Inspired by the connectivity in the brain, recurrent connections have been widely applied in artificial neural networks (ANNs). However, unlike recurrence in traditional ANNs that has been well studied with various structures proposed, the exploration of RSNNs is still immature due to the complex spatial-temporal dynamics. Most RSNNs suffer from two problems. 1. Due to the lack of architectural guidance, random recurrent connectivity

is often adopted, which does not guarantee good performance. 2. Training of RSNNs is in general challenging, bottlenecking achievable model accuracy. As introduced in section 1.3, the recurrent connections in most existing works of RSNNs are sparsely and randomly generated. However, the randomly generated connections may not be optimal and thus limit the performance. In addition, the complex network dynamics created by the random recurrent connections also hinder the network training from learning tools and severely limit the practical application of RSNNs.

In this chapter, we propose two structured approaches for designing high-performance RSNNs and mitigating the training challenges resulted from random recurrent connections as in the prior works.

The first RSNN architecture is called Skip-Connected Self-Recurrent SNN (ScSr-SNN). Recurrence in ScSr-SNN is introduced in a stereotyped manner by adding self-recurrent connections to spiking neurons. The SNNs with self-recurrent connections can realize recurrent behaviors similar to those of more complex RSNNs while the error gradients can be more straightforwardly calculated due to the mostly feedforward nature of the network. In addition, the network dynamics is further enriched by skip connections between nonadjacent layers.

In the recurrent layer of ScSr-SNN, each recurrent neuron only has a self-recurrent connection to the neuron itself. Although such a structure is easy to implement, its connections are so simple that it cannot exploit the full power of recurrence. Therefore, we further propose a novel recurrent structure called the Laterally-Inhibited Self-Recurrent Unit (LISR), which consists of one excitatory neuron with a self-recurrent connection wired together with an inhibitory neuron through excitatory and inhibitory synapses. The self-recurrent connection of the excitatory neuron mitigates the information loss caused by the firing-and-resetting mechanism and maintains the long-term neuronal memory. The lateral inhibition from the inhibitory neuron to the correspond-

ing excitatory neuron, on the one hand, adjusts the firing activity of the latter. On the other hand, it plays as a forget gate to clear the memory of the excitatory neuron. The connectivity of LISR and ScSr-SNN not only offers a structured approach for designing high-performance RSNNs but also mitigates the training challenges resulting from random recurrent connections as in the prior works.

For better illustration, we rewrite the discretized LIF neuron model in (2.4). The neuronal membrane voltage $u_p[t]$ of postsynaptic neuron p at time t is given by

$$u_p[t] = \theta_m u_p[t-1](1 - s_p[t-1]) + \sum_q w_{pq} a_q[t], \tag{6.1}$$

where w_{pq} is the synaptic weight from presynaptic neuron q to postsynaptic neuron p, and $a_q[t]$ the PSC induced by the spikes from neuron q. $\theta_m = 1 - \frac{1}{\tau_m}$ and the $1 - s_p[t-1]$ term reflects the effect of firing-and-resetting mechanism. The spiking neuron generates an output spike when $u_p[t]$ reaches the predetermined threshold V_{th} and reset the $u_p[t]$ to the rest potential.

6.1 ScSr-SNN Architecture

In this section, we first propose the self-recurrent architecture for SNNs. The recurrence is only introduced by self-recurrent connections of individual spiking neurons, i.e., there exist no lateral connections between different neurons within a layer. We demonstrate that, with self-recurrent connections, SNNs are able to realize recurrent behaviors similar to those of more complex RSNNs while the error gradients can be more straightforwardly calculated due to the mostly feedforward nature of the network. Then, we show that the skip connections can help the formation of recurrent structures,introduce more tunability, and thus improve performance. In this section, we apply the same expression

as introduced in 5.1 to describe the activity of the forward pass between two adjacent feedforward layers. For clear demonstration, we rewrite the equation here:

$$\begin{aligned} \boldsymbol{a}^{(l-1)}[t] &= (1 - \frac{1}{\tau_s})\boldsymbol{a}^{(l-1)}[t-1] + \boldsymbol{s}^{(l-1)}[t], \\ \boldsymbol{u}^{(l)}[t] &= \theta_m \boldsymbol{u}^{(l)}[t-1] + \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)}[t], \\ \boldsymbol{s}^{(l)}[t] &= H\left(\boldsymbol{u}^{(l)}[t] - V_{th}\right), \end{aligned} \tag{6.2}$$

where $\boldsymbol{W}^{(l)}$ is the presynaptic weights, $\boldsymbol{a}^{(l-1)}[t]$ presynaptic potentials (PSCs) from neurons in layer $l-1$ at time t, $\boldsymbol{s}^{(l-1)}[t]$ output spike trains of the $l-1$ layer at time t, $\boldsymbol{u}^{(l)}[t]$ membrane potentials of the l layer neurons at time t, and N_l the number of neurons at layer l.

6.1.1 Self-recurrent Architecture

The idea of connecting neurons back to themself in ANNs was introduced in [81]. It presented that by introducing constraints on the recurrent weight matrix, RNNs can learn longer-term patterns with standard stochastic gradient descent. More specifically, it claimed that a kind of longer-term memory can be formed by making part of the recurrent weight matrix close to the identity matrix. It is the same as adding self-recurrent connections to part of hidden neurons. After that, [82] proposed an independently recurrent neural network (IndRNN) with self-recurrent connections where neurons in one layer are independent of each other. It is shown that multiple IndRNNs can be stacked to construct a deep network especially combined with residual connections over layers, and the deep ANNs can be trained robustly. In this chapter, we apply a similar idea to SNNs. We show that applying self-recurrence to SNNs can realize recurrent behaviors similar to those of more complex RSNNs. In addition, the self-recurrent connection can also improve local memory.

For simplicity, we adopt the method in [19] to replace the threshold and synaptic model with a gate function $g()$. Thus, the PSC is defined as

$$\boldsymbol{a}^{(l)}[t] = g\left(\boldsymbol{u}^{(l)}[t]\right), \tag{6.3}$$

$g()$ reveals the relation between the membrane potential and the postsynaptic PSC. It is introduced here to simplify the analysis. In the experiments, we still use (6.2) to simulate the network activities.

In a single layer, the computation of each neuron is independent while neurons are correlated through multiple layers. Neurons in the same self-recurrent layer only recurrently connected to themselves and can be described as

$$\begin{aligned}\boldsymbol{u}^{(l)}[t] = \theta_m \boldsymbol{u}^{(l)}[t-1] \\ + \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)}[t] + \boldsymbol{W}_s^{(l)} \circ g\left(\boldsymbol{u}^{(l)}[t-1]\right),\end{aligned} \tag{6.4}$$

where $\boldsymbol{W}_s^{(l)} = \left[w_{s,1}^{(l)}, \cdots, w_{s,N_l}^{(l)}\right]$ is a vector weight matrix of self-recurrent connections in layer l , $w_{s,i}^{(l)}$ the weight of neuron i's self-recurrent connection in layer l, and $\circ$ the Hadamard product.

Two self-recurrent layers can work similarly to the fully connected recurrent layer. To illustrate this, we approximately derive the behavior of two self-recurrent layers and compare it with a fully connected recurrent layer.

Figure 6.1 presents the layer l and layer $l+1$ of a network while layer l has self-

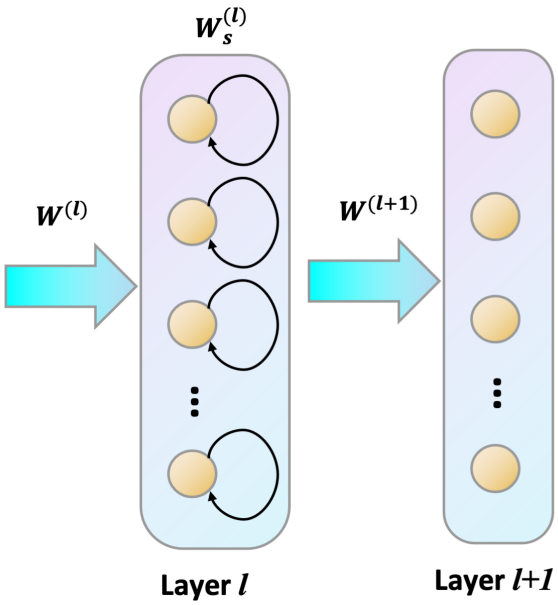

Figure 6.1: Self-recurrent pass of SNNs.

recurrent connections. In this case, the behavior can be expressed as

$$\begin{aligned} \boldsymbol{u}^{(l)}[t] &= \theta_m \boldsymbol{u}^{(l)}[t-1] \\ &\quad + \boldsymbol{W}^{(l)} \boldsymbol{a}^{(l-1)}[t] + \boldsymbol{W}_s^{(l)} \circ g\left(\boldsymbol{u}^{(l)}[t-1]\right), \\ \boldsymbol{u}^{(l+1)}[t] &= \theta_m \boldsymbol{u}^{(l+1)}[t-1] + \boldsymbol{W}^{(l+1)} g\left(\boldsymbol{u}^{(l)}[t]\right). \end{aligned} \tag{6.5}$$

More aggressively, we approximate the $g(v)$ as a linear function $g(v) = hv$. h is a constant value to represent the proportion from membrane potential to the postsynaptic PSC. Then, the Eq.(6.5) can be combined as

$$\begin{aligned} \boldsymbol{u}^{(l+1)}[t] &= \theta_m \boldsymbol{u}^{(l+1)}[t-1] + \boldsymbol{W}^{(l)} \boldsymbol{a}^{(l-1)}[t] \\ &\quad + \boldsymbol{W}_1^{(l)} \boldsymbol{a}^{(l+1)}[t-1] + \boldsymbol{W}_2^{(l)} \boldsymbol{a}^{(l+1)}[t-2], \\ \boldsymbol{W}_1^{(l)} &= \boldsymbol{W}^{(l+1)} \left(\frac{\theta_m}{h} + \boldsymbol{W}_s^{(l)}\right) \left(\boldsymbol{W}^{(l+1)}\right)^{-1} \\ \boldsymbol{W}_2^{(l)} &= -\theta_m \boldsymbol{W}_1^{(l)}. \end{aligned} \tag{6.6}$$

The behavior of two layers network illustrated in Eq.(6.6) is similar to the single fully connected recurrent layer which is described as

$$\boldsymbol{u}^{(l)}[t] = \theta_m \boldsymbol{u}^{(l)}[t-1] + \boldsymbol{W}^{(l)} \boldsymbol{a}^{(l-1)}[t] + \boldsymbol{W}_r^{(l)} \boldsymbol{a}^{(l)}[t-1], \tag{6.7}$$

where $\boldsymbol{W}_r^{(l)}$ is recurrent connection weights matrix.

Compared to the Eq.(6.7), two layers self-recurrent structure can be viewed as a single recurrent layer with two groups of recurrent connections which have the delay of 1 and 2 time steps. Eq.(6.6) also indicates the constraints of two layers self-recurrent structure that: 1. the recurrent weights $\boldsymbol{W}_1^{(l)}$ and $\boldsymbol{W}_2^{(l)}$ should be diagonalizable; 2. $\boldsymbol{W}_2^{(l)}$ is the negative of $\boldsymbol{W}_1^{(l)}$ with the scalar θ_m.

Moreover, by including the reset mechanism and self-recurrent connections, Eq.(6.4) can be written as

$$\boldsymbol{u}^{(l)}[t] = \theta_m \boldsymbol{u}^{(l)}[t-1](1 - \boldsymbol{s}^{(l)}[t-1]) + \boldsymbol{W}^{(l)} \boldsymbol{a}^{(l-1)}[t] + \boldsymbol{W}_s^{(l)} \circ \boldsymbol{a}^{(l)}[t-1]. \tag{6.8}$$

As shown in the first right term, a neuron will lose previous information after it fires and resets the membrane potential. With the self-recurrent connections, the output signals are passed back to the original neurons. From the neuronal perspective, positive feedback can be viewed as compensation for the information loss caused by the firing-and-resetting mechanism. On the other hand, negative feedback can control the neuron's firing activity and regulate the network dynamics.

To sum it up, a two layers self-recurrent structure behaves similarly to a fully connected recurrent layer. However, compared to the existing random connected RSNNs, ScSr-SNN brings several benefits from the self-recurrent connections:

- ScSr-SNN have a simpler but more structured architecture than the randomly gen-

erated RSNNs.

- ScSr-SNN simplify the forward and backward computation in the recurrent structure. The error gradients can be calculated straightforwardly in ScSr-SNN due to the mostly feedforward nature of the network. Thus, the straightforward calculation also cost less computational resources.

- Since the self connections are local within the layer, they can be not only applied to fully connected SNNs but also applicable to other structures like spiking CNNs.

6.1.2 Skip Connections

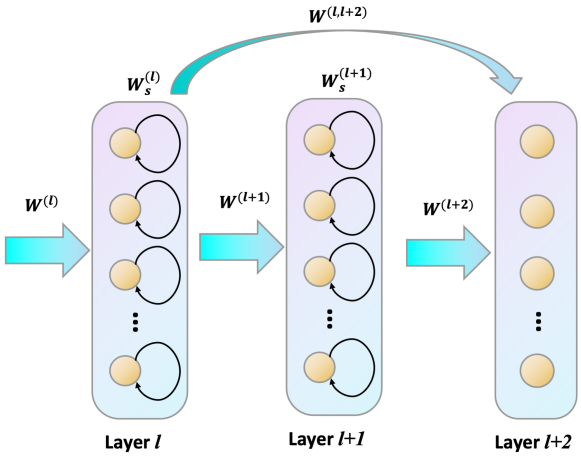

Figure 6.2: Skip-connected pass of SNNs.

As shown in Figure 6.2, we adopt the skip connections which directly connect two non-adjacent layers to enrich the network dynamics. The skip connections benefit the network for three reasons. First, two layers with self-recurrent connections can work similarly to a fully connected recurrent layer. Therefore, in Figure 6.2, we can view layer l and $l+1$ as a recurrent layer while the $l+1$ and $l+2$ layers also form an equivalent

recurrent layer. With the additional skip connections from layer l to $l+2$, these two layers can also be treated as an approximate recurrent layer. The skip connections bring more possibilities and dynamics to form different structures.

Second, the skip connections pass high-layer information to a certain layer and introduce more features. Third, during training, the skip connections form an alternative path for the calculation of backpropagated error gradient and provide additional tunability for the network. Owing to these three features, in Section 6.6.2, we demonstrate that the performance improvement and faster convergence speed are achieved by applying the skip connections.

6.2 LISR architecture

In this section, we present the structure for the proposed Laterally-Inhibited Self-Recurrent Unit (LISR). It has clear and structured recurrent connections which can benefit the network in several aspects.

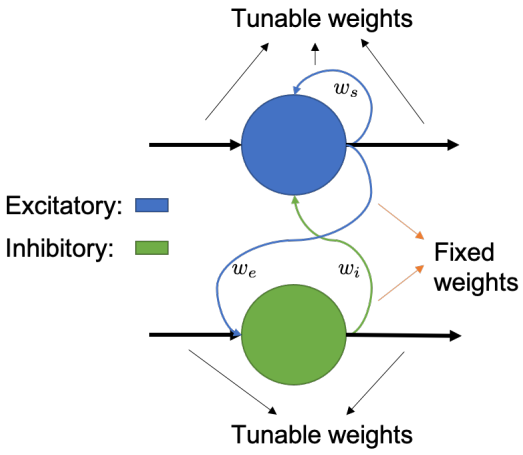

Figure 6.3: Laterally-Inhibited Self-Recurrent unit.

In the recurrent layer, neurons are grouped into pairs. Each LISR consists of one excitatory neuron (E) with a self-recurrent connection and its corresponding inhibitory

neuron (I). As shown in Fig. 6.3, there are three recurrent connections in a LISR, including a self-recurrent connection of E, an excitatory recurrent connection from E to I, and an inhibitory recurrent connection from I to E. Among these recurrent connections, the self-recurrent connection is trained by the learning rule while the weights of the other two connections are fixed. In addition, the neurons of two adjacent layers are fully connected. In this chapter, we suppose all the recurrent connections have delay of 1 time step and feedforward connections have no delay.

In the rest of this chapter, we use e and i in the subscript to denote the variables of the excitatory neuron and inhibitory neuron, respectively. For the inhibitory neuron, by introducing the excitatory recurrent connection, the expression of the LIF neuron model changes from (6.1) to

$$u_i[t] = \theta_m u_i[t-1](1 - s_i[t-1]) + I_i[t] + w_e a_e[t-1] \tag{6.9}$$

where w_e is the fixed weight of the excitatory connection and $a_e[t-1]$ the PSC of the excitatory neuron.

Similarly, the membrane potential of the excitatory neuron can be concluded as

$$\begin{aligned} u_e[t] = \theta_m u_e[t-1](1 - s_e[t-1]) + \sum_q w_{eq} a_q[t] \\ + w_s a_e[t-1] + w_i a_i[t-1] \end{aligned} \tag{6.10}$$

where w_s is the weight of self-recurrent connection, w_i the fixed weight of the inhibitory connection, and $a_i[t-1]$ the PSC of the inhibitory neuron.

In the proposed LISR, the excitatory neuron with self-recurrent connection plays a major role in information processing and feature extraction while receiving lateral inhibition from the inhibitory neuron. Thus, the following analysis is focused on the

excitatory neuron.

As we introduced in Section 6.1, [81], [82], and our proposed ScSr-SNN have applied the self-recurrent connection and demonstrated it as an effective structure for recurrent networks. In this approach, the self-recurrent connection is only applied to the excitatory neuron. As expressed in (6.10), the neuron resets its membrane potential to 0 after firing. All the previous information accumulated in this neuron is lost. Despite the fact that the firing-and-resetting mechanism keeps the dynamics and oscillation of the network [39], there are still situations in which maintaining long-term memories of neurons is beneficial. Thus, self-recurrent connection plays the role of refreshing the memory. In this chapter, the self-recurrent connection is initialized to be a non-negative value then is trained by the learning algorithm. After training, the weights of self-recurrent connections are learned to determine how much previous information should be kept. Although the self-recurrent connections have other benefits as demonstrated in [82], in the proposed LISR, they are mainly used to mitigate the information loss caused by the firing-and-resetting mechanism. Thus, the temporal contextual information is refreshed and held in the internal states of the recurrent structure.

On the other hand, the excitatory neuron also accepts inhibition from the inhibitory neuron. Its membrane potential is depressed by a fixed amount when a spike comes from the inhibitory connection. In other words, while previous information is kept through the self-recurrent connection, the inhibitory connection determines when the existing information should be abandoned. Therefore, the inhibition serves as a gating mechanism to control the flow of information through neurons. In the meanwhile, the inhibitory neuron accepts inputs from presynaptic layer and the corresponding excitatory neuron. The weights connected to presynaptic layer are trained to learn when the inhibition should be generated. From the network perspective, the inhibitory connections also play roles such as filtering input signals, regulating network activities, and maintaining network

dynamics [39, 83].

Moreover, the proposed LISR only contains recurrent connections inside itself. Thus, unlike most existing RSNN works that the networks only have one recurrent layer, this structured unit can be readily exploited as a basic building block for constructing multi-layered networks. Fig. 6.4 demonstrates a deep SNN implementing the proposed LISR. The network has $l+1$ hidden layers with full connections between the adjacent layers. Each layer is constructed by repeating LISRs. The LISRs of the same layer are independent without recurrent connections between each other. In addition, each LISR has three recurrent connections as introduced in this section.

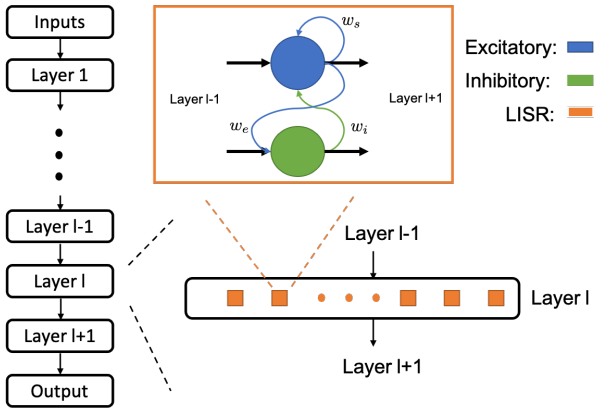

Figure 6.4: Deep SNN based on LISR.

6.3 Backpropagation for ScSr-SNN

We adopt the TSSL-BP method proposed in Chapter 5 to train both the ScSr-SNN and the LISR. TSSL-BP captures the error backpropagation across two types of inter-neuron and intra-neuron dependencies and leads to state-of-the-art performance with extremely low latency. In the Chapter 5, we only derive the TSSL-BP for feedforward SNNs. In this chapter, we extend its applicability to the proposed structures.

Same as the definition of TSSL-BP in Section 5.2.2, the loss function adopt in this chapter is

$$L = \sum_{k=0}^{N_t} E[t_k] = \sum_{k=0}^{N_t} \frac{1}{2}((\epsilon * \boldsymbol{d})[t_k] - (\epsilon * \boldsymbol{s})[t_k])^2, \tag{6.11}$$

where $\epsilon(\cdot)$ is a kernel function. $\boldsymbol{d} = [\boldsymbol{d}[t_0], \cdots, \boldsymbol{d}[t_{N_t}]]$ and $\boldsymbol{s} = [\boldsymbol{s}[t_0], \cdots, \boldsymbol{s}[t_{N_t}]]$ are the desired and the actual spike trains in the output layer, respectively.

In this section, we consider the backpropagation in three cases: 1. feedforward layer, 2. self-recurrent layer, 3. self-recurrent layer with skip-connections to a post layer. In the rest of this chapter, variables associated with neurons in the layer l have (l) as the superscript.

6.3.1 Feedforward Layer

For the feedforward layer, the backpropagation flow is derived in Section 5.2.2. For completeness, we conclude the major part of the derivation. From (6.1), the error gradient with respect to the presynaptic weight w_{pq} from neuron q in the layer $l-1$ to neuron p in the layer l can be defined as

$$\begin{aligned} \frac{\partial L}{\partial w_{pq}} &= \sum_{k=0}^{N_t} \frac{\partial E[t_k]}{\partial w_{pq}} = \sum_{k=0}^{N_t} \sum_{m=0}^{k} \frac{\partial E[t_k]}{\partial u_p^{(l)}[t_m]} \frac{\partial u_p^{(l)}[t_m]}{\partial w_{pq}} \\ &= \sum_{m=0}^{N_t} a_q^{(l)}[t_m] \sum_{k=m}^{N_t} \frac{\partial E[t_k]}{\partial u_p^{(l)}[t_m]} = \sum_{m=0}^{N_t} a_q^{(l)}[t_m] \delta_p^{(l)}[t_m], \end{aligned} \tag{6.12}$$

where $\delta_p^{(l)}[t_m]$ denotes the error for neuron p of layer l at time t_m and is defined as:

$$\delta_p^{(l)}[t_m] = \sum_{k=m}^{N_t} \frac{\partial E[t_k]}{\partial u_p^{(l)}[t_m]} = \sum_{k=m}^{N_t} \frac{\partial E[t_k]}{\partial a_p^{(l)}[t_k]} \frac{\partial a_p^{(l)}[t_k]}{\partial u_p^{(l)}[t_m]}. \tag{6.13}$$

In this approach, the neurons in the output layer are regular feedforward neurons without recurrent connection. Therefore, the weights of output neuron o are updated by

$$\frac{\partial L}{\partial w_{oq}} = \sum_{m=0}^{N_t} a_q[t_m] \sum_{k=m}^{N_t} \frac{\partial E[t_k]}{\partial a_o[t_k]} \frac{\partial a_o[t_k]}{\partial u_o[t_m]}, \tag{6.14}$$

where $\frac{\partial E[t_k]}{\partial a_o[t_k]}$ depends on the choice of the loss function.

For hidden layer without recurrent connection, the backpropagated error $\delta_p^{(l)}[t_m]$ of neuron p in layer l can be derived from the error δ of the post layer:

$$\begin{aligned} \delta_p^{(l)}[t_m] &= \sum_{k=m}^{N_t} \sum_{j=k}^{N_t} \frac{\partial a_p^{(l)}[t_k]}{\partial u_p^{(l)}[t_m]} \sum_{h=1}^{N^{(l+1)}} \left(\frac{\partial u_h^{(l+1)}[t_k]}{\partial a_p^{(l)}[t_k]} \frac{\partial E[t_j]}{\partial u_h^{(l+1)}[t_k]} \right) \\ &= \sum_{k=m}^{N_t} \frac{\partial a_p^{(l)}[t_k]}{\partial u_p^{(l)}[t_m]} \sum_{h=1}^{N^{(l+1)}} w_{hp} \delta_h^{(l+1)}[t_k]. \end{aligned} \tag{6.15}$$

where $N^{(l+1)}$ denotes the number of neurons in the layer $l+1$.

The calculation of the key term $\frac{\partial a[t_k]}{\partial u[t_m]}$ is included in Chapter 5. We do not repeat the steps but treat it as a known term in this chapter.

6.3.2 Self-recurrent Layer

The structure of the self-recurrent layer is shown in Figure 6.1. Similar to the feedforward case, the weights of the incoming synapses can be calculated according to Eq.(6.12).

Based on Eq.(6.4), the error gradient with respect to the self-recurrent weight $w_{s,p}$ of neuron p in the lth layer can be expressed as:

$$\frac{\partial L}{\partial w_{s,p}} = \sum_{m=1}^{N_t} a_p^{(l)}[t_m - 1] \delta_p^{(l)}[t_m]. \tag{6.16}$$

Compared to the feedforward case, the main difference of the self-recurrent case comes from the derivation of $\delta_p[t_m]$. Except the error signals from the post layer, we also need to take the errors backpropagated from self-recurrent connections into consideration. Thus,

the error $\delta_p[t_m]$ is calculated by:

$$\begin{aligned}\delta_p^{(l)}[t_m] &= \sum_{k=m}^{N_t}\sum_{j=k}^{N_t}\frac{\partial a_p^{(l)}[t_k]}{\partial u_p^{(l)}[t_m]}\sum_{h=1}^{N^{(l+1)}}\left(\frac{\partial u_h^{(l+1)}[t_k]}{\partial a_p^{(l)}[t_k]}\frac{\partial E[t_j]}{\partial u_h^{(l+1)}[t_k]}\right)\\ &+\sum_{k=m}^{N_t}\sum_{j=k+1}^{N_t}\frac{\partial a_p^{(l)}[t_k]}{\partial u_p^{(l)}[t_m]}\left(\frac{\partial u_p^{(l)}[t_k+1]}{\partial a_p^{(l)}[t_k]}\frac{\partial E[t_j]}{\partial u_p^{(l)}[t_k+1]}\right)\\ &=\sum_{k=m}^{N_t}\frac{\partial a_p^{(l)}[t_k]}{\partial u_p^{(l)}[t_m]}\sum_{h=1}^{N^{(l+1)}} w_{hp}\delta_h^{(l+1)}[t_k]+w_{s,p}^{(l)}\sum_{k=m}^{N_t-1}\frac{\partial a_p^{(l)}[t_k]}{\partial u_p^{(l)}[t_m]}\delta_p^{(l)}[t_k+1].\end{aligned} \tag{6.17}$$

In Eq.(6.17), the first term is the same as Eq.(6.15) which represent the error back-propagated from the post layer. The second term is caused by self-recurrent connections. It reveals that membrane potentials $u_p^{(l)}$ of the neuron p in layer l influence its (unweighted) PSCs $a_p^{(l)}$ through fired spikes, and $a_p^{(l)}$ further affect the membrane potentials of $u_p^{(l)}$ at the next time step.

6.3.3 Self-recurrent Layer with skip connections

As shown in Figure 6.2, we suppose layer l has self-recurrent connections. In addition, it also connects to layer $l+1$ the same as the feedforward layer and layer $l+2$ through the skip connections. The changes of BP method still come from the derivation of $\delta_p^{(l)}[t_m]$. In this case, the output (unweighted) PSCs $a_p^{(l)}$ of layer l further directly affects the membrane potentials of neurons at layer $l+2$.

Similar to the derivation of Eq.(6.17), an additional term should be added when the skip connections are taken into account. Therefore, the $\delta_p^{(l)}[t_m]$ can be derived as

$$\begin{aligned}\delta_p^{(l)}[t_m] &= \sum_{k=m}^{N_t}\frac{\partial a_p^{(l)}[t_k]}{\partial u_p^{(l)}[t_m]}\sum_{h=1}^{N^{(l+1)}} w_{hp}\delta_h^{(l+1)}[t_k]+w_{s,p}^{(l)}\sum_{k=m}^{N_t-1}\frac{\partial a_p^{(l)}[t_k]}{\partial u_p^{(l)}[t_m]}\delta_p^{(l)}[t_k+1]\\ &+\sum_{k=m}^{N_t}\frac{\partial a_p^{(l)}[t_k]}{\partial u_p^{(l)}[t_m]}\sum_{g=1}^{N^{(l+2)}} w_{gp}\delta_h^{(l+2)}[t_k],\end{aligned} \tag{6.18}$$

where W_{gp} represents weight of skip connection from neuron p of layer l to neuron g of layer $l+2$.

Figure 6.5 summarizes the forward pass and backward pass of the proposed ScSr-SNN at the single-neuron level. In the figure, neurons in layer l have skip connections to neurons in layer $l+q$. Neurons in layer $l-p$ and $l+1$ are also connected through skip connections. As shown, at the spatial level, the neuron communicates with neurons of different layers via regular feedforward connections and skip connections. However, within the same layer, the neuron only depends on itself at the temporal level via intrinsic parameter and self-recurrent connection. This simple structure results in efficient learning with rich neural dynamics.

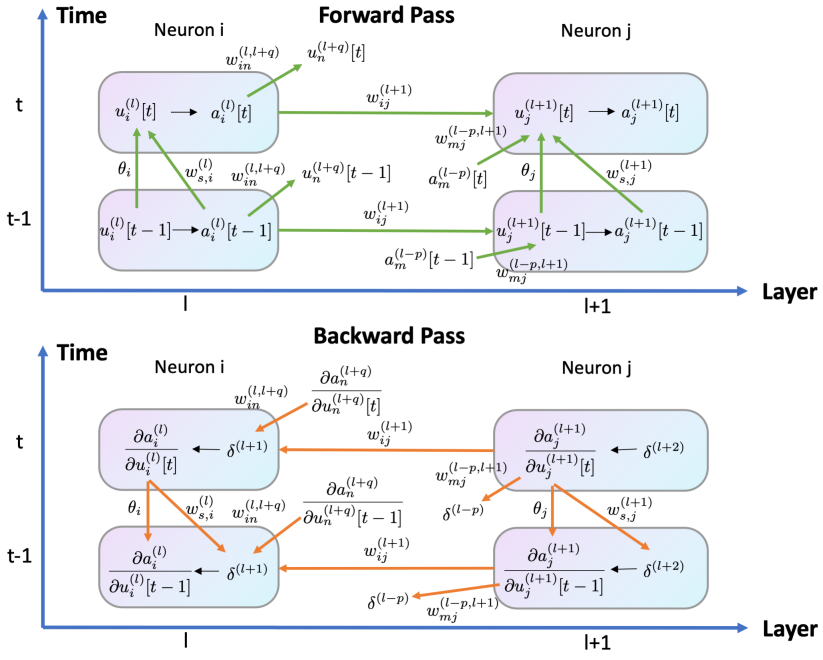

Figure 6.5: Forward and backward pass of ScSr-SNN.

6.4 Backpropagation for LISR

In LISR, the backpropagation for feedforward layers including the output layer is the same as the one for ScSr-SNN in Section 6.3. For the neurons in the hidden layer, we derive the learning rule for the excitatory neuron e and the inhibitory neuron i in the layer l separately.

The weights update of neuron e and neuron i still follows the (6.12). However, due to their special recurrent connections, the derivations of $\delta_e^{(l)}$ and $\delta_i^{(l)}$, the error backpropagated from postsynaptic layer, are different.

For the excitatory neuron, in addition to the error signals from the postsynaptic layer, the error backpropagated from the self-recurrent connection and the excitatory recurrent connection should also be taken into consideration. Thus, the backpropagated error can be calculated by:

$$\begin{aligned}\delta_e^{(l)}[t_m] = &\sum_{k=m}^{N_t}\sum_{j=k}^{N_t}\frac{\partial a_e^{(l)}[t_k]}{\partial u_e^{(l)}[t_m]}\sum_{p=1}^{N^{(l+1)}}\left(\frac{\partial u_p^{(l+1)}[t_k]}{\partial a_e^{(l)}[t_k]}\frac{\partial E[t_j]}{\partial u_p^{(l+1)}[t_k]}\right)\\ &+\sum_{k=m}^{N_t}\sum_{j=k+1}^{N_t}\frac{\partial a_e^{(l)}[t_k]}{\partial u_e^{(l)}[t_m]}\left(\frac{\partial u_e^{(l)}[t_k+1]}{\partial a_e^{(l)}[t_k]}\frac{\partial E[t_j]}{\partial u_e^{(l)}[t_k+1]}\right)\\ &+\sum_{k=m}^{N_t}\sum_{j=k+1}^{N_t}\frac{\partial a_e^{(l)}[t_k]}{\partial u_e^{(l)}[t_m]}\left(\frac{\partial u_i^{(l)}[t_k+1]}{\partial a_e^{(l)}[t_k]}\frac{\partial E[t_j]}{\partial u_i^{(l)}[t_k+1]}\right).\end{aligned} \tag{6.19}$$

where $N^{(l+1)}$ denotes the number of neurons in the layer $l+1$.

(6.19) can be written as

$$\begin{aligned}\delta_e^{(l)}[t_m] = &\sum_{k=m}^{N_t}\frac{\partial a_e^{(l)}[t_k]}{\partial u_e^{(l)}[t_m]}\sum_{p=1}^{N^{(l+1)}}w_{pe}\delta_p^{(l+1)}[t_k]\\ &+\sum_{k=m}^{N_t-1}\frac{\partial a_e^{(l)}[t_k]}{\partial u_e^{(l)}[t_m]}w_s\delta_e^{(l)}[t_k+1]+\sum_{k=m}^{N_t-1}\frac{\partial a_e^{(l)}[t_k]}{\partial u_e^{(l)}[t_m]}w_e\delta_i^{(l)}[t_k+1],\end{aligned} \tag{6.20}$$

where $\delta_p^{(l+1)}[t_k]$ is the error of the neuron p in the layer $l+1$ at time t_k, $\delta_e^{(l)}[t_k+1]$ and $\delta_i^{(l)}[t_k+1]$ the errors of the excitatory neuron itself and the inhibitory neuron at time t_k+1 respectively.

(6.20) reveals that membrane potential $u_e^{(l)}$ of the excitatory neuron in layer l influences its (unweighted) PSC $a_e^{(l)}$ through spikes, and $a_e^{(l)}$ further affects the membrane potentials of its postsynaptic neurons. Its first term reflects that the error of the excitatory neuron is accumulated from the errors of all postsynaptic layer neurons. The second and third terms indicate that the errors are also backpropagated from the excitatory neuron itself and the inhibitory neuron.

Similarly, the error of inhibitory neuron can be obtained by

$$\delta_i^{(l)}[t_m] = \sum_{k=m}^{N_t} \frac{\partial a_i^{(l)}[t_k]}{\partial u_i^{(l)}[t_m]} \sum_{p=1}^{N^{(l+1)}} w_{pi}\delta_p^{(l+1)}[t_k] + \sum_{k=m}^{N_t-1} \frac{\partial a_i^{(l)}[t_k]}{\partial u_i^{(l)}[t_m]} w_i \delta_e^{(l)}[t_k+1], \tag{6.21}$$

where the first term represents errors from postsynaptic layer and second term is the error from the excitatory neuron.

6.5 Experiments and Results

In this chapter, we test the proposed ScSr-SNN and LISR on the speech dataset TI46-Alpha [56] and the neuromorphic speech dataset N-TIDIGITS [61]. In addition, the proposed LISR structure is also evaluated on the neuromorphic video dataset DVS-Gesture [63]. The LISR networks in the experiments are formed such that all the hidden layers are composed of LISRs. In other words, a hidden layer with n neurons has $\frac{n}{2}$ LISRs with $\frac{n}{2}$ excitatory neurons and $\frac{n}{2}$ inhibitory neurons.

6.5.1 Parameter Settings

The simulation step size is set to 1 ms. The parameters like threshold and learning rate are empirically tuned. Table 6.1 lists the typical values adopted for each dataset. AdamW [84] is adopted as the optimizer. The simulation step size is set to 1 ms. The hyperparameters and weights initialization are empirically tuned.

Parameter	**TI46-Alpha**	**N-TIDIGITS**	**DvsGesture**
τ_m	16 ms	64 ms	64 ms
τ_s	8ms	8 ms	8 ms
learning rate	0.0005	0.0002	0.0001
Threshold V_{th}	1 mV	1 mV	1 mV
Batch Size	50	50	5
Time steps	100	300	300
Epochs	400	400	100

Table 6.1: Parameters settings.

In ScSr-SNN, the weight matrices of self-recurrent connections and skip connections are initialized following the normal distribution with a mean of 0 and standard deviation of 1. No axon and synaptic delay or refractory period are adopted in the feedforward pass whereas the self-recurrent connections have 1 ms delay. No dropout or normalization is applied. Moreover, we also separately apply the proposed self-recurrent structure and skip-connected structure to reveal their individual effects of boosting performance. For all the results reported in this chapter, we use Sr-SNNs to denote the networks with only self-recurrent connections in their recurrent layers. In addition, ScSr-SNN represent the full proposed structure. In the experiments, all the ScSr-SNN have 3 hidden layers and the skip connections are from the first hidden layer to the third hidden layer.

For LISR, the fully connected weights between layers are initialized by the He Normal initialization proposed in [85]. The self recurrent weights are initialized to 0.5 while the weights of the excitatory recurrent connection and the inhibitory recurrent connection

are fixed to 1 and -2, respectively.

Practical issues such as desired output selection, warm-up mechanism, and the boundary of derivatives, follow the same solutions in Section 5.4.2.

6.5.2 Loss Function

For the BP method used in this chapter, the loss function can be defined by any errors that measure the distance between the actual outputs and the desired outputs. In our experiments, since hundreds of time steps are required for simulating speech and video inputs, we choose the accumulated output PSCs to define the error which is similar to the firing count used in many existing works [16, 17].

We suppose the simulation time steps for a sample is N_t. Furthermore, for neuron o of the output layer, we define the desired output as d_o and real output as r_o where $r_o = \sum_{k=1}^{N_t} a_o[t_k]$ and d_o is manually determined. Therefore, the loss is determined by the square error of the outputs

$$L = \sum_{k=1}^{N_t} E[t_k] = \sum_{o}^{N^{(out)}} \frac{1}{2}(d_o - r_o)^2, \tag{6.22}$$

where $N^{(out)}$ is the number of neurons in the output layer.

Furthermore, the error at each time step is simply defined by the averaged loss through all the time steps:

$$E[t_k] = \frac{L}{N_t}, \quad E_o[t_k] = \frac{(d_o - r_o)^2}{2N_t}. \tag{6.23}$$

With the loss function defined above, the error δ can be calculated through the (6.15, 6.17, 6.18, 6.20, 6.21).

6.5.3 TI46-Alpha

In Table 6.2, we compare the proposed structures with several existing results on the full TI46-Alpha dataset. [69] applies the state vector of the reservoir to train the single readout layer using BP. Its result shows that only training the single readout layer of a recurrent LSM is inadequate for this challenging task, demonstrating the necessity of a more structured and deep SNN. The ST-RSBP proposed in Chapter 4 demonstrates the best-reported performance of TI46-Alpha. It has one recurrent layer and two feedforward layers trained with the ST-RSBP method. However, both recurrent networks in [69] and ST-RSBP are randomly generated. We show that, with the proposed ScSr-SNN method, we can achieve a performance of 95.13% which is 1.78% better than the best-reported result we obtained with ST-RSBP method. In the table, we also demonstrate that the self-recurrent connections and skip connections can both improve performance when they are applied separately. In addition, with the same training rule TSSL-BP, our proposed methods can boost 1.99% performance on the same network size compared to the feedforward networks trained by our proposed TSSL-BP method.

The network for the proposed LISR structure only has one hidden layer with 800 neurons. In other words, the hidden layer contains 400 LISRs with 400 excitatory neurons and 400 inhibitory neurons. In the last two rows of Table 6.2, we compare the performance between the proposed structure and feedforward SNN. These two experiments have the same network size, learning rule, preprocessing steps, and hyperparameters. The only difference is the network structure in the hidden layer. As shown, by implementing the proposed structure, the network can achieve 5.03% performance improvement with almost the same number of tunable parameters. Among all the results, the proposed LISR outperforms the result of ST-RSBP by 2.73%. Furthermore, it achieves the best performance with only a similar or smaller number of parameters which leads to high-

efficient training and inference.

Table 6.2: Accuracy on TI46-Alpha

Network Structure	Learning Rule	Hidden Layers	# Params	Accuracy
LSM [69]	Non-spiking BP	2000	$52,000$	78%
Feedforward SNNs [16]	HM2BP	800	$83,200$	89.92%
RSNNs	ST-RSBP	$400-400-400$	$363,313$	93.35%
Feedforward SNNs	TSSL-BP	$400-400-400$	$361,600$	93.14%
Sr-SNNs	TSSL-BP	$400-400-400$	$362,800$	94.62%
ScSr-SNN	TSSL-BP	$400-400-400$	$522,800$	**95.13**%
Sr-SNNs	TSSL-BP	800	$84,000$	93.06%
Feedforward SNNs	TSSL-BP	800	$83,200$	91.05%
LISR	TSSL-BP	800	$83,600$	**96.08**%

6.5.4 N-TIDIGITS

For the N-TIDIGITS dataset, performance is compared with the feedforward networks trained by HM2BP [16], RSNNs trained by ST-RSBP, and the Skip-Connected Self-Recurrent SNN (ScSr-SNN) trained by TSSL-BP. In addition, we not only compare the performance with previous work on RSNNs but also with the well-known RNNs in non-spiking networks such as Gated Recurrent Unit (GRU) and Long-Short Term Memory (LSTM). The GRU and LSTM networks are trained by the non-spiking BP method. As shown in Table 6.3, both ScSr-SNN and LISR methods outperform the state-of-the-art results in the existing works.

We demonstrate the performance of the proposed LISR structure by two networks. The first network only has one hidden layer with 400 neurons while another network has three hidden layers with 400 neurons in each layer. All the hidden layers are implemented with the proposed LISR. Similarly, the feedforward SNNs with the same network size are also tested. By comparing with the feedforward counterparts, the proposed structure is not only effective for the single-layer network but also highly improves performance

with multiple hidden layers. For the single-layer network, the proposed method can impressively improve performance by 9.26% compared to the feedforward SNN with a similar number of parameters.

Furthermore, the LISR structure also achieves more than 5% better performance than the widely-adopted recurrent structures of ANNs, the GRU and LSTM, on this dataset. One possible reason for the LISR network significantly outperforming ANNs is that the dataset is neuromorphic based with spikes as inputs. Thus, the SNNs are more likely to handle the spatial-temporal information inside the dataset effectively.

Table 6.3: Accuracy on N-TIDIGITS

Network Structure	Learning Rule	Hidden Layers	# Params	Accuracy
Feedforward SNN [16]	HM2BP	$250-250$	$81,250$	89.69%
GRU [61]	Non-spiking BP	$200-200-100$	$314,700$	90.90%
Phase LSTM [61]	Non-spiking BP	$250-250$	$818,750$	91.25%
Feedforward SNN	TSSL-BP	400	$30,000$	84.84%
LISR	TSSL-BP	400	$30,200$	**94.10%**
RSNN	ST-RSBP	$400-400-400$	$351,241$	93.90%
Sr-SNNs	TSSL-BP	$400-400-400$	$351,200$	94.02%
ScSr-SNN	TSSL-BP	$400-400-400$	$511,200$	**95.07**%
Feedforward SNN	TSSL-BP	$400-400-400$	$350,000$	91.03%
LISR	TSSL-BP	$400-400-400$	$350,600$	**96.65**%

6.5.5 DVS-Gesture

In the experiments on the DVS-Gesture dataset, all the networks have the same size. The inputs are first processed by the pooling layer of 4×4 pooling kernel size. Thus, the inputs to the 512 neurons hidden layer have 2 channels with the size of 32×32 in each channel.

As demonstrated in Table 6.4, the proposed LISR structure can improve performance by up to 2.43% compared to the same size feedforward SNNs trained by the proposed

TSSL-BP or STBP [15]. In addition, performance of the proposed method also outperforms the ANN structures including vanilla RNN and LSTM. In [86], a rate-coding-inspired loss function is proposed for enhancing performance of ANNs on neuromorphic image datasets. However, our proposed method still achieves more than 1.74% performance improvement over the rate-coding-inspired loss function, and our method uses a much smaller number of parameters.

Table 6.4: Accuracy on DVS-Gesture

Network Structure	Learning Rule	Hidden Layers	# Params	Accuracy
Feedforward SNN [86]	STBP	$P4-512$	$1,054,208$	87.50%
RNN [86]	Non-spiking BP	$P4-512$	$1,316,352$	52.78%
LSTM [86]	Non-spiking BP	$P4-512$	$5,250,560$	88.19%
Feedforward SNN	TSSL-BP	$P4-512$	$1,054,208$	88.19%
LISR	TSSL-BP	$P4-512$	$1,054,464$	**89.93**%

6.6 Analysis

6.6.1 Effects of Self-recurrent Connections

In the proposed method, the self-recurrent connections are constructed so that the network can realize recurrent behaviors similar to those of more complex RSNNs while the error gradients can be calculated more straightforwardly. The weights of self-recurrent connections are randomly initialized following the normal distribution with the mean of 0 and standard deviation of 1. By the BP method, the weights of the connections can be trained to minimize the loss function.

We take the well-trained ScSr-SNN on TI46-Alpha as an example. The network contains 3 layers with 400 neurons in each layer. We record the weights of self-recurrent connections after training. Figure 6.6 shows the distribution of the 1200 self-recurrent

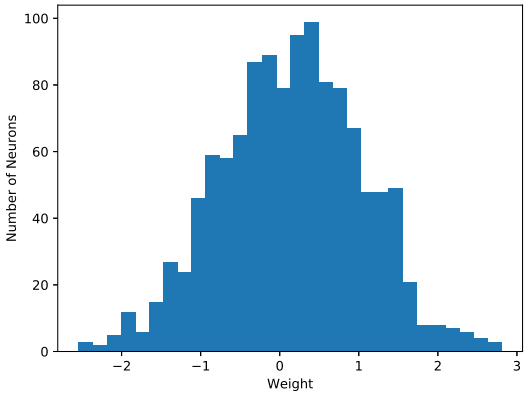

Figure 6.6: Weights distribution of well-trained self-recurrent connections

weights. As shown, there are about 60% positive self-recurrent connections. From the network perspective, the well-trained self-recurrent weights guarantee the complex dynamics of the RSNNs and minimize the output loss. From the single neuron level, on one hand, the positive self-recurrent connections refresh the information of the neuron and thus maintain the single-neuron memory. On the other hand, the negative self-recurrent connection depresses the neuron and can be considered as a regulation for the neuron's activity.

6.6.2 Effects of Skip Connections

The skip connections mainly play three roles. First, the skip connections combined with self-recurrent connections introduce additional recurrent structures and thus further enhance neural dynamics. Second, the skip connections pass high-level information to a certain layer and introduce more features. Finally, the skip connections provide an alternative path for the gradient.

When the network goes deeper, the effects of skip connections become even stronger. We compare two structures, ScSr-SNN and Sr-SNNs, on the TI-Alpha dataset. The

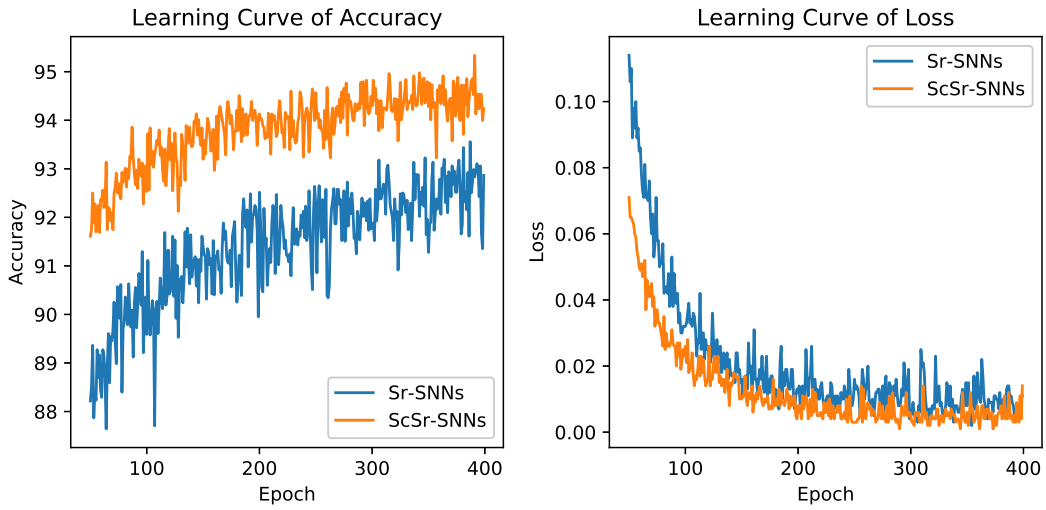

Figure 6.7: Effects of skip connections

networks have six hidden layers with 100 neurons in each layer. The only difference between these two networks is that ScSr-SNN has skip connections from the first layer to the fifth layer. As shown in Figure 6.7, with the skip connections, the performance can be improved for more than 1.5%. Apart from the performance improvement, the ScSr-SNN network can also achieve the same loss with up to 50 fewer epochs than Sr-SNNs. The network dynamics and more features introduced by skip connections lead to the performance improvement. In the meanwhile, the faster convergence is benefited from the additional path to pass the backpropagated errors.

There are many possible ways to connect non-adjacent layers with skip connections in a deep network. It's necessary to know if more skip connections lead to better performance. We conduct a few experiments on the TI46-Alpha dataset. The network has 4 hidden layers with 200 neurons in each layer. As shown in Table 6.5, we compare two kinds of connections. The one skip connections network only has skip connections from the first hidden layer to the third hidden layer. The two skip connections means the first

hidden layer to the third hidden layer and the second hidden layer to the fourth hidden layer are both connected. The results show that the proposed methods with only skip connections from the first layer to the third layer achieve the best result. This may be because one skip connections can enrich the network dynamics while multiple skip connections lead to the instability of the network and also cause complexity for the proposed BP method to well train the network.

Structures	Best
Feedforward SNNs	92.08%
ScSr-SNN: one skip connections	94.27%
ScSr-SNN: two skip connections	93.56%

Table 6.5: Performances on TI46-Alpha with more layers

6.6.3 Computational Efficiency

Owing to the simple structure of self-recurrent connections, the proposed ScSr-SNN has lower computational complexity compared to fully connected RSNNs. For a layer with n neurons, the proposed self-recurrent connections only introduce $2n$ more tunable parameters. However, for a fully connected recurrent layer, it has n^2 parameters in the recurrent weight matrix. During simulation, the time cost mainly comes from the error backpropagated through recurrent connections, because the gradient must be calculated time step by time step. Thus, the proposed structure can be more efficient than fully connected RSNNs.

More specifically, we use the networks in the experiments of TI46-Alpha as an example. Each network has 3 hidden layers with 400 neurons in each layer. In addition, the inputs have 78 channels and the output layer has 26 neurons. Therefore, the number of tunable parameters of the feedforward network is $361,600$. For the proposed method, Sr-SNNs have $362,800$ parameters. Thus, by comparing the number of parameters, we can

conclude that there's almost no computational overhead by applying self-recurrent connections. After implementing the skip connections, ScSr-SNN have $522,800$ parameters. The largely increased number of parameters may result in additional computational costs. Thus, a trade-off between the cost and performance should be taken into consideration when applying skip connections.

Moreover, the proposed LISR also contributes to the high-efficient training and inference of the network. For a hidden layer with n neurons, since the weights of excitatory and inhibitory connections are fixed and only self-recurrent connections are trained, the implementation of LISR only introduces $2.5n$ more parameters for the forward pass and $0.5n$ parameters for the backward pass. The number of additional recurrent weights is small compared to the parameters between layers. Since the computational cost of simulation is highly related to the number of parameters, the LISR network can enhance the network performance with almost no additional cost compared to feedforward SNN. In addition, the TSSL-BP, which can train networks over a short temporal window of a few time steps, is adopted as the learning rule for the proposed methods to further reduce the computational overhead.

6.6.4 Firing Activity

When considering the implementation of the proposed structures on neuromorphic hardware, low power consumption becomes an important constraint. In most cases, the power consumption is highly related to the firing rate of the whole network. A good network should not only demonstrate decent performance but also keeps relatively low firing activities.

To demonstrate the firing sparsity of the proposed ScSr-SNN, we select the well-trained ScSr-SNN on TI46-Alpha. We randomly apply one sample to the well-trained

network from the test set. The firing rate of each hidden neuron is recorded. As shown in Figure 6.8, about 55% neurons are silent while 26% neurons have firing rates less than 10%. In addition, only less than 3% neurons have firing rates that are higher than 40%. Therefore, from the firing activity point, the proposed method also demonstrates decent computational efficiency.

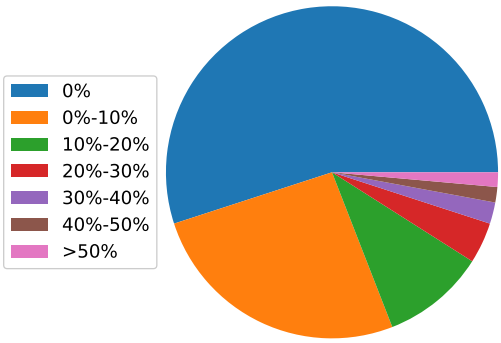

Figure 6.8: Firing acitivity of well-trained ScSr-SNN

To analyze firing events of LISR, we select two well-trained networks, one feedforward SNN and one LISR network. Both of the networks have only one hidden layer with 800 neurons. An alphabet sample with 'A' speech of the TI46-Alpha dataset is used for the experiment.

Fig. 6.9 presents the firing events of all neurons in the hidden layer of the two networks. Each blue dot in the figure represents a spike. For the LISR network, the first 400 neurons are excitatory and the other 400 neurons are inhibitory. As shown in Fig. 6.9, the activity of the LISR network is denser than the activity of the feedforward network. Even for the excitatory neurons which receive strong inhibition from the inhibitory neuron, more firing events are observed compared to the neurons in the feedforward network.

Moreover, we calculate the firing rate of each neuron in the hidden layers of the

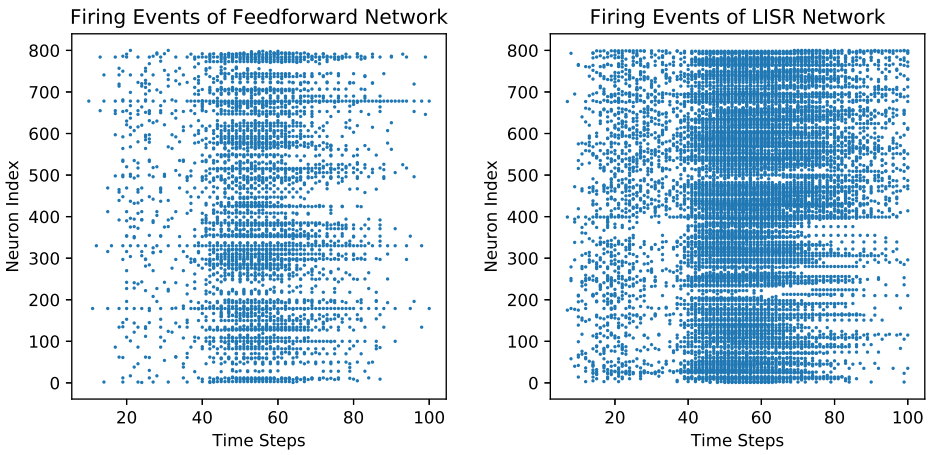

Figure 6.9: Firing events on a TI46-Alpha sample.

feedforward network and the LISR network. In Fig. 6.10, the firing rates of the networks are categorized into seven groups. The number of neurons that have firing rates within certain thresholds is summed together. As shown in Fig. 6.10, about half of the neurons in the feedforward network are silent while only 14.375% neurons have firing rates greater than 10%. However, for the same input sample, only 23% neurons of the LISR network are silent. In the meanwhile, more than 48% neurons have firing rates greater than 20%, and 12.125% neurons have more than 30% firing rates.

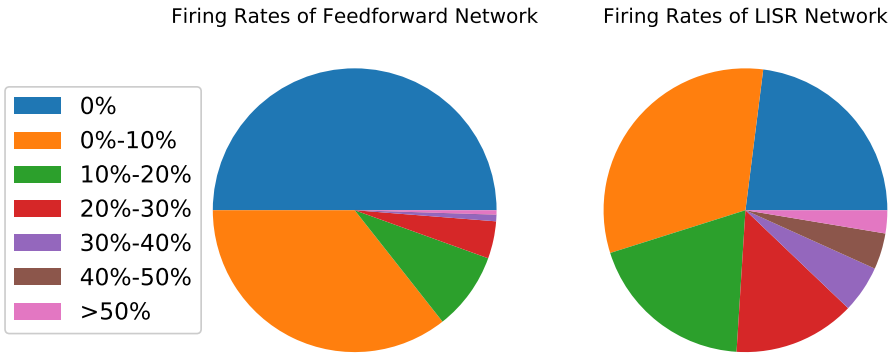

Figure 6.10: Firing rates on a TI46-Alpha sample.

As illustrated above, the LISR network enhances the network activity so that more neurons are involved in learning and information processing. Moreover, since the spatial-temporal information is passed through spikes, the neuron requires a certain level of activity to transfer and filter information. Furthermore, only when the activity is high enough can the neuron be sensitive to the small inputs. Therefore, the proposed structure benefits the network performance by refreshing and gating the information maintained inside the neurons as well as adjusting the network activities. On the other hand, although the overall firing rates of the LISR network are slightly higher, its firing activity is still sparse with more than half of neurons having a firing rate of 10% or lower. Thus, the sparsity of firing events, which is one of the crucial features of biologically inspired networks, is maintained.

6.7 Summary and Discussions

In this chapter, we propose two architectures of RSNNs with structured recurrent connectivity. Both of them can be efficiently trained by our proposed TSSL-BP and achieve state-of-the-art performances.

For the Skip-Connected Self-Recurrent SNNs (ScSr-SNN), its benefits can be summarized as:

- Self-recurrent Connections: (1) The network can realize recurrent behaviors similar to those of more complex RSNNs. (2) A neuron loses all its information after generating a spike and resetting membrane potential to 0. With the positive self-recurrent connections, such information will be obtained by the neuron again. This process can maintain the single-neuron memory. (3) The structure is easy to interpret due to the independence of neurons in each layer. (4) It simplifies the forward and backward computation in the recurrent structure.

- Skip connections: (1) The skip connections combined with self-recurrent connections introduce additional recurrent structure as demonstrated in this chapter. Thus, it further enhances network dynamics. (2) The skip connections pass high-layer information to a certain layer and introduce more features. (3) The skip connections provide an alternative path for the gradient.

For Laterally-Inhibited Self-Recurrent Unit (LISR), we demonstrate the proposed structure and illustrate that the LISR can be easily applied to deep RSNNs. In addition, the learning rule is derived to be eligible to train this unique structure. In the results, the proposed method is evaluated on three datasets cover speeches, neuromorphic speeches, and neuromorphic images. The experimental results consistently show that the proposed structure can outperform the existing methods as well as the feedforward counterparts with a similar or smaller number of parameters. The impressive performance improvement comes from the refreshing and gating mechanism, and the regulation of network activities introduced by the LISR network.

In the future, we would like to evaluate the effectiveness of the proposed structure on hardware.We believe the proposed structures will benefit the brain-inspired computing community from both a structural and algorithmic perspective.

Chapter 7

Hybrid Risk-Mitigating Architectural Search

In neural circuits, recurrent connectivity plays a crucial role in network function and stability. However, existing recurrent spiking neural networks (RSNNs) are often constructed by random connections without optimization. RSNNs can produce rich dynamics that are critical for memory formation and learning. Although in Chapter 6 we proposed two effective RSNN architectures ScSr-SNN and LISR, systemic architectural optimization of RSNNs is still an opening challenge.

Neural architectural search (NAS), the process of automating non-spiking ANNs construction, becomes prevalent recently after achieving state-of-the-art performance on various tasks [87, 88]. Different types of strategies such as reinforcement learning [89], gradient-based optimization [90], and evolutionary algorithms [91] have been proposed to find optimal architectures of traditional CNNs and RNNs. In contrast, architectural optimization of SNNs has received little attention. Only recently, [92] adopted a simulated annealing algorithm to learn the optimal hyperparameters of liquid state machine (LSM) models through a three-step search. The number of liquids in each layer, num-

ber of neurons in each liquid, and internal parameters like connectivity percentage and excitatory vs. inhibitory neuron ratio are optimized during the process. Similarly, a surrogate-assisted evolutionary search method was applied in [93] to optimize the hyperparameters of LSM such as density of connections, input connection probability, distribution of synaptic strength, and time constant of the neuron model. However, both works focused only on LSM for which hyperparameters indirectly affecting recurrent connections as opposed to specific connectivity patterns were optimized. The recurrence in the network was still randomly determined without any optimization after hyperparameters were chosen.

In this chapter, we aim to enable the systemic design of large RSNNs via a new scalable RSNN architecture and automated architectural optimization. RSNNs can create complex network dynamics both in time and space, which manifests itself as an opportunity for achieving great learning capabilities and a challenge in practical realization. It is important to strike a balance between theoretical computational power and architectural complexity. Firstly, we argue that composing RSNNs based on well-optimized building blocks small in size, or recurrent motifs, can lead to an architectural solution scalable to large networks while achieving high performance. We assemble multiple recurrent motifs into a layer architecture called Sparsely-Connected Recurrent Motif Layer (SC-ML). The motifs in each SC-ML share the same *topology*, defined by the size of the motif, i.e., the number of neurons, and the recurrent connectivity pattern between the neurons. The motif topology is determined by the proposed architectural optimization while the weights within each motif may be tuned by a standard training algorithm, e.g., backpropagation. Motifs in a recurrent SC-ML layer are wired together using sparse lateral connections determined by imposing spatial connectivity constraints. As such, there exist two levels of structured recurrence: recurrence within each motif and recurrence between the motifs at the SC-ML level. The fact that the motifs are small in size and that inter-motif con-

nectivity is sparse alleviates the difficulty in architectural optimization and training of these motifs and SC-ML. Furthermore, multiple SC-ML layers can be stacked and wired using additional feedforward weights to construct even larger recurrent networks.

Secondly, we demonstrate a method called Hybrid Risk-Mitigating Architectural Search (HRMAS) to optimize the proposed recurrent motifs and SC-ML layer architecture. HRMAS is an alternating two-step optimization process hybridizing bio-inspired intrinsic plasticity for mitigating the risk in architectural optimization. Facilitated by gradient-based methods [90] and our proposed TSSL-BP, the first step of optimization is formulated to optimize network architecture defined by size of the motif, intra and inter-motif connectivity patterns, types of these connections, and the corresponding synaptic weight values, respectively. While structural changes induced by the architectural-level optimization are essential for finding high-performance RSNNs, they may be misguided due to discontinuity in architectural search, and limited training data, hence over-fitting. We mitigate the risk of network instability and performance degradation caused by architectural change by introducing a novel biologically-inspired "self-repairing" mechanism through intrinsic plasticity, which has the same spirit of homeostasis during neural development [94]. The intrinsic plasticity is introduced to the second step of each HRMAS iteration and acts as unsupervised fast self-adaption to mitigate the risks imposed by structural and synaptic weight modifications introduced by the first step during the RSNN architectural "evolution".

7.1 Sparsely-Connected Recurrent Motif Layer (SC-ML)

Unlike the traditional non-spiking RNNs that are typically constructed with units like LSTM or GRU, the structure of existing RSNNs is random without specific optimization. The complex recurrent connectivity hinders RSNN performance and prevents scaling to large RSNNs. The proposed SC-ML is composed of multiple sparsely-connected recurrent *motifs*, where each motif consists of a group of recurrently connected spiking neurons, as shown in Figure 7.1. The motifs in each SC-ML share the same topology, which is defined as the size of the motif, i.e., the number of neurons, and the recurrent connectivity pattern between the neurons. Within the motif, synaptic connections can be constructed between any two neurons or to introduce self-recurrent connections back to the same neurons. Thus the problem of the recurrent layer optimization is greatly reduced to that of learning the optimal motif and sparse inter-motif connectivity, alleviating the difficulty in architectural optimization and allowing scalability to large networks.

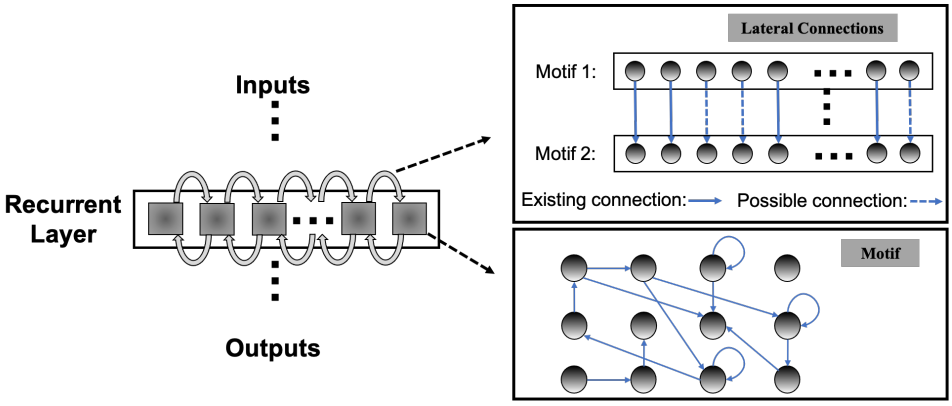

Figure 7.1: Sparsely-Connected Recurrent Motif Layer.

This motif-based structure is motivated from both a biological and a computational perspective. First, from a biological point of view, there is evidence that the neocortex is

not only organized in layered minicolumn structures but also into synaptically connected clusters of neurons within such structures [95, 96]. For example, networks of pyramidal cells cluster into multiple groups of a few dozen neurons each. Second, from a computational perspective, optimizing the connectivity of the basic building block, i.e., the motif, is a more simplified problem than optimizing the connectivity of the whole recurrent layer for which the number of possible inter-neuron connections increases exponentially with neuron count. Third, by constraining most recurrent connections inside the motifs and allowing a few lateral connections between neighboring motifs to exchange information across the SC-ML, the total number of recurrent connections is limited. This leads to a great deal of sparsity as observed in biological networks [97].

Figure 7.1 presents an example of SC-ML with 12-neuron motifs. The lateral inter-motif connections can be introduced as the mutual connections between two corresponding neurons in neighboring motifs or constructed in other ways to ensure sparsity and reduce complexity. With the proposed SC-ML, a multi-layer large RSNN can be easily composed by stacking multiple SC-MLs with feedforward weights between layers. Within a multi-layered network, information processing is facilitated through local processing of different motifs, communication of motif-level responses via inter-motif connections, and extraction and processing of higher-level features layer by layer.

7.2 Hybrid Risk-Mitigating Architectural Search

To systematically optimize the motif topology and lateral connections of SC-ML and thus fully elevate the performance of RSNNs, we propose an optimization framework called Hybrid Risk-Mitigating Architectural Search (HRMAS), where each optimization iteration consists of two alternating steps.

7.2.1 Hybrid Risk-Mitigating Architectural Search Framework

In HRMAS, all recurrent connections are categorized into three types: inhibitory, excitatory, and non-existence. An inhibitory connection has a negative weight and is fixed without training in our current implementation. The weight of an excitatory connection is positive and trained by a backpropagation (BP) method. HRMAS is an alternating two-step optimization process hybridizing architectural optimization with intrinsic plasticity (IP). The first step of each HRMAS optimization iteration optimizes the topology of the motif and inter-motif connectivity in SC-ML and the corresponding synaptic weights hierarchically. Specially, the optimal number of neurons in the motif is optimized over a finite set of motif sizes. All possible intra-motif connections are considered and the type of each connection is optimized, which may lead to a sparser connectivity if the connection types of certain synapses are determined to be "non-existence". At the inter-motif level, a sparse motif-to-motif connectivity constraint is imposed, e.g., neurons in one motif are only allowed to be wired up with the corresponding neurons in the neighboring motifs. Inter-motif connections also fall under one of the three types. Hence, a greater level of sparsity is produced with the emergence of connections of type "non-existence". Strategies like this reduce the complexity of the network and its optimization. The second step in each HRMAS iteration executes an unsupervised IP rule to stabilize the network function and mitigate potential risks caused by architectural changes.

Figure 7.2 illustrates the incremental optimization strategy we adopt for the architectural parameters. Using the two-step optimization detailed later, initially all architectural parameters including motif size and connectivity are optimized. After several training iterations, we choose the optimal motif size from a set of discrete options. As this most critical architectural parameter is set, we continue to optimize the remaining architectural parameters defining connectivity, allowing fine-turning of performance based on the

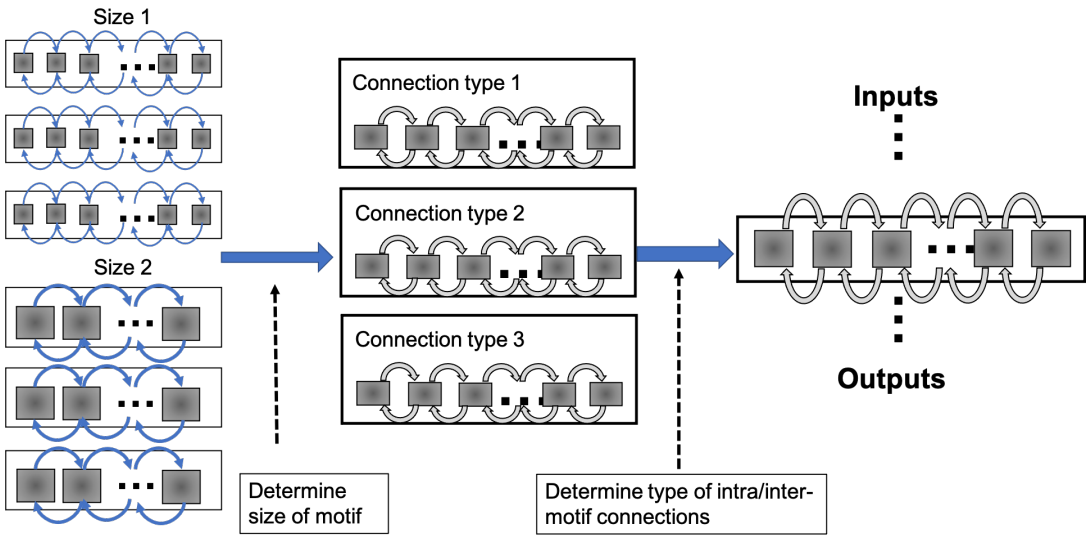

Figure 7.2: Architectural optimization in HRMAS.

chosen motif size.

7.2.2 Comparison with prior neural architectural search work of non-spiking RNNs

Neural architecture search (NAS) has been applied for architectural optimization of traditional non-spiking RNNs, where a substructure called cell is optimized by a search algorithm [89]. Based on the optimized substructure, multiple cells are stacked to form a network. A cell contains nodes organized in a directed acyclic graph which is usually a tree. As in Figure 7.3, the cell takes the input x_t at time t and produces the current state h_t based on the previous state h_{t-1}. The operation, e.g. based on a specific combination or activation function at each node is optimized during architectural search. Nevertheless, this NAS approach may not be the best fit for RSNNs. First, recurrence in the cell as in Figure 7.3 is only created by feeding h_{t-1} back to the cell while connectivity inside the cell is feedforward. While a long history of practice led to widely adopted LSTM and GRU models and the optimal cell found by the above NAS procedure can be complex

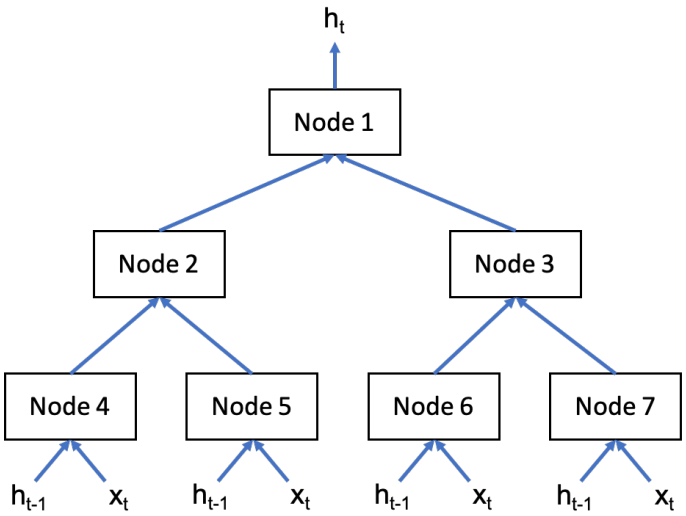

Figure 7.3: Cell in traditional RNNs.

than an LSTM/GRU cell, the overall operations and connectivity of the final RNN do not go beyond an LSTM-like architecture. Finally, the considered combination operations and activation functions like addition and elementwise multiplication are not biologically plausible.

In comparison, in RSNNs based on the proposed SC-ML architecture, we add onto the memory effects resulted from temporal integration of individual spiking neurons by introducing sparse intra or inter-motif connections. This corresponds to a scalable and biologically plausible RSNN architectural design space that closely mimics the micro-circuits in the nervous system. Furthermore, we develop the novel alternating two-step HRMAS framework hybridizing gradient-based optimization and biologically-inspired intrinsic plasticity for robust NAS of RSNNs.

7.2.3 Alternating Two-Step Optimization in HRMAS

The alternating two-step optimization in HRMAS is inspired by the evolution in neural development. As shown in Figure 7.4, neural circuits may experience weight changes

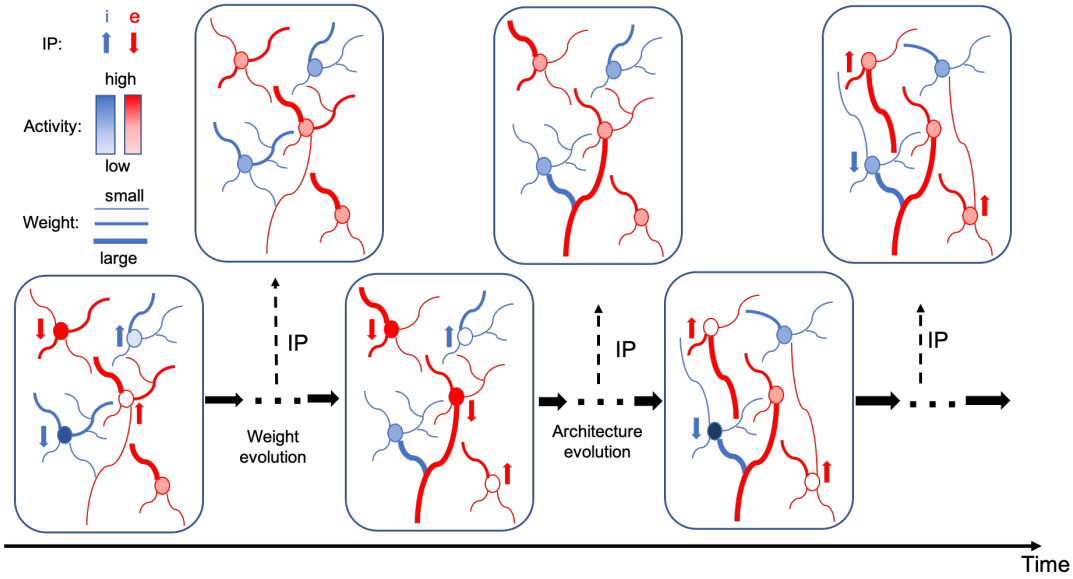

Figure 7.4: Evolution in neural development.

through synaptic plasticity. Over a longer time scale, circuit architecture, i.e., connectivity, may evolve through learning and environmental changes. In addition, spontaneous firing behaviors of individual neurons may be adapted by intrinsic plasticity (IP). We are motivated by the important role of local IP mechanisms in stabilizing neuronal activity and coordinating structural changes to maintain proper circuit functions [94]. We view IP as a "fast-paced" self-adapting mechanism of individual neurons to react to and minimize the risks of weight and architectural modifications. As shown in Figure 7.5, we define the architectural parameters (motif size and intra/inter-motif connection types weights), synaptic weights, and intrinsic neuronal parameters as α, w, and β, respectively. Each HRMAS optimization iteration consists of two alternating steps. In the first step, we optimize α and w hierarchically based on gradient-based optimization using backpropagation (BP). In Figure 7.5, δ is the backpropagated error obtained via the employed BP method. In the second step, we use an unsupervised IP rule to adapt the intrinsic neuronal parameters of each neuron over a time window ("IP window") during which

training examples are presented to the network. IP allows the neurons to respond to the weight and architectural changes introduced in the first step and mitigate possible risks caused by such changes. In Step 1 of the subsequent iteration, the error gradients w.r.t the synaptic weights and architectural parameters are computed based on the most recent values of β updated in the preceding iteration.

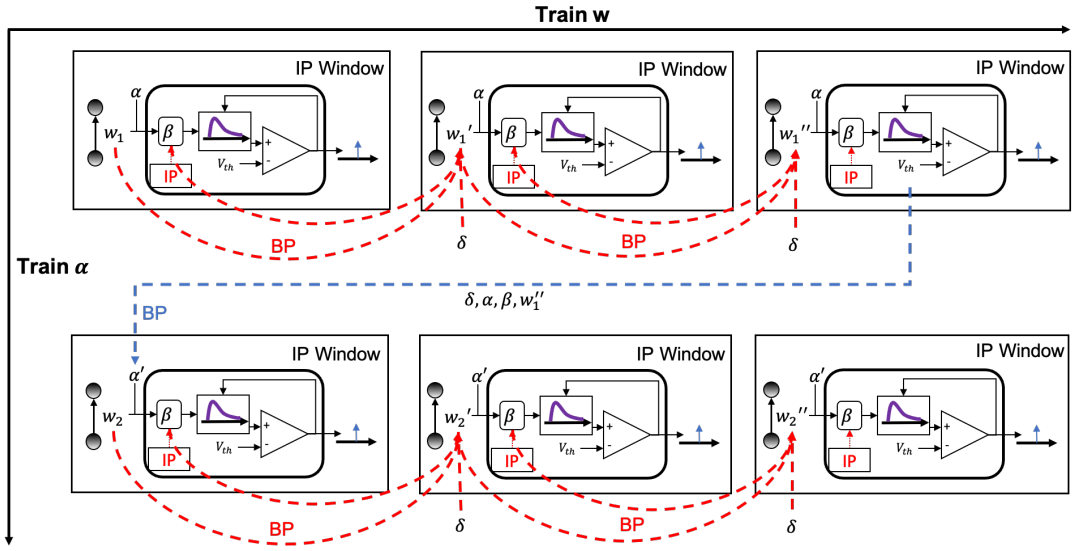

Figure 7.5: Proposed HRMAS.

The first step of the k-th HRMAS iteration solves a bi-level optimization problem using BP:

$$min_{\alpha}\mathcal{L}_{valid}(\alpha, w^*(\alpha), \beta^*_-) \tag{7.1}$$

$$s.t. \quad w^*(\alpha) = arg_w min\mathcal{L}_{train}(\alpha, w, \beta^*_-), \tag{7.2}$$

where $\mathcal{L}_{valid}$ and $\mathcal{L}_{train}$ are the loss functions defined based on the validation and training sets used to train α and w respectively; β^*_- is the intrinsic parameter values updated in the preceding $(k-1)$-th iteration; $w^*(\alpha, \beta)$ denotes the optimal synaptic weights under the architecture specified by α. The second step of the k-th iteration solves the optimization

problem below:

$$\beta^* = arg_\beta min \mathcal{L}_{ip}(\alpha^*, w^*, \beta) \tag{7.3}$$

$\mathcal{L}_{ip}$ is the local loss to be minimized by the IP rule as further discussed in Section 7.2.6.

7.2.4 Gradient-based Optimization in HRMAS

Optimizing the weight and architectural parameters by solving the bi-level optimization problem of (7.1, 7.2) can be computational expensive. We adapt the recent method [90] to reduce computational complexity by relaxing the discrete architectural parameters to continuous ones for efficient gradient-based optimization. It significantly reduces the computational cost of architecture search by approximating the bi-level optimization problem, relaxing the discrete architectural parameters to continuous ones, and solving the continuous model by gradient descent.

Without loss of generality, we consider a multi-layered RSNN consisting of one or more SC-ML layers, where connections between layers are assumed to be feedforward. We focus on one such SC-ML layer, as shown in Figure 7.6, to discuss the proposed gradient-based optimization.

The number of neurons in the SC-ML layer is fixed. The motif size is optimized such that each neuron is partitioned into a specific motif based on the chosen motif size. The largest white square in Figure 7.6 shows the layer-connectivity matrix of all intra-layer connections of the whole layer, where the dimension of the matrix corresponds to the neuron count of the layer. We superimpose three sets of smaller gray squares onto the layer-connectivity matrix, one for each of the three possible motif sizes of v_1, v_2, and v_3 considered. Choosing a particular motif size packs neurons in the layer into multiple motifs, and the corresponding gray squares in Figure 7.6 illustrate the intra-motif connectivity introduced within the SC-ML layer.

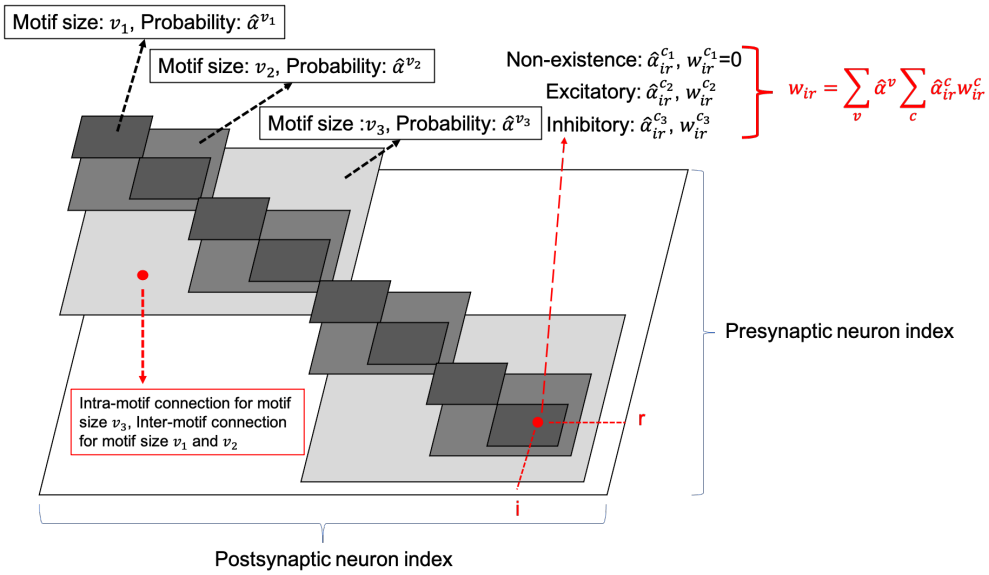

Figure 7.6: SC-ML with relaxed architectural parameters.

The entry of the layer-connectivity matrix at row r and column i specifies the existence and nature of the connection from neuron r to neuron i. We consider multiple motif size and connection type choices during architectural search using continuous-valued paramterizations α^v and α_{ir}^c, respectively for each motif size v and connection type c. we relax the categorical choice of each motif size using a softmax over all possible options: $\hat{\alpha}^v = \frac{exp(\alpha^v)}{\sum_{v' \in \mathcal{V}} exp(\alpha^{v'})}$, and similarly relax the categorical choice of each connection type based on the corresponding motif size: $\hat{\alpha}_{ir}^c = \frac{exp(\alpha_{ir}^c)}{\sum_{c' \in \mathcal{C}} exp(\alpha_{ir}^{c'})}$. Here, $\mathcal{C}$ and $\mathcal{V}$ are the set of all possible connection types and motif sizes, respectively; $\hat{\alpha}^v$ and $\hat{\alpha}_{ir}^c$ are the continuous-valued categorical choice of motif size v and connection type c, respectively, which can also be interpreted as the probability of selecting the corresponding the motif size or connection type. In this chapter, we use hat over the variable to denote the architectural parameter processed by softmax. Then, the task of architecture optimization is reduced to learn a set of continuous variables $\hat{\alpha} = \{\hat{\alpha}^c, \hat{\alpha}^v\}$. With the continuous architectural parameters, a gradient-based method like BP is applicable to learn the recurrent

connectivity.

In [90], the bi-level optimization problem is simply approximated to a one-shot model to reduce the expensive computational cost of the inner optimization which can be expressed as

$$\nabla_{\hat{\alpha}}\mathcal{L}_{valid}(\hat{\alpha}, w^*(\hat{\alpha})) = \nabla_{\hat{\alpha}}\mathcal{L}_{valid}(\hat{\alpha}, w - \eta\nabla_w\mathcal{L}_{train}(w, \hat{\alpha})), \tag{7.4}$$

where η is the learning rate for a step of inner loop. Both the weights of the search network and the architectural parameters are trained by the BP method.

The architectural gradient can be approximated by

$$\frac{d\mathcal{L}_{valid}}{d\hat{\alpha}}(\hat{\alpha}) = \nabla_{\hat{\alpha}}\mathcal{L}_{valid}(\hat{\alpha}, w^*) - \eta\nabla_w\mathcal{L}_{valid}(\hat{\alpha}, w^*)\nabla^2_{\hat{\alpha},w}\mathcal{L}_{train}(w^*, \hat{\alpha})). \tag{7.5}$$

The complexity is reduced by using the finite difference approximation around $w^{\pm} = w \pm \epsilon\nabla_w\mathcal{L}_{valid}(\hat{\alpha}, w^*)$ for small perturbation ϵ to compute the gradient of $\nabla_{\hat{\alpha}}\mathcal{L}_{valid}(\hat{\alpha}, w^*)$. Finally the architectural updates in (7.5) can be calculated as

$$\frac{d\mathcal{L}_{valid}}{d\hat{\alpha}}(\hat{\alpha}) = \nabla_{\hat{\alpha}}\mathcal{L}_{valid}(\hat{\alpha}, w^*) - \frac{\eta}{2\epsilon}(\nabla_{\hat{\alpha}}\mathcal{L}_{train}(w^+, \hat{\alpha}) - \nabla_{\hat{\alpha}}\mathcal{L}_{train}(w^-, \hat{\alpha})). \tag{7.6}$$

7.2.5 Backpropagation via HRMAS framework

As shown in Figure 7.6, the synaptic weight of the connection from neuron r to neuron i is expressed as the summation of weights under all possible motif sizes and connection types weighted by the respective continuous-valued categorical choices (selection probabilities).

Based on the leaky integrate-and-fire (LIF) neuron model in (2.4), the neuronal membrane voltage $u_i[t]$ of neuron i in the SC-ML layer at time t is given by integrating cur-

rents from all inter-layer inputs and intra-layer recurrent connections under all possible architectural parameterizations:

$$u_i[t] = (1 - \frac{1}{\tau})u_i[t-1] + \frac{R}{\tau}(\sum_j w_{ij}a_j[t] + \sum_{v\in\mathcal{V}}(\hat{\alpha}^v \sum_r^{I_i^v}\sum_{c\in\mathcal{C}}(\hat{\alpha}_{ir}^c w_{ir}^c a_r[t-1]))), \tag{7.7}$$

where R and τ are the resistance and time constant of the membrane, w_{ij} the synaptic weight from neuron j in the previous layer to neuron i, w_{ir}^c the recurrent weight from neuron r to neuron i of connection type c, and $a_j[t]$ the (unweighted) postsynaptic current (PSC) converted from spikes of presynaptic neuron j through a synaptic model. To reduce clutter in the notation, we use I_i^v to denote the number of presynaptic connections afferent onto neuron i's input in the recurrent layer when choosing motif size v, which includes both inter and intra-motif connections. We further drop the explicit dependence of $\hat{\alpha}_{ir}^c$ on $\hat{\alpha}^v$. Through (7.7), the continuous architecture parameterizations influence the integration of input currents, and hence firing activities of neurons in all layers and affect the loss function defined at the output layer. As such, the task of architecture optimization reduces to the one that learns the set of optimal continuous variables $\hat{\alpha}^c$ and $\hat{\alpha}^v$. The final architecture is constructed by choosing the parameterizations with the highest selection probabilities obtained from the optimization.

During the learning, We define the loss function as

$$L = \sum_{k=0}^{T} E[t_k], \tag{7.8}$$

where T is the total time steps and $E[t_k]$ the loss at t_k. From (7.7), the membrane potential $u_i[t]$ of the neuron i at time t demonstrates contribution to all future fires and losses of the neuron through its PSC $a_i[t]$. Therefore, the error gradient with respect to

the presynaptic weight w_{ij} from neuron j to neuron i can be defined as

$$\frac{\partial L}{\partial w_{ij}} = \sum_{m=0}^{T} \frac{R}{\tau} a_j[t_m] \sum_{k=m}^{T} \frac{\partial E[t_k]}{\partial u_i[t_m]} = \sum_{m=0}^{T} \frac{R}{\tau} a_j[t_m] \delta_i[t_m], \tag{7.9}$$

where $\delta_i[t_m]$ denotes the error for neuron i at time t_m and is defined as:

$$\delta_i[t_m] = \sum_{k=m}^{T} \frac{\partial E[t_k]}{\partial u_i[t_m]} = \sum_{k=m}^{T} \frac{\partial E[t_k]}{\partial a_i[t_k]} \frac{\partial a_i[t_k]}{\partial u_i[t_m]}. \tag{7.10}$$

In this chapter, the output layer is regular feedforward layer without recurrent connection. Therefore, the weight w_{oj} of output neuron o is updated by

$$\frac{\partial L}{\partial w_{oj}} = \sum_{m=0}^{T} \frac{R}{\tau} a_j[t_m] \sum_{k=m}^{T} \frac{\partial E[t_k]}{\partial a_o[t_k]} \frac{\partial a_o[t_k]}{\partial u_o[t_m]}, \tag{7.11}$$

where $\frac{\partial E[t_k]}{\partial a_o[t_k]}$ depends on the choice of the loss function.

Now, we focus on the backpropagation in the recurrent hidden layer while the feedforward hidden layer case can be derived similarly. For a neuron i in SC-ML, in addition to the error signals from the next layer, the error backpropagated from the recurrent connections should also be taken into consideration. The backpropagated error can be calculated by:

$$\begin{aligned}
\delta_i[t_m] &= \sum_{k=m}^{T} \sum_{j=k}^{T} \frac{\partial a_i[t_k]}{\partial u_i[t_m]} \sum_{p=1}^{N_p} \left(\frac{\partial u_p[t_k]}{\partial a_i[t_k]} \frac{\partial E[t_j]}{\partial u_p[t_k]} \right) + \sum_{k=m}^{T} \sum_{j=k+1}^{T} \frac{\partial a_i[t_k]}{\partial u_i[t_m]} \sum_{r}^{N_r} \left(\frac{\partial u_r[t_k+1]}{\partial a_i[t_k]} \frac{\partial E[t_j]}{\partial u_r[t_k+1]} \right) \\
&= \sum_{k=m}^{T} \frac{\partial a_i[t_k]}{\partial u_i[t_m]} \sum_{p=1}^{N} \left(\frac{R}{\tau} w_{pi} \delta_p[t_k] \right) + \sum_{k=m}^{T-1} \frac{\partial a_i^{(l)}[t_k]}{\partial u_i^{(l)}[t_m]} \sum_{v \in \mathcal{V}} \left(\hat{\alpha}^v \sum_{r}^{O_i^v} \sum_{c \in \mathcal{C}} \frac{R}{\tau} \hat{\alpha}_{ri}^c w_{ri}^c \delta_r[t_k+1] \right),
\end{aligned} \tag{7.12}$$

where N_p and N_r are the number of neurons in the next layer and the number of neurons in this recurrent layer, respectively. δ_p and δ_r are the errors of the neuron p in the next

layer and the error from the neuron r through the recurrent connection. O_i^v represents all the postsynaptic neurons of neuron i's outputs in the recurrent layer when choosing motif size v, which includes both inter and intra-motif connections.

The key term in (7.12) is $\frac{\partial a[t_k]}{\partial u[t_m]}$ which reflects the effect of neuron's membrane potential on its output PSC. Due to the non-differentiable spiking events, it becomes the main difficulty for the BP of SNNs. Various approaches are proposed to handle this problem such as probability density function of spike state change [17], surrogate gradient [98], and the TSSL-BP method proposed in Chapter 5.

With the error backpropagated according to (7.12), the weights and architectural parameters can be updated by gradient descent as:

$$\begin{aligned}
&\Delta w_{ij} \propto \delta_i[t]\frac{R}{\tau}a_j[t], \quad \Delta\hat{\alpha}^v \propto \sum_i^{N_r}\delta_i[t]\frac{R}{\tau}\sum_r^{I_i^v}(\sum_{c\in\mathcal{C}}\hat{\alpha}_{ir}^c w_{ir}^c a_r[t-1]),\\
&\Delta w_{ir}^c \propto \delta_i[t]\frac{R}{\tau}\sum_{v\in\mathcal{V}}(\hat{\alpha}^v\hat{\alpha}_{ir}^c a_r[t-1]), \quad \Delta\hat{\alpha}_{ir}^c \propto \delta_i[t]\frac{R}{\tau}\sum_{v\in\mathcal{V}}(\hat{\alpha}^v w_{ir}^c a_r[t-1]).
\end{aligned} \tag{7.13}$$

where $\delta_i[t]$ is the backpropagated error for neuron i at time t given in (7.12), N_r is the number of neurons in this recurrent layer, R and τ are the leaky resistance and membrane time constant, two intrinsic parameters adapted by the IP rule, $a_j[t]$ and $a_r[t]$ are the (unweighted) postsynaptic currents (PSCs) generated based on synpatic model by the presynaptic neuron j in the preceding layer and the r-th neuron in this recurrent layer, respectively.

7.2.6 Risk Minimizing Optimization with Intrinsic Plasticity

For architectural optimization of non-spiking RNNs, gradient-based methods are shown to be unstable in some cases due to misguided architectural changes and conversion from the optimized continuous-valued parameterization to a discrete architectural solu-

tion, hindering the final performance and demolishing the effectiveness of learning [99]. Adaptive regularization which modifies the regularization strength (weight decay) guided by the largest eigenvalue of $\nabla^2_\alpha \mathcal{L}_{valid}$ was proposed to address this problem [99]. While this method shows promise for non-spiking RNNs, it is computationally intensive due to frequent expensive eigenvalue computation, severely limiting its scalability.

To address similarly observed risks due to architectural modifications for RSNNs, we propose an efficient biologically-inspired risk-mitigating technique. In biological circuits, it has been shown that Intrinsic Plasticity (IP), also known as homeostasis plasticity, plays an important role in minimizing these risks. IP is a self-adaptive mechanism of biological neurons that maintains homeostasis and shapes the dynamics of neural circuits [27, 28, 29]. It has been observed in neural development that IP not only maintains the stability of neuronal activity but also can act at a network level to coordinate changes in connectivity and excitability across multiple neurons to stabilize circuit function [100, 94]. Inspired by these biological observations, the HRMAS framework incorporates the IP rule into the proposed architectural optimization process at the second step of each optimization iteration. IP is based on local neural firing activities and performs online adaptation with minimal additional computational overhead.

IP has been applied in spiking neural networks for locally regulating neuron activity [33, 20]. In this chapter, we make use of IP for a very different purpose: mitigating the risk of RSNN architectural modifications. We adopt the SpiKL-IP rule proposed in Chapter 3 for all recurrent neurons during architecture optimization. SpiKL-IP adapts the intrinsic parameters of a spiking neuron while minimizing the KL-divergence from the output firing rate distribution to a targeted exponential distribution. It both maintains a level of network activity and maximizes the information transfer from the input to the output for each neuron.

From the SpiKL-IP in Chapter 3, We adapt leaky resistance and membrane time

constant, two intrinsic neuronal parameters of each neuron using SpiKL-IP which effectively solves the optimization problem in (7.3) in an online manner. The two neuronal parameters R and τ are updated according to the approximate average firing rate of the neuron by:

$$\Delta R = \frac{2y\tau V_{th} - W - V_{th} - \frac{1}{\mu}\tau V_{th} y^2}{RW}, \quad \Delta\tau = \frac{-1 + \frac{y}{\mu}}{\tau}, \quad W = \frac{V_{th}}{e^{\frac{1}{\tau y}} - 1}, \tag{7.14}$$

where μ is the desired mean firing rate, y the average firing rate of the neuron. Similar to biological neurons, we use the intracellular calcium concentration $\phi[t]$ as a good indicator of the averaged firing activity and y can be expressed with the time constant of calcium concentration τ_{cal} as

$$\phi_i[t] = (1 - \frac{1}{\tau_{cal}})\phi_i[t-1] + s_i[t], \quad y_i[t] = \frac{\phi_i[t]}{\tau_{cal}}. \tag{7.15}$$

Algorithm 1: HRMAS - Hybrid Risk-Mitigating Architectural Search

Initialize weights w, intrinsic parameters β, architectural parameters α, and correspondingly $\hat{\alpha}$. **while** *no converged* **do**
  Update $\hat{\alpha}$ by $\eta_1 \nabla_{\hat{\alpha}} \mathcal{L}_{valid}(\hat{\alpha}, w - \eta_2 \nabla_w \mathcal{L}_{train}(\hat{\alpha}, w, \beta))$;
  Update w by $\eta_2 \nabla_w \mathcal{L}_{train}(\hat{\alpha}, w, \beta)$;
  $\beta \longleftarrow$ SpiKL-IP$(\hat{\alpha}, w)$
end
Fix the motif size to the option with highest selection probability and exclude motif size from α.
Optimize the remaining architectural parameters by repeating the above.
Generate the final architecture by setting each connection type to the option with highest selection probability; Train the weights of the final architecture and evaluate performance.

We explicitly express the neuronal parameters R and τ of neuron i tuned through time as $R_i[t]$ and $\tau_i[t]$, since they are adjusted by the IP rule at each time step. They are

updated by

$$R_i[t] = R_i[t-1] - \gamma \Delta R_i, \quad \tau_i[t] = \tau_i[t-1] - \gamma \Delta \tau_i, \tag{7.16}$$

where γ is the learning rate of the SpiKL-IP rule.

In addition, by including time-variant neuronal parameters R and τ into (7.12) and (7.13), the one time step architectural parameter and weight updates change to

$$\begin{aligned}
\delta_i[t_m] &= \sum_{k=m}^{T} \frac{\partial a_i[t_k]}{\partial u_i[t_m]} \sum_{p=1}^{N} \left(\frac{R_p[t_k]}{\tau_p[t_k]} w_{pi} \delta_p[t_k]\right) \\
&\quad + \sum_{k=m}^{T-1} \frac{\partial a_i^{(l)}[t_k]}{\partial u_i^{(l)}[t_m]} \sum_{v \in \mathcal{V}} \left(\hat{\alpha}^v \sum_{r}^{O_i^v} \sum_{c \in \mathcal{C}} \frac{R_r[t_k+1]}{\tau_r[t_k+1]} \hat{\alpha}_{ri}^c w_{ri}^c \delta_r[t_k+1]\right) \\
&\Delta w_{ij} \propto \delta_i[t] \frac{R_i[t]}{\tau_i[t]} a_j[t], \quad \Delta \hat{\alpha}^v \propto \sum_{i}^{N_r} \delta_i[t] \frac{R_i[t]}{\tau_i[t]} \sum_{r}^{I_i^v} \left(\sum_{c \in \mathcal{C}} \hat{\alpha}_{ir}^c w_{ir}^c a_r[t-1]\right) \\
&\Delta w_{ir}^c \propto \delta_i[t] \frac{R_i[t]}{\tau_i[t]} \sum_{v \in \mathcal{V}} (\hat{\alpha}^v \hat{\alpha}_{ir}^c a_r[t-1]), \quad \Delta \hat{\alpha}_{ir} \propto \delta_i[t] \frac{R_i[t]}{\tau_i[t]} \sum_{v \in \mathcal{V}} (\hat{\alpha}^v w_{ir}^c a_r[t-1]).
\end{aligned} \tag{7.17}$$

Finally, the proposed alternating two-step optimization of HRMAS is summarized in Algorithm 1.

7.3 Experiments and Results

The proposed HRMAS optimized RSNNs with the SC-ML layer architecture and five motif size options are evaluated on speech dataset TI46-Alpha [56], neuromorphic speech dataset N-TIDIGITS [61], neuromorphic video dataset DVS-Gesture [63], and neuromorphic image dataset N-MNIST [59]. The performances are compared with recently reported state-of-the-art manually designed architectures of SNNs and ANNs such as feedforward SNNs, RSNNs, Liquid State Machine(LSM), and LSTM. For the proposed work, the architectural parameters are optimized by HRMAS with the weights values

trained on a training set and architectural parameters learned on a validation set as shown in Algorithm 1. The accuracy of each HRMAS optimized network is evaluated on a separate testing set with all weights reinitialized.

7.3.1 Experimental Settings

In the experiments of this chapter, the fully connected weights between layers are initialized by the He Normal initialization proposed in [85]. The recurrent weights of excitatory connections are initialized to 0.2 and tuned by the BP method. The weights of inhibitory connections are initialized to -2 and fixed. The simulation step size is set to 1 ms. The parameters like thresholds and learning rate are empirically tuned. No synaptic delay is applied for feedforward connections while recurrent connections have 1 time step delay. No refractory period, normalization, or dropout is used. Adam [68] is adopted as the optimizer. The mean of the accuracy reported is obtained by repeating the experiments five times.

Our experiments contain two phases. In the first phase, the weights are trained via the training set while the validation set is used to optimize architectural parameters. In the second phase, the motif topology and type of lateral connections are fixed after obtaining the optimal architecture. All the weights of the network are reinitialized. Then, the new network is trained on the training set and tested on the testing set. The test performance is reported in the paper. In addition, since all the datasets adopted in this chapter only contain training sets and testing sets, our strategy is to divide the training set. In the first phase, the training set is equally divided into a training subset and a validation subset. Then, the architecture is optimized on these subsets. In the second phase, since all the weights are reinitialized, we can train the weights with the full training set and test on the testing set. Note that the testing set is only used for the final evaluation.

Table 7.1 lists the typical constant values of parameters adopted in our experiments for each dataset. The SC-ML size denotes the number of neurons in the SC-ML. In our experiments, each network contains one SC-ML as the hidden layer. In addition, five motif sizes are predetermined before the experiment. The HRMAS framework optimizes the motif size from one of the five options.

Parameter	TI46-Alpha	N-TIDIGITS	DvsGesture	N-MNIST
τ_m	16 ms	64 ms	64 ms	16 ms
τ_s	8 ms	8 ms	8 ms	8 ms
τ_{cal}	16 ms	16 ms	16 ms	16 ms
learning rate	0.0005	0.0005	0.0001	0.0005
Batch Size	50	50	20	50
Time steps	100	300	400	100
Epochs for searching	300	200	60	30
Epochs for testing	400	400	150	100
SC-ML size	800	800	512	512
Motif size options	[5, 10, 16, 25, 40]		[2, 4, 8, 16, 32]	

Table 7.1: Parameters settings.

7.3.2 Results

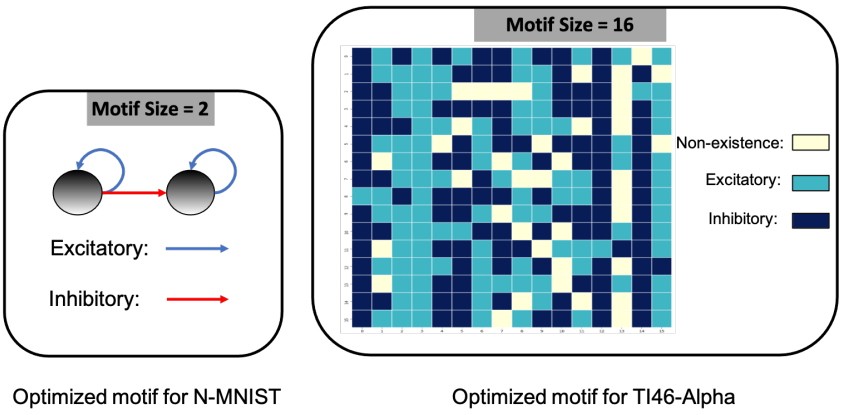

Figure 7.7: Optimized motif topologies.

Figure 7.7 presents two examples of the motif topology optimized by HRMAS over five size options $[2, 4, 8, 16, 32]$ for the N-MNIST dataset and five size options $[5, 10, 16, 25, 40]$ for the TI-Alpha dataset. The final optimized motif sizes are 2 and 16, respectively.

Table 7.2 shows the results on the TI46-Alpha dataset. The HRMAS-optimized RSNN network has one hidden SC-ML layer with 800 neurons, and outperforms all other models while achieving 96.44% accuracy with mean of 96.08% on the testing set. The proposed RSNN outperforms the LSM model of [69] and feedforward SNN of [16] by 18.44% and 6.52%, respectively. It also outperforms the larger multi-layered RSNN with more tunable parameters trained by the ST-RSBP proposed in Chapter 4 by 3.1%. In Chapter 6, the manually designed ScSr-SNN and LISR structure demonstrates improved performances using the same TSSL-BP method. Our automated HRMAS architectural search produces better performing networks.

Table 7.2: Accuracy on TI46-Alpha

Network Structure	Learning Rule	Hidden Layers	Mean	Best
LSM [69]	Non-spiking BP	2000	–	78%
Feedforward SNN [16]	HM2BP	800	89.83%	89.92%
RSNN	ST-RSBP	$400-400-400$	93.06%	93.35%
Feedforward SNN	TSSL-BP	800	–	91.05%
Sr-SNN	TSSL-BP	800	–	93.06%
Sr-SNN	TSSL-BP	$400-400-400$	94.17%	94.62%
HRMAS	TSSL-BP	800	96.08%	**96.44%**

In Table 7.3, we show that a HRMAS-optimized RSNN with a 400-neuron SC-ML layer outperforms several state-of-the-art results on the N-TIDIGITS dataset including feedforward networks trained by HM2BP [16] and RSNNs trained by ST-RSBP. Our RSNN has a more than 3% performance gain over the widely adopted recurrent structures of ANNs, the GRU and LSTM. It also significantly outperforms a feedforward SNN with the same network size, preprocessing steps, and hyperparameters trained with the same

TSSL-BP method with an accuracy improvement of almost 9.82%, demonstrating the potential of automated architectural optimization.

Table 7.3: Accuracy on N-TIDIGITS

Network Structure	Learning Rule	Hidden Layers	Mean	Best
Feedforward SNN [16]	HM2BP	$250-250$	–	89.69%
GRU [61]	Non-spiking BP	$200-200-100$	–	90.90%
Phase LSTM [61]	Non-spiking BP	$250-250$	–	91.25%
RSNN	ST-RSBP	$400-400-400$	93.63%	93.90%
Feedforward SNN	TSSL-BP	400	–	84.84%
Feedforward SNN	TSSL-BP	$400-400-400$	89.55%	89.85%
HRMAS	TSSL-BP	400	94.27%	**94.66%**

On DVS-Gesture and N-MNIST, Table 7.4 compares a HRMAS-optimized RSNN with models including feedforward SNNs trained by TSSL-BP or STBP [15] with the same size, and non-spiking vanilla RNN and LSTM. Note that although our RSNN and the LSTM model have the same number of units in the recurrent layer, the LSTM model has a much greater number of tunable parameters. Moreover, [86] proposed a rate-coding-inspired loss function to improve the performance of the reported ANNs. Our HRMAS-optimized model surpasses all other models.

7.3.3 Ablation Analysis

We conduct ablation studies on the RSNN optimized by HRMAS for the TI46-Alpha dataset to reveal the contributions of various proposed techniques. With all proposed techniques included, the HRMAS-optimized RSNN achieves 96.44% accuracy. As shown in Table 7.5, removal of the IP rule from the second step of the HRMAS optimization iteration visibly degrades the performance, showing the efficacy of intrinsic plasticity for mitigating risks of architectural changes. A similar performance degradation is observed when the sparse inter-motif connections are excluded from the SC-ML layer architecture.

Table 7.4: Accuracy on DVS-Gesture and N-MNIST

Network Structure	Learning Rule	Hidden Layers	Mean	Best
DVS-Gesture				
Feedforward SNN [86]	STBP	$P4-512$	–	87.50%
RNN [86]	Non-spiking BP	$P4-512$	–	52.78%
LSTM [86]	Non-spiking BP	$P4-512$	–	88.19%
Feedforward SNN	TSSL-BP	$P4-512$	–	88.19%
HRMAS	TSSL-BP	$P4-512$	88.40%	**90.28%**
N-MNIST				
Feedforward SNN [86]	STBP	512	–	98.19%
RNN [86]	Non-spiking BP	512	–	98.15%
LSTM [86]	Non-spiking BP	512	–	98.69%
Feedforward SNN	TSSL-BP	512	–	98.32%
HRMAS	TSSL-BP	512	98.60%	**98.72%**

Without imposing a structure in the hidden layer by using motifs as a basic building block, HRMAS can optimize all possible connectivity patterns and types of the large set of 800 hidden neurons. However, this creates a large and highly complex architectural search space, rendering a tremendous performance drop.

Finally, we compare the HRMAS model with an RSNN of a fixed architecture with full recurrent connectivity in the hidden layer. The application of the BP method is able to train the latter model since no architectural optimization is involved. However, albeit its significantly increased model complexity due to dense connections, this model has a large performance drop in comparison with the RSNN fully optimized by HRMAS.

Table 7.5: Ablation studies of HRMAS on TI46-Alpha

Setting	Accuracy	Setting	Accuracy
Without IP	95.20%	Without inter-motif connections	95.73%
Without motif	88.35%	Fully connected RSNN	94.10%

7.4 Summary and Discussions

In this chapter, we present an RSNN architecture based on SC-ML layers composed of multiple recurrent motifs with sparse inter-motif connections as a solution to constructing large recurrent spiking neural models. We further propose the automated architectural optimization framework HRMAS hybridizing the "evolution" of the architectural parameters and corresponding synaptic weights based on backpropagation and biologically-inspired mitigation of risks of architectural changes using intrinsic plasticity. We show that HRMAS-optimized RSNNs impressively improve performance on four datasets over the previously reported state-of-the-art RSNNs and SNNs. By producing sparse and scalable network architectures with optimized connectivity, this approach may contribute to development of high-performance RSNNs on both general-purpose and dedicated neuromorphic computing platforms. By sharing the Pytorch implementation with the community, the proposed HRMAS framework may motivate new advances in design of high-performance brain-inspired recurrent spiking neural models and deployment of such models on energy-efficient neurmorphic computing hardware.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

This dissertation presents computationally powerful training algorithms and systematic exploration of recurrent spiking neural networks (RSNNs) for embracing high-performance and energy-efficient spiking neuron networks. We summarize the major contributions of this dissertation as follows.

While intrinsic plasticity (IP) was attempted for spiking neurons in the past, the prior IP rules lacked a rigorous treatment in their development, and the efficacy of these rules was not verified using practical learning tasks. In Chapter 3, we address the theoretical and practical limitations of the existing works by proposing the SpiKL-IP rule. SpiKL-IP aims to tune the intrinsic parameters of a spiking neuron while minimizing the KL-divergence from the targeted exponential distribution to the actual output firing rate distribution. However, several challenges must be addressed as we work toward achieving the above goal. First, we rigorously relate the output firing rate with the static input current by deriving the firing-rate transfer function (FR-TF). FR-TF provides a basis for allowing the derivation of the SpiKL-IP rule that minimizes the KL-divergence.

Furthermore, we cast SpiKL-IP in a suitable form to enable the online application of IP tuning. Finally, we address one major challenge associated with applying SpiKL-IP under realistic contexts where the input current to each spiking neuron may be time-varying, which leads to the final IP rule that has no dependency on the instantaneous input level and effectively tuning the neural model parameters based upon averaged firing activities. Simulation studies demonstrate that the application of SpiKL-IP to individual neurons in isolation or as part of a larger spiking neural network robustly produces the desired exponential distribution. The evaluation of SpiKL-IP under real-world speech tasks shows that SpiKL-IP noticeably outperforms two existing IP rules and can significantly boost recognition accuracy by up to more than 16%.

Training of SNNs via BP is challenged by two fundamental issues. First, from an algorithmic perspective, the complex neural dynamics in both spatial and temporal domains make the BP process obscure. Moreover, the errors are hard to be precisely backpropagated due to the non-differentiability of discrete spike events. Second, a large number of time steps are typically required for emulating SNNs in time to achieve decent performance, leading to high latency and rendering spike-based computation unscalable to deep architectures. We proposed two types of BP methods to handle these problems.

In Chapter 4, we present the novel spike-train level backpropagation algorithm ST-RSBP, which can transparently train all types of SNNs including RSNNs without unfolding in time. The employed S-PSP model improves the training efficiency at the spike-train level and also addresses key challenges of RSNNs training in handling temporal effects and gradient computation of loss functions with inherent discontinuities for accurate gradient computation. More specifically, in ST-RSBP, the given rate-coded errors can be efficiently computed and back-propagated through layers without costly unfolding the network in time and through expensive time point by time point computation. Moreover, ST-RSBP handles the discontinuity of spikes during BP without altering and smoothing

the microscopic spiking behaviors. The problem of network unfolding is dealt with accurate spike-train level BP such that the effect of all spikes is captured and propagated in an aggregated manner to achieve accurate and fast training. As such, both rate and temporal information in the SNN are well exploited during the training process. Based upon challenging speech and image datasets including TI46 [56], N-TIDIGITS [61], and MNIST, ST-RSBP is able to train SNNs with an accuracy surpassing that of the current state-of-the-art SNN BP algorithms and conventional non-spiking deep learning models.

In Chapter 5, we present the novel temporal spike sequence learning via a backpropagation (TSSL-BP) method to train deep SNNs. The main purposes of the proposed TSSL-BP method are to: (1) more precisely handle the non-differentiability of the spiking activation function while keeping all-or-none firing characteristics of spiking neurons; (2) provide a way to train SNNs within a small number of time steps (i.e. latency) to reduce response time, improve accuracy and efficiency. The efficacy and precision of TSSL-BP allow it to successfully train SNNs over a very short temporal window, e.g. over 5-10 time steps, enabling ultra-low latency spike computation. As shown in Section 5.4, TSSL-BP significantly improves accuracy and runtime efficiency of BP training on several well-known image datasets of MNIST [53], NMNIST [59], FashionMNIST [54], and CIFAR10 [55]. Specifically, it achieves up to 3.98% accuracy improvement over the previously reported SNN work on CIFAR10, a challenging dataset for all prior SNNs BP methods.

To explore computationally powerful RSNN architectures, we first proposed two manually designed structures that can significantly improve network performance compared to existing works and feedforward counterparts. Then, we present a systemic architectural optimization to search for optimal RSNN structures.

In Section 6.1, we propose a new RSNN structure called Skip-Connected Self-Recurrent SNN (ScSr-SNN) to offer a simple and structured approach for designing high perfor-

mance RSNNs and mitigating the training challenges resulted from random recurrent connections as in the prior works. The main contributions of this work are: (1) We proposed the self-recurrent architecture for SNNs. The recurrence is only introduced by self-recurrent connections of individual spiking neurons, i.e., there exist no lateral connections between different neurons within a layer. We demonstrate that, with self-recurrent connections, SNNs are able to realize recurrent behaviors similar to those of more complex RSNNs while the error gradients can be more straightforwardly calculated due to the mostly feedforward nature of the network. (2) We show that the skip connections can help the formation of recurrent structures, introduce more tunability, and thus improve performance. (3) We rigorously derive the backpropagation algorithm that can handle self-recurrent connections and skip connections. Based on challenging speech datasets TI46 [56] and neuromorphic speech dataset N-TIDIGITS [61], the proposed ScSr-SNN can boost performance by up to 2.85% compared with other types of RSNNs trained by state-of-the-art BP methods.

Although ScSr-SNN is easy to implement, its connections may be so simple that it cannot exploit the full power of recurrence. In Section 6.2, we propose a novel recurrent structure called the Laterally-Inhibited Self-Recurrent Unit (LISR), which consists of one excitatory neuron with a self-recurrent connection wired together with an inhibitory neuron through excitatory and inhibitory synapses. The self-recurrent connection of the excitatory neuron mitigates the information loss caused by the firing-and-resetting mechanism and maintains the long-term neuronal memory. The lateral inhibition from the inhibitory neuron to the corresponding excitatory neuron, on the one hand, adjusts the firing activity of the latter. On the other hand, it plays as a forget gate to clear the memory of the excitatory neuron. Based on speech and image datasets commonly used in neuromorphic computing, RSNNs based on the proposed LISR improve performance significantly by up to 9.26% over feedforward SNNs trained by the proposed TSSL-BP

method with similar computational costs.

Finally, to automize enable the systemic design of large RSNNs, we propose a new scalable RSNN architecture and automated architectural optimization in Chapter 7. We compose RSNNs based on a layer architecture called Sparsely-Connected Recurrent Motif Layer (SC-ML) that consists of multiple small recurrent motifs wired together by sparse lateral connections. The small size of the motifs and sparse inter-motif connectivity leads to an RSNN architecture scalable to large network sizes. We further propose a method called Hybrid Risk-Mitigating Architectural Search (HRMAS) to systematically optimize the topology of the proposed recurrent motifs and SC-ML layer architecture. HRMAS is an alternating two-step optimization process by which we mitigate the risk of network instability and performance degradation caused by architectural change by introducing a novel biologically-inspired "self-repairing" mechanism through intrinsic plasticity. The intrinsic plasticity is introduced to the second step of each HRMAS iteration and acts as unsupervised fast self-adaption to structural and synaptic weight modifications introduced by the first step during the RSNN architectural "evolution". To the best of the our knowledge, this is the first work that performs systematic architectural optimization of RSNNs. We evaluate the proposed techniques on speech dataset TI46-Alpha [56], neuromorphic speech dataset N-TIDIGITS [61], neuromorphic video dataset DVS-Gesture [63], and neuromorphic image dataset N-MNIST [59]. The SC-ML-based RSNNs optimized by HRMAS achieve state-of-the-art performance on all of the four datasets. With the same network size, automated network design via HRMAS outperforms existing RSNNs by up to 3.38% performance improvement.

In conclusion, we expect this dissertation would help move the community forward towards high-performance spiking neural networks and energy-efficient neuromorphic computing.

8.2 Future Work

There are multiple potential directions leading to further exploration and expansion of the existing work. From a training algorithms perspective, we can explore a biologically plausible backpropagation method that not only embraces the computational power from existing BP methods but also facilitate hardware implementation and energy efficiency from a bio-inspired aspect. From the architectural perspective, with the help of our proposed HRMAS method, we can explore the optimized recurrent architecture of various datasets. Furthermore, we can even compare our automatically optimized architecture with the neural circuits discovered in biological neural systems to find the similarity and better understand the connectivity of biological neural networks.

8.2.1 Biologically Plausible Backpropagation

Our proposed learning methods and state-of-the-art training algorithms for SNNs can achieve competitive performance compared to the ANN counterparts. However, these methods all have high computational costs, require extra memory to save neuronal states for backpropagation, and cannot train networks online. These features of existing BP methods are thought to be unbiologically plausible and hinder their applications to neuromorphic hardware. SNNs have started to deliver energy-efficient, massively parallel, and low-latency solutions to AI problems, facilitated by the emerging neuromorphic hardware. To harness these computational benefits, SNNs need to be trained by learning algorithms that adhere to brain-inspired neuromorphic principles, namely event-based, local, and online computations. Therefore, there is a need for a BP method of SNNs that is biologically plausible or hardware friendly so that it can be directly implemented on neuromorphic hardware.

Several recent efforts have sought to tackle these challenges. [101] proposed a deep

learning method using forward and backward neural activity propagation. The learning rule was based on the idea that spike-timing-dependent plasticity (STDP) implements the gradient descent learning rule. [102] proposed a more biologically plausible direct feedback alignment (DFA) method for conventional ANNs where the error is more biologically plausibly fed back to each hidden layer through fixed random feedback connections directly from the output layer, reducing a bulk of the BP complexity. Furthermore, DFA can be performed for all hidden layers concurrently, reducing the backward phase latency. This idea was extended to SNNs in [103] as an event-based BP method. Furthermore, our previous work [104] adapted the ST-RSBP proposed in Chapter 4 with DFA to a hardware-friendly ST-DFA method and demonstrated it on FPGA board. In addition, [105] and [106] also tries to modify the BP method so that the training can be similar to Hebbian learning. Recently, [21] proposed a gradient-based method with a local error which is thought to be more biologically plausible than existing BP methods.

Although various approaches are proposed to build more biologically plausible and hardware-friendly training methods, the lack of SNN learning rules that can achieve both high performance and biological plausibility, hinders the neuromorphic technology from going mainstream.

8.2.2 Biologically Plausible Architecture Explored from Architectural Optimization

How neural circuits in the brain are built and function is a central pursuit of neuroscience. Many works tried to explore and understand the neural circuits from their cell types, mechanism, topology, connectivity, and so on. More recently, [107] report the complete connectomes of both sexes of a tiny roundworm — a major step towards understanding how a brain's function emerges from its form.

In our future work, we can start well-known neural circuits from neuroscience, for example, olfactory circuits. The architecture of olfactory circuits has been widely studied in recent few decades [108, 109]. In addition, different types of neuromorphic circuits inspired by olfactory circuits are proposed for odor recognition and odor location [110, 111, 112]. Olfactory circuits own several interesting characteristics. In the olfactory bulb, multiple neurons are grouped in the glomerulus with one major neuron connected to pre/post layers. Each olfactory bulb has many glomeruli located near its surface. There is typically lateral inhibition between glomeruli and excitatory recurrent connectivity within each glomerulus. As we can see, these features can be easily realized by the SC-ML structure proposed in Chapter 7. Thus, with our powerful architectural optimization method, we can build the recurrent structure optimized based on olfactory datasets and check the similarity between the optimized network and neural circuits observed biologically. We believe these experiments can help us better understand the connectivity and functions of recurrent spiking neural networks.

Bibliography

[1] W. Maass, *Networks of spiking neurons: the third generation of neural network models*, *Neural networks* **10** (1997), no. 9 1659–1671.

[2] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber, *Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation*, *IEEE Journal of Solid-State Circuits* **48** (2013), no. 8 1943–1953.

[3] 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* **102** (2014), no. 5 699–716.

[4] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, *et. al.*, *Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip*, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **34** (2015), no. 10 1537–1557.

[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* **38** (2018), no. 1 82–99.

[6] R. Kempter, W. Gerstner, and J. L. Van Hemmen, *Hebbian learning and spiking neurons*, *Physical Review E* **59** (1999), no. 4 4498.

[7] S. Huang, C. Rozas, M. Trevino, J. Contreras, S. Yang, L. Song, T. Yoshioka, H.-K. Lee, and A. Kirkwood, *Associative hebbian synaptic plasticity in primate visual cortex*, *Journal of Neuroscience* **34** (2014), no. 22 7575–7579.

[8] N. Caporale and Y. Dan, *Spike timing-dependent plasticity: a hebbian learning rule*, *Annu. Rev. Neurosci.* **31** (2008) 25–46.

[9] Y. Jin and P. Li, *Ap-stdp: A novel self-organizing mechanism for efficient reservoir computing*, in *2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 1158–1165, IEEE, 2016.

[10] G. Srinivasan, P. Panda, and K. Roy, *Stdp-based unsupervised feature learning using convolution-over-time in spiking neural networks for energy-efficient neuromorphic computing*, *ACM Journal on Emerging Technologies in Computing Systems (JETC)* **14** (2018), no. 4 1–12.

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

[12] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, *nature* **521** (2015), no. 7553 436.

[13] S. M. Bohte, J. N. Kok, and H. La Poutre, *Error-backpropagation in temporally encoded networks of spiking neurons*, *Neurocomputing* **48** (2002), no. 1-4 17–37.

[14] J. H. Lee, T. Delbruck, and M. Pfeiffer, *Training deep spiking neural networks using backpropagation*, *Frontiers in neuroscience* **10** (2016) 508.

[15] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, *Spatio-temporal backpropagation for training high-performance spiking neural networks*, *Frontiers in neuroscience* (2018).

[16] 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*, pp. 7005–7015, 2018.

[17] S. B. Shrestha and G. Orchard, *Slayer: Spike layer error reassignment in time*, in *Advances in Neural Information Processing Systems*, pp. 1412–1421, 2018.

[18] F. Zenke and S. Ganguli, *Superspike: Supervised learning in multilayer spiking neural networks*, *Neural computation* **30** (2018), no. 6 1514–1541.

[19] D. Huh and T. J. Sejnowski, *Gradient descent for spiking neural networks*, in *Advances in Neural Information Processing Systems*, pp. 1433–1443, 2018.

[20] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, *Long short-term memory and learning-to-learn in networks of spiking neurons*, in *Advances in Neural Information Processing Systems*, pp. 787–797, 2018.

[21] G. Bellec, F. Scherr, E. Hajek, D. Salaj, R. Legenstein, and W. Maass, *Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets*, *arXiv preprint arXiv:1901.09049* (2019).

[22] 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*, pp. 1–8, IEEE, 2015.

[23] 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*, pp. 1117–1125, 2015.

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

[25] P. J. Werbos, *Backpropagation through time: what it does and how to do it*, *Proceedings of the IEEE* **78** (1990), no. 10 1550–1560.

[26] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, *Deep learning in spiking neural networks*, *Neural Networks* (2018).

[27] E. Marder, L. Abbott, G. G. Turrigiano, Z. Liu, and J. Golowasch, *Memory from the dynamics of intrinsic membrane currents*, *Proceedings of the national academy of sciences* **93** (1996), no. 24 13481–13486.

[28] R. Baddeley, L. F. Abbott, M. C. Booth, F. Sengpiel, T. Freeman, E. A. Wakeman, and E. T. Rolls, *Responses of neurons in primary and inferior temporal visual cortices to natural scenes*, *Proceedings of the Royal Society of London B: Biological Sciences* **264** (1997), no. 1389 1775–1783.

[29] N. S. Desai, L. C. Rutherford, and G. G. Turrigiano, *Plasticity in the intrinsic excitability of cortical pyramidal neurons*, *Nature neuroscience* **2** (1999), no. 6 515.

[30] M. Stemmler and C. Koch, *How voltage-dependent conductances can adapt to maximize the information encoded by neuronal firing rate*, *Nature neuroscience* **2** (1999), no. 6 521.

[31] B. Schrauwen, M. Wardermann, D. Verstraeten, J. J. Steil, and D. Stroobandt, *Improving reservoirs using intrinsic plasticity*, *Neurocomputing* **71** (2008), no. 7-9 1159–1171.

[32] J. Triesch, *A gradient rule for the plasticity of a neuron's intrinsic excitability*, in *International Conference on Artificial Neural Networks*, pp. 65–70, Springer, 2005.

[33] A. Lazar, G. Pipa, and J. Triesch, *Fading memory and time series prediction in recurrent networks with different forms of plasticity*, *Neural Networks* **20** (2007), no. 3 312–322.

[34] A. J. Watt and N. S. Desai, *Homeostatic plasticity and stdp: keeping a neuron's cool in a fluctuating world*, *Frontiers in synaptic neuroscience* **2** (2010) 5.

[35] C. Li and Y. Li, *A spike-based model of neuronal intrinsic plasticity*, *IEEE Transactions on Autonomous Mental Development* **5** (2013), no. 1 62–73.

[36] C. Li, *A model of neuronal intrinsic plasticity*, *IEEE Transactions on Autonomous Mental Development* **3** (2011), no. 4 277–284.

[37] X. Li, W. Wang, F. Xue, and Y. Song, *Computational modeling of spiking neural network with learning rules from stdp and intrinsic plasticity*, *Physica A: Statistical Mechanics and its Applications* **491** (2018) 716–728.

[38] P. Panda and K. Roy, *Learning to generate sequences with combination of hebbian and non-hebbian plasticity in recurrent spiking neural networks*, *Frontiers in neuroscience* **11** (2017) 693.

[39] G. Buzsaki, *Rhythms of the Brain*. Oxford University Press, 2006.

[40] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, *Neural computation* **9** (1997), no. 8 1735–1780.

[41] H. Jaeger, *The "echo state" approach to analysing and training recurrent neural networks-with an erratum note*, *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report* **148** (2001), no. 34 13.

[42] A. Graves, A.-r. Mohamed, and G. Hinton, *Speech recognition with deep recurrent neural networks*, in *2013 IEEE international conference on acoustics, speech and signal processing*, pp. 6645–6649, IEEE, 2013.

[43] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, *Learning phrase representations using rnn encoder-decoder for statistical machine translation*, in *EMNLP*, 2014.

[44] A. Voelker, I. Kajić, and C. Eliasmith, *Legendre memory units: Continuous-time representation in recurrent neural networks*, in *Advances in Neural Information Processing Systems*, pp. 15570–15579, 2019.

[45] W. Maass, T. Natschläger, and H. Markram, *Real-time computing without stable states: A new framework for neural computation based on perturbations*, *Neural computation* **14** (2002), no. 11 2531–2560.

[46] Y. Zhang, P. Li, Y. Jin, and Y. Choe, *A digital liquid state machine with biologically inspired learning and its application to speech recognition*, *IEEE transactions on neural networks and learning systems* **26** (2015), no. 11 2635–2649.

[47] Q. Wang and P. Li, *D-lsm: Deep liquid state machine with unsupervised recurrent reservoir tuning*, in *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 2652–2657, IEEE, 2016.

[48] G. Srinivasan, P. Panda, and K. Roy, *Spilinc: Spiking liquid-ensemble computing for unsupervised speech and image recognition*, *Frontiers in neuroscience* **12** (2018).

[49] A. Morrison, M. Diesmann, and W. Gerstner, *Phenomenological models of synaptic plasticity based on spike timing*, *Biological cybernetics* **98** (2008), no. 6 459–478.

[50] F. Ponulak and A. Kasiński, *Supervised learning in spiking neural networks with resume: sequence learning, classification, and spike shifting*, *Neural computation* **22** (2010), no. 2 467–510.

[51] A. Maes, M. Barahona, and C. Clopath, *Learning spatiotemporal signals using a recurrent spiking network that discretizes time*, *PLoS computational biology* **16** (2020), no. 1 e1007606.

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

[53] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, *Proceedings of the IEEE* **86** (1998), no. 11 2278–2324.

[54] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms*, *arXiv preprint arXiv:1708.07747* (2017).

[55] A. Krizhevsky, V. Nair, and G. Hinton, *The cifar-10 dataset*, *online: http://www. cs. toronto. edu/ kriz/cifar.html* (2014).

[56] M. Liberman, R. Amsler, K. Church, E. Fox, C. Hafner, J. Klavans, M. Marcus, B. Mercer, J. Pedersen, P. Roossin, D. Walker, S. Warwick, and A. Zampolli, *TI 46-word LDC93S9*, 1991.

[57] R. Lyon, *A computational model of filtering, detection, and compression in the cochlea*, in *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'82.*, vol. 7, pp. 1282–1285, IEEE, 1982.

[58] M. Slaney, *Auditory toolbox*, *Interval Research Corporation, Tech. Rep* **10** (1998), no. 1998 1194.

[59] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, *Converting static image datasets to spiking neuromorphic datasets using saccades*, *Frontiers in neuroscience* **9** (2015) 437.

[60] P. Lichtsteiner, C. Posch, and T. Delbruck, *A 128 $\times$128 120 db 15μs latency asynchronous temporal contrast vision sensor*, *IEEE journal of solid-state circuits* **43** (2008), no. 2 566–576.

[61] J. Anumula, D. Neil, T. Delbruck, and S.-C. Liu, *Feature representations for neuromorphic audio spike streams*, *Frontiers in neuroscience* **12** (2018) 23.

[62] R. G. Leonard and G. Doddington, *Tidigits speech corpus*, *Texas Instruments, Inc* (1993).

[63] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, *et. al.*, *A low power, fully event-based gesture recognition system*, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7243–7252, 2017.

[64] A. J. Bell and T. J. Sejnowski, *An information-maximization approach to blind separation and blind deconvolution*, *Neural computation* **7** (1995), no. 6 1129–1159.

[65] A. Papoulis and S. U. Pillai, *Probability, random variables, and stochastic processes*. Tata McGraw-Hill Education, 2002.

[66] P. Dayan and L. F. Abbott, *Theoretical neuroscience*, vol. 806. Cambridge, MA: MIT Press, 2001.

[67] R. Legenstein and W. Maass, *Edge of chaos and prediction of computational performance for neural circuit models*, *Neural Networks* **20** (2007), no. 3 323–334.

[68] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, *arXiv preprint arXiv:1412.6980* (2014).

[69] P. Wijesinghe, G. Srinivasan, P. Panda, and K. Roy, *Analysis of liquid ensembles for enhancing the performance and accuracy of liquid state machines*, *Frontiers in Neuroscience* **13** (2019) 504.

[70] P. Y. Simard, D. Steinkraus, J. C. Platt, *et. al.*, *Best practices for convolutional neural networks applied to visual document analysis.*, in *ICDAR*, vol. 3, pp. 958–962, 2003.

[71] O. Booij and H. tat Nguyen, *A gradient descent rule for spiking neurons emitting multiple spikes*, *Information Processing Letters* **95** (2005), no. 6 552–558.

[72] Y. Wu, L. Deng, G. Li, J. Zhu, Y. Xie, and L. Shi, *Direct training for spiking neural networks: Faster, larger, better*, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 1311–1318, 2019.

[73] E. Hunsberger and C. Eliasmith, *Training spiking deep networks for neuromorphic hardware*, *arXiv preprint arXiv:1611.05141* (2016).

[74] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, *Going deeper in spiking neural networks: Vgg and residual architectures*, *Frontiers in neuroscience* **13** (2019).

[75] D. Fisher, I. Olasagasti, D. W. Tank, E. R. Aksay, and M. S. Goldman, *A modeling framework for deriving the structural and functional architecture of a short-term memory microcircuit*, *Neuron* **79** (2013), no. 5 987–1000.

[76] K. Daie, M. S. Goldman, and E. R. Aksay, *Spatial patterns of persistent neural activity vary with the behavioral context of short-term memory*, *Neuron* **85** (2015), no. 4 847–860.

[77] M. Oster, R. Douglas, and S.-C. Liu, *Computation with spikes in a winner-take-all network*, *Neural computation* **21** (2009), no. 9 2437–2465.

[78] Y. Chen, *Mechanisms of winner-take-all and group selection in neuronal spiking networks*, *Frontiers in computational neuroscience* **11** (2017) 20.

[79] K. Kar, J. Kubilius, K. Schmidt, E. B. Issa, and J. J. DiCarlo, *Evidence that recurrent circuits are critical to the ventral stream's execution of core object recognition behavior*, *Nature neuroscience* **22** (2019), no. 6 974–983.

[80] K. Rajaei, Y. Mohsenzadeh, R. Ebrahimpour, and S.-M. Khaligh-Razavi, *Beyond core object recognition: Recurrent processes account for object recognition under occlusion*, *PLoS computational biology* **15** (2019), no. 5 e1007001.

[81] T. Mikolov, A. Joulin, S. Chopra, M. Mathieu, and M. Ranzato, *Learning longer memory in recurrent neural networks*, *arXiv preprint arXiv:1412.7753* (2014).

[82] S. Li, W. Li, C. Cook, C. Zhu, and Y. Gao, *Independently recurrent neural network (indrnn): Building a longer and deeper rnn*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5457–5466, 2018.

[83] M. Bartos, I. Vida, and P. Jonas, *Synaptic mechanisms of synchronized gamma oscillations in inhibitory interneuron networks*, *Nature reviews neuroscience* **8** (2007), no. 1 45–56.

[84] I. Loshchilov and F. Hutter, *Decoupled weight decay regularization*, in *International Conference on Learning Representations*, 2018.

[85] K. He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[86] W. He, Y. Wu, L. Deng, G. Li, H. Wang, Y. Tian, W. Ding, W. Wang, and Y. Xie, *Comparing snns and rnns on neuromorphic vision datasets: Similarities and differences*, *Neural Networks* **132** (2020) 108–120.

[87] T. Elsken, J. H. Metzen, F. Hutter, *et. al.*, *Neural architecture search: A survey.*, *J. Mach. Learn. Res.* **20** (2019), no. 55 1–21.

[88] M. Wistuba, A. Rawat, and T. Pedapati, *A survey on neural architecture search*, *arXiv preprint arXiv:1905.01392* (2019).

[89] B. Zoph and Q. V. Le, *Neural architecture search with reinforcement learning*, in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.

[90] H. Liu, K. Simonyan, and Y. Yang, *Darts: Differentiable architecture search*, in *International Conference on Learning Representations*, 2018.

[91] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, *Regularized evolution for image classifier architecture search*, in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, 01, pp. 4780–4789, 2019.

[92] S. Tian, L. Qu, L. Wang, K. Hu, N. Li, and W. Xu, *A neural architecture search based framework for liquid state machine design*, *Neurocomputing* **443** (2021) 174–182.

[93] Y. Zhou, Y. Jin, and J. Ding, *Surrogate-assisted evolutionary search of spiking neural architectures in liquid state machines*, *Neurocomputing* **406** (2020) 12–23.

[94] N.-W. Tien and D. Kerschensteiner, *Homeostatic plasticity in neural development*, *Neural development* **13** (2018), no. 1 1–7.

[95] R. Perin, T. K. Berger, and H. Markram, *A synaptic organizing principle for cortical neuronal groups*, *Proceedings of the National Academy of Sciences* **108** (2011), no. 13 5419–5424.

[96] H. Ko, S. B. Hofer, B. Pichler, K. A. Buchanan, P. J. Sjöström, and T. D. Mrsic-Flogel, *Functional specificity of local synaptic connections in neocortical networks*, *Nature* **473** (2011), no. 7345 87–91.

[97] S. C. Seeman, L. Campagnola, P. A. Davoudian, A. Hoggarth, T. A. Hage, A. Bosma-Moody, C. A. Baker, J. H. Lee, S. Mihalas, C. Teeter, *et. al.*, *Sparse recurrent excitatory connectivity in the microcircuit of the adult mouse and human cortex*, *Elife* **7** (2018) e37349.

[98] E. O. Neftci, H. Mostafa, and F. Zenke, *Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks*, *IEEE Signal Processing Magazine* **36** (2019), no. 6 51–63.

[99] A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter, *Understanding and robustifying differentiable architecture search*, in *International Conference on Learning Representations*, 2019.

[100] A. Maffei and A. Fontanini, *Network homeostasis: a matter of coordination*, *Current opinion in neurobiology* **19** (2009), no. 2 168–173.

[101] Y. Bengio, D.-H. Lee, J. Bornschein, T. Mesnard, and Z. Lin, *Towards biologically plausible deep learning*, *arXiv preprint arXiv:1502.04156* (2015).

[102] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, *Random synaptic feedback weights support error backpropagation for deep learning*, *Nature communications* **7** (2016), no. 1 1–10.

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

[104] J. Lee, R. Zhang, W. Zhang, Y. Liu, and P. Li, *Spike-train level direct feedback alignment: sidestepping backpropagation for on-chip training of spiking neural nets*, *Frontiers in neuroscience* **14** (2020) 143.

[105] A. Tavanaei and A. Maida, *Bp-stdp: Approximating backpropagation using spike timing dependent plasticity*, *Neurocomputing* **330** (2019) 39–47.

[106] 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*, pp. 1–8, 2019.

[107] S. J. Cook, T. A. Jarrell, C. A. Brittin, Y. Wang, A. E. Bloniarz, M. A. Yakovlev, K. C. Nguyen, L. T.-H. Tang, E. A. Bayer, J. S. Duerr, *et. al.*, *Whole-animal connectomes of both caenorhabditis elegans sexes*, *Nature* **571** (2019), no. 7763 63–71.

[108] S. Nagayama, R. Homma, and F. Imamura, *Neuronal organization of olfactory bulb circuits*, *Frontiers in neural circuits* **8** (2014) 98.

[109] T. A. Cleland, *Construction of odor representations by olfactory bulb microcircuits*, *Progress in brain research* **208** (2014) 177–203.

[110] K. C. Persaud, S. Marco, and A. Gutierrez-Galvez, *Neuromorphic olfaction*, .

[111] A. Vanarse, A. Osseiran, and A. Rassau, *An investigation into spike-based neuromorphic approaches for artificial olfactory systems*, *Sensors* **17** (2017), no. 11 2591.

[112] N. Imam and T. A. Cleland, *Rapid online learning and robust recall in a neuromorphic olfactory circuit*, *Nature Machine Intelligence* **2** (2020), no. 3 181–191.