# Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

## ABSTRACT

## LOCAL LEARNING ALGORITHMS FOR STOCHASTIC SPIKING NEURAL NETWORKS

**by**
**Bleema Rosenfeld**

This dissertation focuses on the development of machine learning algorithms for spiking neural networks, with an emphasis on local three-factor learning rules that are in keeping with the constraints imposed by current neuromorphic hardware. Spiking neural networks (SNNs) are an alternative to artificial neural networks (ANNs) that follow a similar graphical structure but use a processing paradigm more closely modeled after the biological brain in an effort to harness its low power processing capability. SNNs use an event based processing scheme which leads to significant power savings when implemented in dedicated neuromorphic hardware such as Intel's Loihi chip.

This work is distinguished by the consideration of stochastic SNNs based on spiking neurons that employ a stochastic spiking process, implementing generalized linear models (GLM) rather than deterministic thresholded spiking. In this framework, the spiking signals are random variables which may be sampled from a distribution defined by the neurons. The spiking signals may be observed or latent variables, with neurons whose outputs are observed termed visible neurons and otherwise termed hidden neurons. This choice provides a strong mathematical basis for maximum likelihood optimization of the network parameters via stochastic gradient descent, avoiding the issue of gradient backpropagation through the discontinuity created by the spiking process.

Three machine learning algorithms are developed for stochastic SNNs with a focus on power efficiency, learning efficiency and model adaptability; characteristics that are valuable in resource constrained settings. They are studied in the context of

applications where low power learning on the edge is key. All of the learning rules that are derived include only local variables along with a global learning signal, making these algorithms tractable to implementation in current neuromorphic hardware.

First, a stochastic SNN that includes only visible neurons, the simplest case for probabilistic optimization, is considered. A policy gradient reinforcement learning (RL) algorithm is developed in which the stochastic SNN defines the policy, or state-action distribution, of an RL agent. Action choices are sampled directly from the policy by interpreting the outputs of the read-out neurons using a first to spike decision rule. This study highlights the power efficiency of the SNN in terms of spike frequency.

Next, an online meta-learning framework is proposed with the goal of progressively improving the learning efficiency of an SNN over a stream of tasks. In this setting, SNNs including both hidden and visible neurons are considered, posing a more complex maximum likelihood learning problem that is solved using a variational learning method. The meta-learning rule yields a hyperparameter initialization for SNN models that supports fast adaptation of the model to individualized data on edge devices.

Finally, moving away from the supervised learning paradigm, a hybrid adversarial training framework for SNNs, termed SpikeGAN, is developed. Rather than optimize for the likelihood of target spike patterns at the SNN outputs, the training is mediated by an auxiliary discriminator that provides a measure of how similar the spiking data is to a target distribution. Because no direct spiking patterns are given, the SNNs considered in adversarial learning include only hidden neurons. A Bayesian adaptation of the SpikeGAN learning rule is developed to broaden the range of temporal data that a single SpikeGAN can estimate. Additionally, the online meta-learning rule is extended to include meta-learning for SpikeGAN, to enable efficient generation of data from sequential data distributions.

# LOCAL LEARNING ALGORITHMS FOR STOCHASTIC SPIKING NEURAL NETWORKS

by
Bleema Rosenfeld

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Engineering

Helen and John C. Hartmann Department of
Electrical and Computer Engineering

May 2022

**APPROVAL PAGE**

**LOCAL LEARNING ALGORITHMS FOR STOCHASTIC SPIKING NEURAL NETWORKS**

**Bleema Rosenfeld**

---

Dr. Alexander Haimovich, Committee Chair — Date
Distinguished Professor of Electrical and Computer Engineering, NJIT

---

Dr. Bipin Rajendran, Dissertation Co-Advisor — Date
Reader in Engineering, King's College London, London, U.K.

---

Dr. Osvaldo Simeone, Dissertation Co-Advisor — Date
Professor of Information Engineering, King's College London, London, U.K.

---

Dr. Joerg Kliewer, Committee Member — Date
Professor of Electrical and Computer Engineering, NJIT

---

Dr. Hongya Ge, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, NJIT

---

Dr. Abdallah Khreishah, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, NJIT

## BIOGRAPHICAL SKETCH

**Author:** Bleema Rosenfeld

**Degree:** Doctor of Philosophy

**Date:** May 2022

**Undergraduate and Graduate Education:**

- Doctor of Philosophy in Computer Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2022
- Bachelor of Science in Computer Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2017

**Major:** Computer Engineering

**Presentations and Publications:**

B. Rosenfeld, B. Rajendran, O. Simeone, "Spiking Generative Adversarial Networks with a Neural Network Discriminator: Local Training, Bayesian Models, and Continual Meta-Learning," *submitted IEEE Transactions on Computers, Special Issue on Software, Hardware, and Applications for Neuromorphic Computing*, July 2022.

B. Rosenfeld, B. Rajendran, O. Simeone, "Fast On-Device Adaptation for Spiking Neural Networks via Online-Within-Online Meta-Learning," *IEEE Data Science and Learning Workshop*, 2021.

B. Rosenfeld, O. Simeone, B. Rajendran, "Learning First-to-Spike Policies for Neuromorphic Control Using Policy Gradients," *IEEE International Workshop on Signal Processing Advances in Wireless Communications*, 2019.

*This thesis is dedicated to my family, without whom it would have been impossible to complete. To my parents, who have shown interest in my research, unwavering belief in me, and have played an irreplaceable role in my childrens' lives at times that I've been busy with school. To my husband, Ari, whose tremendous understanding, dedication, and encouragement have enabled me to persevere and focus on my studies.*

## ACKNOWLEDGMENT

First and foremost, I would like to express my sincere gratitude to my advisors, Dr. Bipin Rajendran and Dr. Osvaldo Simeone for their advice, patience and availability throughout my studies. I am deeply grateful to my advisor Dr. Rajendran for inspiring me to pursue graduate studies in this field through his dynamic classes on neuromorphic computing. I have consistently benefited from his experience and insightful perspectives on my work and from his support and encouragement. I thank my advisor Dr. Simeone for his direction and tutelage. I have benefited immensely from his comprehensive understanding of probabilistic learning and information theory, and the opportunity to learn from his principled and grounded research style.

I extend my appreciation to the dedicated members of my PhD committee, Dr. Alexander Haimovich, Dr. Joerg Kliewer, Dr. Hongya Ge, and Dr. Abdallah Khreishah, for their insightful comments and suggestions on my work. I would especially like to acknowledge Dr. Haimovich for taking me on as a student mid way through my studies and welcoming me into his lab.

Furthermore, I am thankful to the National Science Foundation for supporting my research under grant No. 1710009. I would also like to thank the department of Electrical and Computer Engineering for awarding me the Kupfrian Endowed Fellowship in the 2017-2018 school year and for the subsequent teaching assistant opportunities they have given me that helped financially support my studies.

I would like to thank my fellow researchers Anakha V. Babu, Shruti Kulkarni, Vinay Joshi, Alireza Bagheri, Hyeryung Jang, and Nicolas Skatchkovsky for their camaraderie, interesting discussions, and assistance during our time together.

Lastly, I am so appreciative to my family for their support and encouragement throughout my studies.

## TABLE OF CONTENTS

**TABLE OF CONTENTS**
**(Continued)**

**Chapter** **Page**

## TABLE OF CONTENTS
## (Continued)

**Chapter** **Page**

## LIST OF TABLES

**Table** **Page**

## LIST OF FIGURES

## LIST OF FIGURES
## (Continued)

**Figure** **Page**

## LIST OF FIGURES
## (Continued)

**Figure** **Page**

## LIST OF FIGURES
## (Continued)

**Figure** **Page**

# CHAPTER 1

# INTRODUCTION

## 1.1 Spiking Neural Networks

A spiking neural network (SNN) is a form of graphical model that is inspired by the connectionist view of cognition in the biological brain and the dynamics of biological neurons [101, 46, 75]. These models are closely related to the well know artificial neural networks (ANNs) that have driven the large advances in the field of machine learning and artificial intelligence over the past decade [113]. Like ANNs, SNNs are characterized by the ability to learn connection strengths between nodes as a means to extract features of a signal that are important for prediction and inference. In SNNs the nodes are modeled more closely after biological neurons such that they process binary signals (known as spikes) and each node integrates incoming signals at the time of the event to compute its output. These attributes underlie two compelling advantages of SNNs. Firstly, due to an SNN's event-driven activity, its energy consumption depends mostly on the number of spikes that are output by its neurons. This is because the idle energy consumption of neuromorphic chips that have been developed as dedicated SNN processors is generally extremely low (see, e.g., [80, 24]) making SNNs ideal for low power inference and learning on the edge. Second, through time domain processing, SNNs support prediction and inference for naturally temporal signals such as event based vision and audio signals [125, 76, 33], biological signals [111, 18], or radar [4].

An SNN includes a set of nodes, or neurons, and a set of weighted directed edges that define the connection strengths between them. There are various mathematical models for neuronal dynamics that are used to define the behavior of each node [39], from detailed models like the Hodgkin Huxley model [45] to much simpler models

such as the leaky integrate and fire (LIF) model [1]. In general, the neuron model includes the membrane potential which is computed as a function of input signals (exogenous or from other nodes) and connection weights, as well as a process by which output signals are generated. Biological neural spiking is inherently stochastic, yet historically, spiking neuron models have treated spike generation as a deterministic process such that spiking is governed by a set of differential equations [45] or modeled as a discontinuous threshold gated process [1]. Research has shown that stochasticity in neural network models in fact contributes to creative problem solving and is useful for sampling in probabilistic inference problems [95]. In this work, a stochastic neuron model is adopted and these potential benefits are explored. The neurons considered here are modeled using a probabilistic Generalized Linear Model (GLM) [97], a generalization of the Spike Response Model (SRM) [39], in which spike generation is a probabilistic process [52].

The spiking neurons process binary, time series signals rather than static real valued signals such as those processed by ANNs. In addition to the standard spatial information encoding, these spatio-temporal signals encode information in the spike timing. Each node applies a filter to incoming signals, called the pre-synaptic signals, to compute their effect on the instantaneous membrane potential as a function of the spike timing. The membrane potential is also affected by the previous output of the neuron, called the post-synaptic signals, by applying a filter to those signals as well. A classic example of a synaptic filter is the exponential decay filter which gives more weight to recent spikes to cause an increase in the membrane potential via the pre-synaptic spikes, or to decrease it to simulate refractoriness in the case of filters applied to post-synaptic spikes. In deterministic spiking models an output signal is generated when the membrane potential exceeds a threshold value, at which point it is also reset to zero (or some fixed reset value). In the probabilistic model that is the focus of this work, the spiking signal of each neuron is a binary random

variable that is sampled from a Bernoulli distribution with probability defined as a function of the membrane potential. The binary processing framework implemented by both deterministic and stochastic SNNs is significantly different from classic neural network processing and raises several challenges unique to SNN optimization.

### 1.1.1 Challenges

Here we discuss some of the main challenges in the development of machine learning algorithms for SNNs that will be addressed in this work. These are

- Encoding real valued signals into the spike domain and decoding or interpreting the spiking output of the SNN.
- The discontinuity presented by the spiking process renders optimization via backpropagation of errors to be inapplicable.
- Neuromorphic chips require local update rules to support energy efficient learning.

*Encoding and Decoding* Data is generally available in the form of real valued signals (e.g., RGB pixel values, amplitude of audio) that are incompatible with the spike based processing implemented by SNNs. While there are a variety of fixed encoding and decoding schemes (see [5] for a survey of encoding methods) the choice of encoder and decoder is a hyperparameter that must be tuned to the learning problem. Specifically, in supervised learning problems, the target output must be expressed in terms of spikes which poses the problem of defining targets that balance sparsity to support energy efficiency and optimized learning. One way to address this trade-off is using the first to spike decision rule which is discussed in Section 2.2.1.

*Spiking Discontinuity* The process by which spikes are generated by the neurons in SNNs introduces a discontinuity into the mathematical model. This is most often the case because the spike is artificially generated when the membrane potential crosses a threshold value (LIF and SRM neurons). In the case of a stochastic spiking

process, the discontinuity arises because the spike is a random variable. As a result of the spiking discontinuity, the functions realized by SNNs are non-differentiable with the implication that the standard method of backpropagation of error gradients used for stochastic optimization in ANNs is not readily applicable to SNN optimization. There are many algorithms for supervised SNN learning that suggest the use of a surrogate gradients as a solution to this problem (e.g., [90, 134, 30]). Others eschew backpropagation for more biologically plausible unsupervised algorithms based on spike time dependent plasticity (STDP) [12, 77]. We approach this issue by applying probabilistic inference to a stochastic spiking network in which the optimization is a function of the membrane potential of the neurons rather than of the spikes themselves (see Chapter 2).

*Local Learning* Local learning refers to learning algorithms in which the parameters associated with each neuron are updated only as a function of values that are local to that neuron such as the pre- and post-synaptic spikes and membrane potential of that neuron. This is in contrast to update rules such as those derived using backpropagation of errors in which the parameters are each updated as a function of values from downstream nodes in the network as well. Local learning is a constraint applied by state of the art neuromorphic hardware [24] to support low latencies and energy efficiency by avoiding expensive memory accesses.

## 1.2 Related Work

We propose machine learning algorithms for SNNs with a basis in probabilistic learning that is centered around the log likelihood of observed data. A key feature of the learning rules that we derive is their dependence on only local values and a global learning signal, described as a three-factor learning rule. In Section 1.2.1 We review the basics of three-factor learning rules as well as their basis in spike time dependant plasticity (STDP). We also discuss three main classes of SNN learning

algorithms, namely, surrogate gradient error backpropagation with a highlight on the literature that develop on-chip learning rules, unsupervised STDP, and ANN-to-SNN conversion. In Section 1.2.2 we review related work in probabilistic learning for SNNs.

### 1.2.1 SNN Training Methods

Spike time dependent plasticity is a biologically inspired mechanism by which the strength of synaptic connections between neurons depends on the relative timing of the pre- and post-synaptic spikes [120, 86, 79]. The presence of STDP modulated connection strengths in various neural circuits has been well demonstrated [20]. STDP is implemented in SNN models in various forms either as an exact accounting for the difference in spike timing [120, 86] which can be generalized to apply to filtered spike trains [86], or as a soft condition on spike order [79]. Three-factor learning rules include an additional term that is attributed as a global error signal which has its biological basis in various forms of neuromodulation [38].

Given the massive success of backpropagation based algorithms in deep learning, a popular direction of research has been to apply backpropagation to achieve the same results in SNNs through direct optimization. This is complicated by the fact that backpropagation is not readily applicable to SNNs because of their inherently discontinuous gradient. A variety of direct SNN training algorithms have been developed that employ a smooth approximation to the spiking behavior or the discontinuous portion of the gradient to adapt backpropagation to SNNs [90, 9, 115, 134]. References [115, 134] successfully apply backpropagation with some form of surrogate gradient to minimize a van Rossum [129] like error on the output spike trains. They thereby train large feedforward and convolutional SNNs that achieve high accuracy in tasks such as classification of MNIST, NMNIST, DVS Gesture and TIDIGITS. Reference [9] applies an adaptation of backpropagation through time to a recurrent SNN and shows its application for long and short term

sequential learning problems such as sequential MNIST and TIMIT as well as an extension to the learning to learn framework.

The main problem with the use of these backpropagation algorithms for SNNs is that they do not inherently support local learning rules which are a requirement in current neuromorphic hardware. There have been several recent works that propose adaptations to make backpropogation algorithms for SNNs compatible with on-chip learning. In [124], the read-out layer of an SNN is trained on-chip using a three-factor rule that includes the direct error between the SNN output spikes and the target spikes while the hidden layers are still pre-trained offline. In [10], eligibility traces are used to maintain local error signals. Others, such as [114, 138], experiment with heuristic error feedback schemes such as feedback alignment [69] or straight through estimator [11] to remove the dependence of the backpropagated error signal on downstream values.

SNNs can also be trained in an unsupervised manner through STDP alone, thus avoiding the problem of error assignment over time. The goal in unsupervised SNN learning is to extract correlations between the spike timing that are descriptive of the features of the data to support classification or clustering. After learning a representation of the data another phase of supervised learning is necessary to make practical use of it. Unsupervised STDP learning has recently been applied to deep convolutional SNNs [67, 28] for standard MNIST classification [67] and speech recognition [28]. Given that STDP based update rules make use of only local variables it is a good candidate for implementation in neuromorphic hardware [56].

One standard approach to achieving low power inference by means of an SNN while bypassing the challenges associated with SNN learning is to convert an offline trained ANN into an integrate and fire SNN. Conversion aims at ensuring that the output spiking rates of the neurons in the SNN are proportional to the numerical values output by the corresponding neurons in the ANN [27, 19]. While this approach

has been shown to produce effective SNNs with high test accuracy, its reliance on rate coded spike signals necessitates long processing time to achieve these accuracies as well as limits it's applicability to domains where information is encoded in the spike timing. More recent work has begun to address the issue of long latencies [44, 105] and [143] has realized the converted SNN on a neuromorphic chip. These conversion methods represent a way in which to leverage a fixed SNN for low power inference. However, since an ANN must still be originally trained, they do not take advantage of the promise for energy efficient learning shown by SNNs.

### 1.2.2 Probabilistic Learning for SNNs

Several probabilistic neuron models were introduced and studied in the context of computational neuroscience such as bayesian spiking neurons (BSN) [25, 91], the stochastic spike response model (SRM) [17] and the generalized linear model (GLM) [98]. BSNs and stochastic SRMs have been used for machine learning with the model being trained either via unsupervised STDP [63, 127, 2, 99] or top-down gradient methods [48]. Aside from the biological plausibility of probabilistic synapses [73] it has recently been proposed that synaptic plasticity in the brain implements some form of probabilistic learning [141]. Probabilistic learning methods include unsupervised learning [139], contrastive divergence to estimate maximum likelihood learning for LIF neurons [89] and stochastic sampling to optimize for sparse network connectivity where the parameters are learned via backpropagation through time [8].

## 1.3 Overview of Algorithms Developed

### 1.3.1 Stochastic Policy Based Reinforcement Learning

Artificial neural networks are popular as function approximators in many state-of-the-art reinforcement learning (RL) algorithms. Due the low energy profile of spiking neural networks, they are considered to be important candidates as co-processors to

be implemented in mobile devices that could benefit from online experience based learning. In this study, the use of SNNs as stochastic policies is explored under an energy-efficient first-to-spike action rule, whereby the action taken by the RL agent is determined by the occurrence of the first spike among the output neurons. A policy gradient-based algorithm is derived considering a generalized linear model for spiking neurons. Experimental results demonstrate the capability of online trained SNNs as stochastic policies to gracefully trade energy consumption, as measured by the number of spikes, and control performance. Significant gains are shown as compared to the standard approach of converting an offline trained ANN into an SNN.

### 1.3.2 Online-Within-Online Meta-Learning for Classification

The popularity of spiking neural networks as machine learning models for on-device edge intelligence suggests that they may be used for applications such as mobile healthcare management and natural language processing. In such highly personalized use cases, it is important for the model to be able to adapt to the unique features of an individual with only a minimal amount of training data. Meta-learning has been proposed as a way to train models that are geared towards quick adaptation to new tasks. The few existing meta-learning solutions for SNNs operate offline and require some form of backpropagation that is incompatible with the current neuromorphic edge-devices. This study proposes an *online-within-online meta-learning* rule for SNNs termed OWOML-SNN, that enables lifelong learning on a stream of tasks, and relies on *local, backprop-free, nested updates*.

### 1.3.3 Spatio-Temporal Generative Adversarial Learning

Neuromorphic data carries information in spatio-temporal patterns encoded by spikes. Accordingly, a central problem in neuromorphic computing is training SNNs to reproduce spatio-temporal spiking patterns in response to given spiking

stimuli. Most existing approaches model the input-output behavior of an SNN in a deterministic fashion by assigning each input to a specific desired output spiking sequence. In contrast, in order to fully leverage the time-encoding capacity of spikes, this work proposes to train SNNs so as to match *distributions* of spiking signals rather than individual spiking signals. To this end, the paper introduces a novel hybrid architecture comprising a conditional generator, implemented via an SNN, and a discriminator, implemented by a conventional artificial neural network (ANN). The role of the ANN is to provide feedback during training to the SNN within an adversarial iterative learning strategy that follows the principle of generative adversarial network (GANs). In order to better capture multi-modal spatio-temporal distributions, the proposed approach – termed SpikeGAN – is further extended to support Bayesian learning of the generator's weights. Finally, settings with time-varying statistics are addressed by proposing an online meta-learning variant of SpikeGAN. Experiments bring insights into the merits of the proposed approach as compared to existing solutions based on (static) belief networks and maximum likelihood (or empirical risk minimization). In our experiments, handwritten digit images generated by SpikeGAN are observed to train an ANN classifier with 20% higher accuracy than a comparable belief network. Our experiments also demonstrate the use of SpikeGAN to generate neuromorphic data sets from handwritten digits. It is shown that these data can be used to train an SNN classifier that achieves an accuracy level approaching the baseline accuracy of an SNN classifier trained on rate-encoded real data.

# CHAPTER 2

# BACKGROUND AND MODELS

## 2.1 Neuron Models

### 2.1.1 Generalized Linear Model (GLM) Neuron

In this work, we have considered a probabilistic spiking neuron that implements a generalized linear model (GLM) [98, 55, 51], termed GLM neurons. The neurons process data in the form of binary signals (spikes) over processing time $\tau = 1, 2, ...,$ with each neuron $i$ producing an output spike, $s_{i,\tau} = 1$, or no output spike, $s_{i,\tau} = 0$, at each time $\tau$. As depicted in Figure 2.1, each neuron includes a set of pre-synaptic connections, by which signals from exogenous inputs and other neurons in the network are passed to it, as well as an auto-feedback connection through which the neuron processes its own previous outputs. Pre-synaptic and feedback spikes are integrated via time domain filtering to update the membrane potential at every time $\tau$, giving the instantaneous membrane potential for neuron $i$

$$u_{i,\tau} = \sum_j \alpha_{j,i} * s_{j,\tau-1} + \beta_i * s_{i,\tau-1} + \gamma_i, \tag{2.1}$$

where $\alpha$ and $\beta$ are trainable pre-synaptic and feedback filters, respectively, and $\gamma_i$ is a trainable bias. The expression $\alpha_{j,i} * s_{j,\tau-1}$ denotes the convolution of filter $\alpha_{j,i}$ over a window of $\tau_w$ previous output signals from neuron $j$. Hence, the shape of the synaptic filters $\alpha_{j,i}$ define the response of the membrane potential of neuron $i$ to the output spikes of pre-synaptic neurons $j$. Similarly, the shape of the feedback filter $\beta_i$ defines the response of neuron $i$ to its own previous outputs which can support biological mechanisms such as a refractory period or bursting.

Each pre-synaptic filter is defined as a linear combination of a set of $K_a$ basis functions collected as columns of matrix $A$, such that we have $\alpha_{j,i} = Aw^{\alpha}_{j,i}$ where $w^{\alpha}_{j,i}$

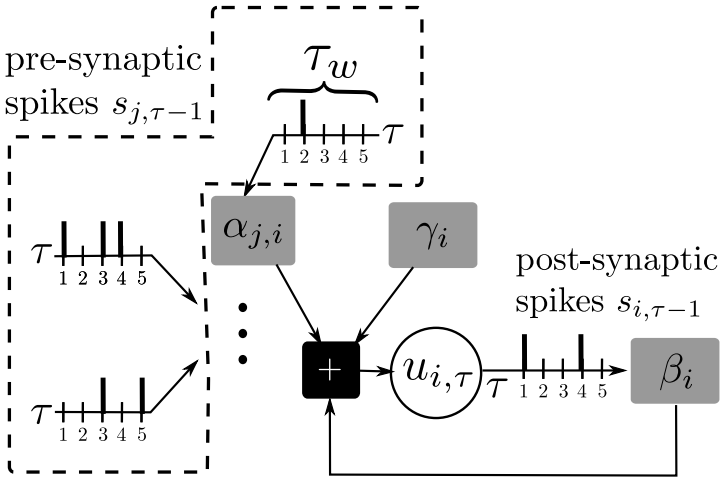


**Figure 2.1** A pictorial representation of a GLM neuron. At each time $\tau$, filters $\alpha_{j,i}$ and $\beta_i$ (gray rectangles) process incoming pre-synaptic signals $s_{j,\tau-1}$ and post-synaptic signals $s_{i,\tau-1}$ respectively. The filter outputs are accumulated along with a bias term $\gamma_i$ to compute the membrane potential of the neuron represented by the white circle labeled $u_{i,\tau}$.

is a $K_a \times 1$ vector of trainable synaptic weights [98]. A specialized case would be the use of a single basis function $K_a = 1$ which can then directly incorporate the weight $w^{\alpha}_{j,i}$. One such example that has been implemented in this work is the exponential decay kernel $\alpha_{j,i} = w^{\alpha}_{j,i} \exp(-\tau/\tau_f)$ with decay rate parameter $\tau_f$. The basis functions may also be defined as a set of raised cosine functions as depicted in Figure 2.2 which allows a detailed and distinct synaptic response and memory to be learned for each connection [97]. Unless otherwise specified, these raised cosine basis function are employed in all simulations throughout this work. The post-synaptic feedback filter $\beta_i = B w^{\beta}_i$ is defined similarly. All of the model parameters are collected in vector $\phi = \{w^{\alpha}, w^{\beta}, \gamma\}$ while parameters of individual neurons $i$ are written as $\phi_i$.

In the GLM neuron, a post-synaptic sample $s_{i,\tau}$ is a random variable whose probability is dependent on the spikes integrated by that neuron. It is defined as a probabilistic function of the neuron's membrane potential $u_{i,\tau}$ at that time as

$$p_{\phi}(s_{i,\tau}|s_{\leq \tau-1}) = p_{\phi}(s_{i,\tau} = 1|u_{i,\tau}) = \sigma(u_{i,\tau}), \tag{2.2}$$

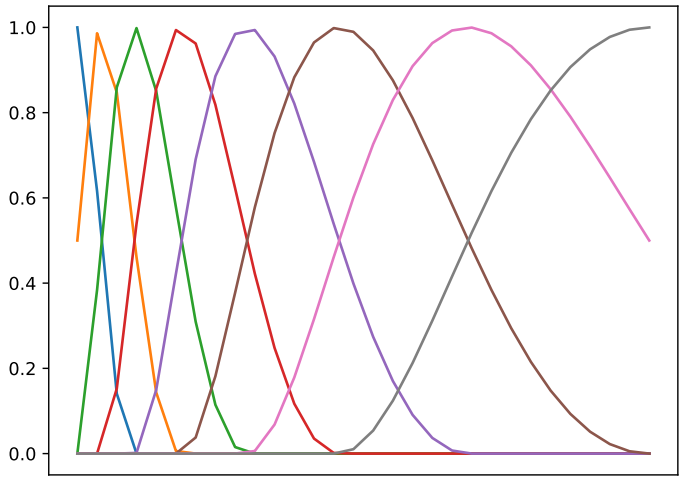


**Figure 2.2** An example of a set of raised cosine basis functions for synaptic filtering.

where $\sigma(x) = (1 + e^{-x})^{-1}$ is the logistic sigmoid function, and $\phi$ is the vector of trainable model parameters. Building on this, the log likelihood of spikes output by all neurons over a defined time period $T$, can be written as

$$\log p_\phi(s_{\leq T}) = \sum_{\tau=1}^{T} \sum_i \log p_{\phi_i}(s_{i,\tau} \mid s_{\leq \tau-1}). \tag{2.3}$$

A set of GLM neurons and the connections between them define the trainable SNN models considered in this work. The neurons are classified into read-out neurons $\mathcal{R}$ and hidden neurons $\mathcal{H}$ with respective spiking outputs denoted as $s_{i,\tau} = x_{i,\tau},\ i \in \mathcal{R}$ and $s_{i,\tau} = h_{i,\tau},\ i \in \mathcal{H}$. In learning settings in which fixed target output spike trains are observed for the read out neurons, the read-out neurons are classified instead as visible neurons $\mathcal{V}$. The observed spikes are denoted $s_{i,\tau} = v_{i,\tau},\ i \in \mathcal{V}$.

### 2.1.2 Integrate and Fire Neuron

The spiking behavior of an integrate and fire (IF) neuron can be described by an internal membrane potential defined as in Equation (2.1) with the key differences that: $(i)$ the synaptic kernels are perfect integrators, that is, they are written as $\alpha_{i,j} = w_{i,j}1$, where $w_{i,j}$ is a trainable synaptic weight and 1 is an all-one vector of $T$ elements; $(ii)$ the neuron spikes deterministically when the membrane potential is

positive, so that parameter $\gamma_j$ plays the role of negative threshold; and (*iii*) there is no auto-feedback term, rather the membrane potential is reset to a reset voltage after an output spike.

## 2.2 Probabilistic Optimization Basics for Stochastic SNNs

The log likelihood of the output spikes of the probabilistic SNN (Equation 2.3) is central to the optimization of the model to produce a desired output in various learning settings. The gradient of the log likelihood with respect to the model parameters is presented in this section as a building block for optimization via stochastic gradient descent (SGD) in later sections via maximum likelihood learning and other methods. The desired output of the SNN may be defined by a stream of observed read out spikes $\{v_{i,\tau}\}_{\tau=1}^{T}$ $i \in \mathcal{V}$ or more broadly by a certain output criterion that informs a set of observed spike streams. One such output criterion that is considered in this work is the first-to-spike decision criterion [6, 79] which is discussed in Section 2.2.1. The gradients of the log likelihood for the case of observed output spike streams are given in Section 2.2.2. An extension using a variational optimization technique to SNNs that include latent variables as well is discussed in Section 2.2.3.

### 2.2.1 First to Spike Decisions

First-to-spike desicion making is an output criterion that takes the decision or choice associated with the neuron that spikes first within the sampling time period, $T$. The use of a single spike decision rule would support lower latencies and improved power efficiency when implemented in neuromorphic hardware. Notably, this criterion allows a set of acceptable output spike streams for each neuron within a single sampling period rather than a single target spike stream per neuron. This is why it must be treated separately from the case of an observed stream of read-out variables.

The first to spike criterion is considered only for networks in which all read-out variables are observed. During training, the spikes of the read-out neurons are clamped to the first to spike criterion associated with each time $\tau$, requiring that there are never any output spikes before the current time. When the model is used for inference, a decision is made as soon as at least one of the read-out signals is sampled as a spike, again ensuring that there are no spikes at previous timesteps. Because there are never any past output spikes from a given neuron $i$, the auto-feedback term $\beta_{j,i} * s_{i,\tau-1}$, of the GLM neuron membrane potential (Equation 2.1) becomes irrelevant in this setting and is dropped.

The probabilistic spiking behavior under the GLM neuron model (given in Equation 2.2) supports the mathematical description of the probability that a desired neuron, $k$, spikes before any of the other neurons. That probability is a sum of the probabilities of observing the spike stream that follows the first to spike criterion at each time $\tau$, given as $P_{\text{first to spike}}(k) = \sum_{\tau=1}^{T} p_\tau(k)$, where

$$p_\tau(k) = \prod_{i \neq k} \prod_{\tau'=1}^{\tau} (1 - \sigma(u_{i,\tau'}))\sigma(u_{k,\tau}) \prod_{\tau'=1}^{\tau-1} (1 - \sigma(u_{k,\tau'})) \tag{2.4}$$

is the probability that the $k$th output neuron spikes for the first time at time $\tau$, while the other neurons do not spike at or before that time.

For optimization purposes, the gradient of the log likelihood of the first-to-spike decision criteria is computed as in [6] giving the gradient expressions

$$\nabla_{w^{\alpha}_{j,i}} \log P_{\text{first to spike}}(k|x,\phi) = \begin{cases} -\sum_{\tau=1}^{T} h_\tau \sigma(u_{i,\tau}) A^T \vec{s}_{j,\tau-1} & i \neq k \\ -\sum_{\tau=1}^{T} (h_\tau \sigma(u_{i,\tau}) - q_\tau) A^T \vec{s}_{j,\tau-1} & i = k, \end{cases} \tag{2.5}$$

and

$$\nabla_{b_i} \log P_{\text{first to spike}}(j|x,\phi) = \begin{cases} -\sum_{\tau=1}^{T} h_\tau \sigma(u_{i,\tau}) & i \neq k \\ -\sum_{\tau=1}^{T} h_\tau \sigma(u_{i,\tau}) - q_\tau & i = k \end{cases} \tag{2.6}$$

where

$$h_\tau = \sum_{\tau'=\tau}^{T} q_{\tau'}, \text{ and } q_\tau = \frac{p_\tau}{\sum_{\tau=1}^{T} p_\tau}$$

and $\vec{s}_{j,\tau-1} = [s_{j,\tau-1}, s_{j,\tau-2}, ..., s_{j,\tau-\tau_w}]^T$ is the $\tau_w \times 1$ window of pre-synaptic spikes that were processed at time $\tau$.

### 2.2.2 Observed Output Spikes

In some learning settings a target output spike stream is given for each of the read-out neurons over a defined time period $T$. These target spike streams may be natively temporal information or they may be defined to represent a choice or a real value based on a fixed decoding scheme. In order to train the probabilistic SNN to support the given spike streams, the outputs of the read-out neurons are considered to be observed variables that are clamped to the target spikes denoted as $\{v_{i,\tau}\}_{\tau=1}^{T}$, $i \in \mathcal{V}$ where the read-out neurons are now classified as visible neurons $\mathcal{V}$.

Considering an SNN that only includes visible neurons, the gradient of the log likelihood over the observed spikes with respect to the SNN parameter vector $\phi$ is given as

$$\nabla_\phi \mathcal{L}(\phi) = \sum_{\tau=1}^{T} \sum_{i \in \mathcal{V}} \nabla_\phi \log p_\phi(v_{i,\tau} \mid u_{i,\tau}). \tag{2.7}$$

This expression of the gradient evaluated for each component of the parameter vector for each neuron $i$ at time $\tau$ is given as in [51] as

$$\begin{aligned}
\nabla_{w_{j,i}^{\alpha}} \log p_{\theta_i}(v_{i,\tau} \mid u_{i,\tau}) &= A^T \vec{s}_{j,\tau-1}(v_{i,\tau} - \sigma(u_{i,\tau})) \\
\nabla_{w_i^{\beta}} \log p_{\theta_i}(v_{i,\tau} \mid u_{i,\tau}) &= B^T \vec{s}_{i,\tau-1}(v_{i,\tau} - \sigma(u_{i,\tau})) \\
\nabla_{\gamma_i} \log p_{\theta_i}(v_{i,\tau} \mid u_{i,\tau}) &= (v_{i,\tau} - \sigma(u_{i,\tau})).
\end{aligned} \tag{2.8}$$

These expressions include only local variables, namely a post-synaptic error term $(v_{i,\tau} - \sigma(u_{i,\tau}))$ and a pre-synaptic term $A^T \vec{s}_{j,\tau-1}$, which is the filtered $\tau_w \times 1$ window

of pre-synaptic spikes $\vec{s}_{j,\tau-1} = [s_{j,\tau-1}, s_{j,\tau-2}, ..., s_{j,\tau-\tau_w}]^T$ at time $\tau$. They therefore support parameter update rules that follow Hebbs theory of synaptic plasticity through associative spiking.

### 2.2.3 Latent and Observed Output Spikes

In probabilistic learning, the distribution that is chosen to describe the likelihood of the observed variables has a large impact on the goodness of the learned solution. To this end, in some problems we make the assumption that the observed variables are jointly distributed with latent variables. Here we review the variational learning rule that is derived in [51] for probabilistic SNNs that include latent variables which is adapted for the algorithms proposed in the following chapters.

In order to obtain the log likelihood over the observed variables in a joint distribution, we must marginalize over all possible values of the latent variables as $\mathcal{L}_{\upsilon_{\leq T}}(\phi) = \log p_\phi(\upsilon_{\leq T}) = \log \sum_{h_{\leq T}} p_\phi(\upsilon_{\leq T}, h_{\leq T})$. As in the previous section, to support probabilistic optimization, we would like to take the gradient of the log likelihood over the observed variables with respect to the parameter $\phi$, however this is intractable given the summation over the latent variables within the log expression. For this reason, we introduce a variational posterior over the latent variables $q_\theta(h_{\leq T} \mid \upsilon_{\leq T})$ with parameter $\theta$ and estimate the log likelihood $\mathcal{L}_{\upsilon_{\leq T}}(\phi)$ using the Evidence Lower BOund (ELBO) as (see e.g., [116])

$$\mathcal{L}_{\upsilon_{\leq T}}(\phi) \geq \sum_{h_{\leq T}} q_\theta(h_{\leq T}) \log \left( \frac{p_\phi(\upsilon_{\leq T}, h_{\leq T})}{q_\theta(h_{\leq T})} \right) \coloneqq L_{\upsilon_{\leq T}}(\phi, \theta) \tag{2.9}$$

which gives the expected value of a learning signal

$$\ell_{\phi,\theta}(\upsilon_{\leq T}, h_{\leq T}) = \log p_\phi(\upsilon_{\leq T}, h_{\leq T}) - \log q_\theta(h_{\leq T}) \tag{2.10}$$

over the latent variables.

Following [51], the variational distribution $q_\theta(h_{\leq T} \mid \upsilon_{\leq T})$ is chosen to be

$$q_{\phi_H}(h_{\leq T} \mid \upsilon_{\leq T}) = \prod_{\tau=1}^{T} p_{\phi_H}(h_\tau \mid h_{\leq \tau-1}, \upsilon_{\leq \tau-1}) = \prod_{\tau=1}^{T} \prod_{i \in \mathcal{H}} p_{\phi_i}(h_{i,\tau} \mid u_{i,\tau}) \tag{2.11}$$

where the variational parameter $\theta$ is now the parameter $\phi_H$ which denotes the set of parameters of a set of hidden neurons $\mathcal{H}$ where $\phi_H = [\phi_i]_{i \in \mathcal{H}}$. This distribution ignores the dependance of $h_\tau$ on values of $\upsilon$ that occur later in the sequence which allows it to be modeled by a set of hidden neurons $\mathcal{H}$ with membrane potential defined as in Equation (2.1). As a result, the latent variables can be easily sampled from the distribution defined by the hidden neurons as in Equation (2.2), with $s_{i,\tau} = h_{i,\tau}$. Substituting the variational distribution in Equation (2.11) into Equation (2.10) gives the specific learning signal

$$\ell_{\phi_V}(\upsilon_{\leq T}, h_{\leq T}) = \sum_{\tau=1}^{T} \log p_{\phi_V}(\upsilon_{\leq \tau} \mid h_{\leq \tau-1}, \upsilon_{\leq \tau-1}) \tag{2.12}$$

where $\phi_V$ is the set of the visible neuron parameters $\phi_V = [\phi_i]_{i \in \mathcal{V}}$.

The gradients of the ELBO with respect to the visible and hidden neuron parameters, $\phi_V$ and $\phi_H$ respectively, are used in place of the gradient of the log likelihood for optimization of probabilistic SNNs with hidden neurons. However, the exact gradients still include an average over the latent variables which would make computation intractable for the models considered in this work. Therefore, the expectation is estimated using Monte Carlo sampling of the latent variables from the distribution defined by the hidden neurons. For a single sample of the latent variables, $h_{\leq T}$, the gradients are then given as

$$\nabla_{\phi_V} L_{\upsilon_{\leq T}}(\phi_V, \phi_H) = \nabla_{\phi_V} \log p_{\phi_V}(\upsilon_{\leq T}) \tag{2.13a}$$

and

$$\nabla_{\phi_H} L_{\upsilon_{\leq T}}(\phi_V, \phi_H) = \ell_{\phi_V}(\upsilon_{\leq T}, h_{\leq T}) \nabla_{\phi_H} \log q_{\phi_H}(h_{\leq T}) \tag{2.13b}$$

where Equation (2.13b) is derived using the standard REINFORCE gradient [116]. The gradient in Equation (2.13a), is the same as in the case where only observed variables are considered and includes only the local gradient of the log likelihood defined by the visible neurons which is given by Equation (2.8). The gradient in Equation (2.13b) evaluates to a three-factor rule in which the global learning signal given in Equation (2.12) modulates the contribution of the local gradient of the log likelihood defined by the hidden neurons. These local gradients for the hidden neurons are the same as in Equation (2.8) with $\upsilon_{i,\tau}$ replaced by $h_{i,\tau}$. The global learning signal can be interpreted as an evaluation of how well the latent variables sampled from the hidden neurons support the observed variables.

# CHAPTER 3

# STOCHASTIC POLICY GRADIENT REINFORCEMENT LEARNING

We begin the study of local learning algorithms for stochastic SNNs comprised of GLM neurons by considering the simplest case in which the spiking outputs of all neurons are observed variables. We choose to develop a reinforcement learning rule because of its many downstream applications (reviewed in Section 3.1) that require energy efficient edge computing solutions that may be improved by an SNN implementation. We adopt the first-to-spike decision rule detailed in Section 2.2.1 which supports the derivation of a local three-factor learning rule based on the principles of maximum likelihood learning. This choice of decision rule addresses the challenge of spike decoding discussed in Chapter 1 while promoting energy efficient sparse representations. In Sections 3.1, 3.2, 3.3, 3.4 and 3.5, previously published in the proceedings of the Signal Processing and Wireless Communications conference [103], we develop the problem setting and algorithm and present simulation results that demonstrate the successful application and energy efficiency of our method and explore the trade-offs between learning efficiency and spike frequency.

## 3.1 Related Work

Artificial neural networks (ANNs) are used as parameterized non-linear models that serve as inductive bias for a large number of machine learning tasks, including notable applications of reinforcement learning (RL) to control problems [50]. Reinforcement learning using ANNs has been broadly applied in robotics control problems [74, 14] and various optimization problems for Internet-of-Things (IoT) mobile applications [23, 37] in which the satisfaction of energy constraints is an important issue (see, e.g., [21]). Due to their lower energy consumption when implemented on specialized

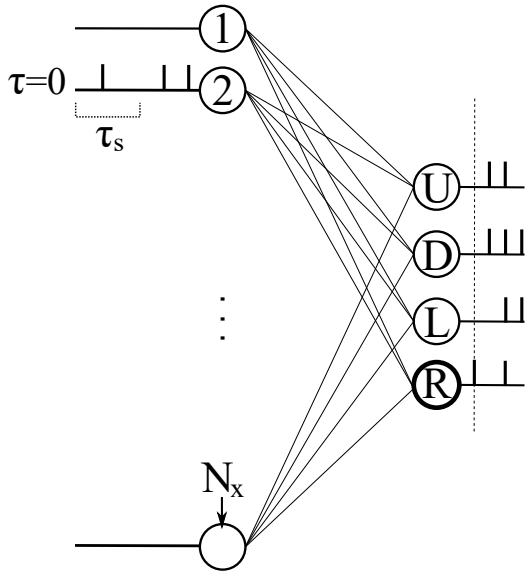


**Figure 3.1** SNN first-to-spike policy with action selected (R in the illustration) among Up, Down, Left, and Right marked with a bold line and decision time marked with a dashed vertical line

hardware, SNNs are considered to be important candidates as co-processors to be implemented in such resource constrained settings (see, e.g., [22, 14]).

Prior work on reinforcement learning using SNNs has by and large adopted deterministic SNN models to define action-value function approximators. This is typically done by leveraging *rate decoding* and either *rate encoding* [140, 88], or time encoding [13]. Under rate encoding and decoding, the spiking rates of input and output neurons represent the information being processed and produced, respectively, by the SNN. A standard approach, to be considered here as baseline, is to train an ANN offline and to then convert the resulting policy into a deterministic SNN with the aim of ensuring that the output spiking rates are close to the numerical output values of the trained ANN [27, 88]. There is also significant work in the theoretical neuroscience literature concerning the definition of reward based biologically plausible online learning rules [36, 112, 130, 58].

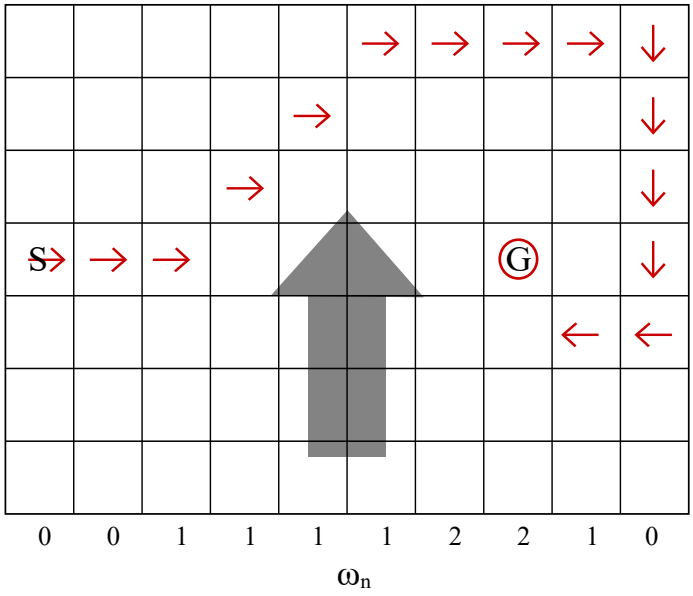


**Figure 3.2** An example of a realization of an action sequence in a windy grid-world problem. The bottom axis indicates the "wind" level per column, $\omega_n$, which causes the agent to be displaced by the given number of grid spaces when it moves into positions in that column.

In all of the reviewed studies, exploration is made possible by a range of mechanisms such as $\epsilon$-greedy in [112] and stochasticity introduced at the synaptic level [88, 130], requiring the addition of some external source of randomness. As a related work, reference [140] discusses the addition of noise to a deterministic SNN model to induce exploration of the state space from a hardware perspective. In contrast, in this study, the use of probabilistic SNN policies are investigated. The first to spike decision criterion detailed in Section 2.2.1 is applied to naturally enable exploration thanks to the inherent randomness of the decision. This makes it possible for the agent to learn while acting in an on-policy fashion. A gradient-based updating rule is derived that leverages the analytical tractability of the first-to-spike decision criterion under the GLM model.

**Algorithm 1** Policy Gradient Rule for First-to-Spike (FtS) SNNs

**Input:** randomly initialized parameter $\theta$, learning rate $\eta_i,\ i = 1, 2, ...$
  $i = 1$
  **repeat**
    **while** $S_t \neq S^{\mathrm{G}}$ **do**
      encode $S_t$ in spike domain
      run SNN and set $A_t \leftarrow$ index of FtS neuron
      observe next state and reward $S_{t+1}, R_{t+1}$
    **end while**
    $V_{t^{\mathrm{G}}+1} = 0$
    **for all** t=$t^{\mathrm{G}} : -1 : 1$ **do**
      $V_t = R_{t+1} + \lambda V_{t+1}$
      $\theta \leftarrow \theta + \eta_i \nabla_\theta \log \pi(A_t|S_t, \theta) V_t$
    **end for**
    $i \leftarrow i + 1$
  **until** convergence

### 3.2 Problem Definition

We consider a standard RL single-agent problem formulated on the basis of discrete-time Markov Decision Processes (MDPs). Accordingly, at every time-step $t = 1, 2, ...$, the RL agent takes an action $A_t$ from a finite set of options based on its knowledge of the current environment state $S_t$ with the aim of maximizing a long-term performance criterion. The agent's policy $\pi(A|S, \phi)$ is a parameterized probabilistic mapping from the state space to the action space, where $\phi$ is the vector of trainable parameters. After the agent takes an action $A_t$, the environment transitions to a new state $S_{t+1}$ which is observed by the agent along with an associated numeric reward signal $R_{t+1}$, where both $S_{t+1}$ and $R_{t+1}$ are generally random functions of the state $S_t$ and action $A_t$ with unknown conditional probability distributions.

An episode, starting at time $t = 0$ in some state $S_0$, ends at time $t^{\mathrm{G}}$, when the agent reaches a goal state $S^{\mathrm{G}}$. The performance of the agent's policy $\pi$ is measured by the long-term discounted average reward

$$V_\pi(S_0) = \sum_{t=0}^{\infty} \lambda^t \mathrm{E}_\pi[R_t], \tag{3.1}$$

where $0 < \lambda < 1$ is a discount factor. The reward $R_t$ is assumed to be zero for all times $t > t^{\mathrm{G}}$. With a proper definition of the reward signal $R_t$, this formulation ensures that the agent is incentivized to reach the terminal state in as few time-steps as possible.

While the approach proposed in this work can apply to arbitrary RL problems with discrete finite action space, we will focus on a standard windy grid-world environment [126]. Accordingly, as seen in Figure 1(b), the state space is an $M \times N$ grid of positions and the action space is the set of allowed horizontal and vertical single-position moves, i.e., Up, Down, Left, or Right. The start state $S_0$ and the terminal state $S^{\mathrm{G}}$ are fixed but unknown to the agent. Each column $n = 1, .., N$ in the grid is subject to some unknown degree of 'wind', which pushes the agent upward by $\omega_n$ spaces when it moves from a location in that column. The reward signal is defined as $R_{t+1} > 0$ if $S_{t+1} = S^{\mathrm{G}}$ and, otherwise, we have $R_{t+1} = 0$.

In order to model the policy $\pi(A|S, \phi)$, as we will detail in the next sections, we adopt the probabilistic SNN model described in detail in Chapter 2. We consider a two layer SNN architecture which includes a set of read out neurons $i \in \mathcal{R}$ with $N_s$ pre-synaptic connections each. The spiking neurons operate over discrete time $\tau = 1, ..., T$, updating the membrane potential at each time instant $\tau$ according to

$$u_{i,\tau} = \sum_{j=1}^{N_s} \alpha_{j,i} * \vec{s}_{j,\tau-1} + \gamma_j, \tag{3.2}$$

in which synaptic filter $\alpha_{j,i}$, defined by $K_a$ raised cosine basis functions [98] and synaptic weights $w^{\alpha}_{j,i}$, is applied to $\vec{s}_{j,\tau-1}$ the vector of $\tau_w$ previous pre-synaptic signals as detailed in Section 2.1.1. This expression is a variation of Equation (2.1) in which the post-synaptic feedback term has been excluded due to its irrelevance in neurons whose outputs are interpreted under the first to spike rule (see Section 2.2.1 for a more detailed explanation).

### 3.3 Policy-Gradient Learning Using First-to-Spike SNN Rule

In this section, we propose an on-policy learning algorithm for RL that uses a first-to-spike SNN as a stochastic random policy. Although the approach can be generalized, we focus here on the fully connected two-layer SNN shown in Figure 3.1. In the SNN, the first layer of $N_x$ neurons encodes the current state of the agent $S_t$, as detailed below, while there is one output GLM neuron for every possible action $A_t$ of the agent, with $N_s = N_x$ inputs each. For example, in the grid world of Figure 3.2, there are four output neurons. The policy $\pi(A|S, \phi)$ is parameterized by the vector $\phi$ of synaptic weights $\{w^{\alpha}_{j,i}\}$ and biases $\{\gamma_i\}$ for all the output neurons as defined in Equation (3.2). We now describe encoding, decoding, and learning rule.

**State Encoding.** A position $S_t$ is encoded into $N_x$ spike trains, i.e., binary sequences, with duration $T$ samples, each of which is assigned to one of the neurons in the input layer of the SNN. We emphasize that the time duration $T$ is internal to the operation of the SNN, and the agent remains at time-step $t$ while waiting for the outcome of the SNN. In order to explore the trade-off between encoding complexity and accuracy, we partition the grid into $N_x$ sections, or windows, each of size $W \times W$. Each section is encoded by one input neuron, so that increasing $W$ yields a smaller SNN at the cost of a reduced resolution of state encoding. Each position $S_t$ on the grid can be described in terms of the index $s(S_t) \in \{1, ..., N_x\}$ of the section it belongs to, and the index $w(S_t) \in \{1, ..., W^2\}$ indicating the location within the section using left-to-right and top-to-bottom ordering. Accordingly, using rate encoding, the input to the $i$th neuron is an i.i.d. spike train with probability of a spike given by

$$p_i = \begin{cases} p_{\min} + \left(\frac{p_{\max} - p_{\min}}{W^2 - 1}\right)(w(S_t) - 1), & \text{if } i = s(S_t) \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

for given parameters $p_{\min},\ p_{\max} \in [0, 1]$ and $p_{\max} \geq p_{\min}$.

**Decoding.** We adopt the first-to-spike decision protocol, so that the output of the SNN directly acts as a stochastic policy, inherently sampling from the distribution $\pi(A|S,\phi)$ induced by the first-to-spike rule which indicates that the action associated with the neuron that spikes first is taken by the agent. If no output neuron spikes during the input presentation time $T$, no action is taken, while if multiple output neurons spike concurrently, an action is chosen from among them at random. Given the synaptic weights and biases in vector $\phi$, the probability that the $j$th output neuron spikes first, and thus the probability that the network chooses action $A = j$, is given by the first-to-spike decision probability defined in Equation (2.4).

**Policy-gradient learning.** After an episode is generated by following the first-to-spike policy, the parameters $\theta$ are updated using the policy gradient method [126]. The gradient of the objective function (Equation 3.1) equals

$$\nabla_\theta V_\pi(S_0) = \mathrm{E}_\pi[V_t \nabla_\theta \log \pi(A_t|S_t,\theta)], \tag{3.4}$$

where $V_t = \sum_{t'=t}^{\infty} \lambda^{t'} R_{t'}$ is the discounted return from the current time-step until the end of the episode and the expectation is taken with respect to the distribution of states and actions under policy $\pi$ (see [126], Chapter 13). We are left with the gradient of the log likelihood of the action variable under the SNN first to spike policy which is equivalent to the log likelihood that the neuron corresponding to a given action is the first to spike. The gradient of the log likelihood of a first to spike decision is given by the expressions in Equations (2.5) and (2.6) as detailed in Section 2.2.1. The first-to-spike policy gradient algorithm is summarized in Algorithm 1, where the gradient in Equation (3.4) is approximated using Monte-Carlo sampling in each episode (see [126], Chapter 5).

## 3.4 Baseline SNN Solution

**Training of the ANN and Conversion into an IF SNN.** A two-layer ANN with four ReLU output units is trained by using the SARSA learning rule with $\epsilon$-greedy action selection in order to approximate the action-value function of the optimal policy [126]. The input to each neuron $i$ in the first layer of the ANN during training is given by the probability value, or spiking rate, $p_i$ defined in Equation (3.3), which encodes the environment state. Each output neuron of the ANN encodes the estimated value, i.e., the estimated long-term discounted average reward, of one of the four actions for the given input state and the action with the maximum value is chosen (under $\epsilon$-greedy action choices) with probability $\epsilon$. The ANN can then be directly converted into a two-layer IF SNN with the same architecture using the state-of-the-art methods proposed in [27], to which we refer for details. The converted SNN is used by means of rate decoding: the number of spikes output by each neuron in the second layer is used as a measure of the value of the corresponding action. We emphasize that, unlike in the proposed solution, the resulting (deterministic) IF SNN does not provide a random policy but rather a deterministic action-value function approximator.

## 3.5 Results and Discussion

In this section, we provide numerical results for the grid world example described in Section 3.2 with $M = 7$, $N = 10$, $S_0$ and $S^{\mathrm{G}}$ at positions (4,1) and (4,8) on the grid respectively, 'wind' level per columns defined by the values $\omega_n$ indicated in Figure 3.2. The number of cosine basis functions, $K_a$, included in filter $\alpha$ is equivalent to the length of the filter, $\tau_w$, for all simulations. Throughout, we set $p_{\min} = 0.5$ and $p_{\max} = 1$ for encoding in the spike domain and a learning schedule, $\eta_i = (\eta_{i-1})/(1 - k(i-1))$ with $\eta_0 = 10^{-2}$. Training is done for 25 epochs of 1000 iterations each, with 500 test episodes to evaluate the performance of the policy after each epoch. Hyper-parameters

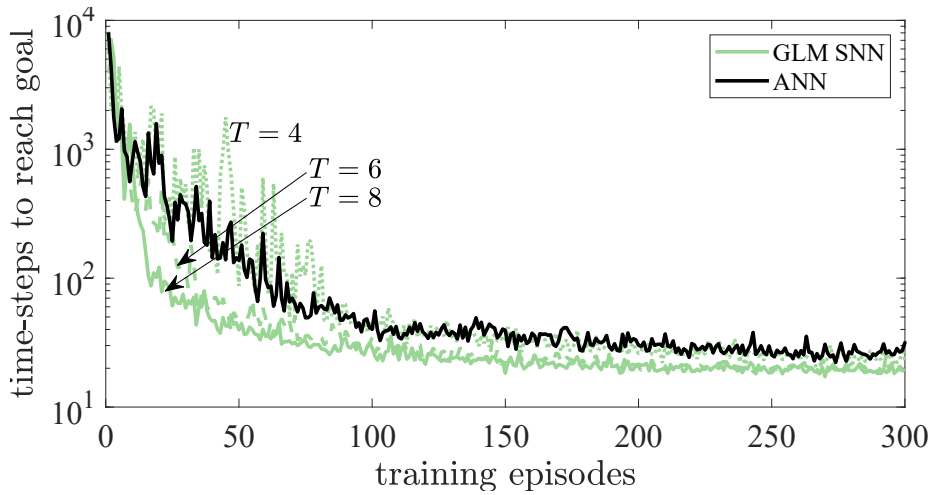


**Figure 3.3** Number of time-steps needed to reach the goal state as a function of the training episodes for the proposed GLM SNN and the reference ANN strategies.

for the SARSA algorithm to be used as described in the previous section are selected as recommended in [85, 70].

Apart from the IF SNN solution described in the previous section, we also use as reference, the performance of an ANN trained using the same policy gradient approach as in Algorithm 1 and having the same two-layer architecture as the proposed SNN. In particular, the input to each input neuron $i$ of the ANN is again given by the probability $p_i$ defined in Equation (3.3), while the output is given by a softmax non-linearity. The output hence encodes the probability of selecting each action. It is noted that, despite having the same architecture, the ANN has fewer parameters than the proposed first-to-spike SNN: while the SNN has $K_a$ parameters for each synapse given its capability to carry out temporal processing, the ANN has conventionally a single synaptic weight per synapse. This reference method is labeled as "ANN" in the figures.

We start by considering the convergence of the learning process along the training episodes in terms of number of time-steps to reach the goal state. To this end, in Figure 3.3, we plot the performance, averaged over the 25 training epochs, of the first-to-spike SNN policy with different values of input presentation duration $T$ and

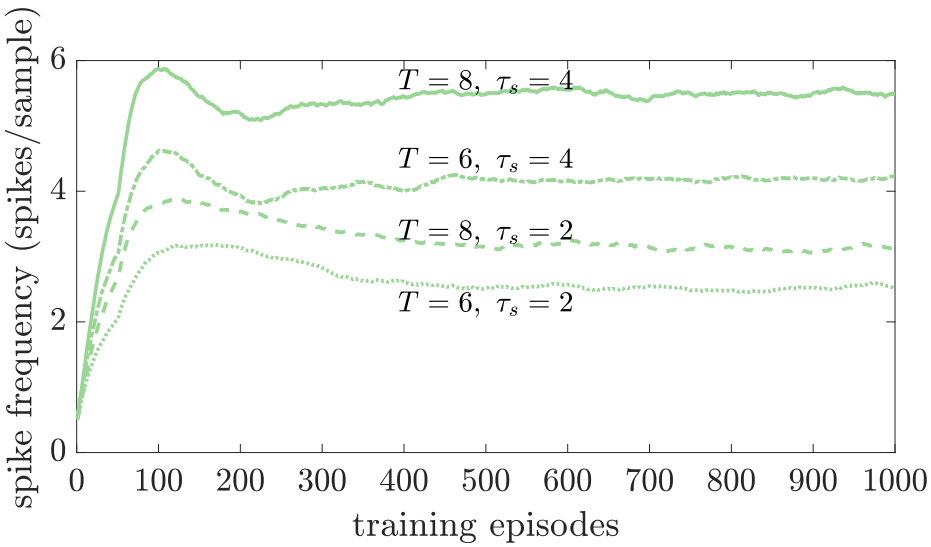


**Figure 3.4** Average spike frequency over the training episodes for the GLM SNN policy.

GLM parameters $K_a = \tau_w = 4$, as well as that of the reference ANN introduced above, both using encoding window size $W = 1$, and hence $N_x = 70$ input neurons. We do not show the performance of the IF SNN since this solution carries out offline learning (see Section 3.4). The probabilistic SNN policy is seen to learn more quickly how to reach the goal point in fewer time-steps as $T$ is increased. This improvement stems from the proportional increase in the number of input spikes that can be processed by the SNN, enhancing the accuracy of input encoding. It is also interesting to observe that the ANN strategy is outperformed by the first-to-spike SNN policy. As discussed, this is due to the capability of the SNN to learn synaptic kernels via its additional weights.

We further investigate the behavior of the first-to-spike SNN during training in Figure 3.4, which plots the spike frequency as a function of the training episodes. The initially very low spike frequency can be interpreted as an exploration phase, where the network makes mostly random action choices by largely neglecting the input spikes. The spike frequency then increases as the SNN learns while exploring effective actions dictated by the first-to-spike rule. Finally, after the first one hundred

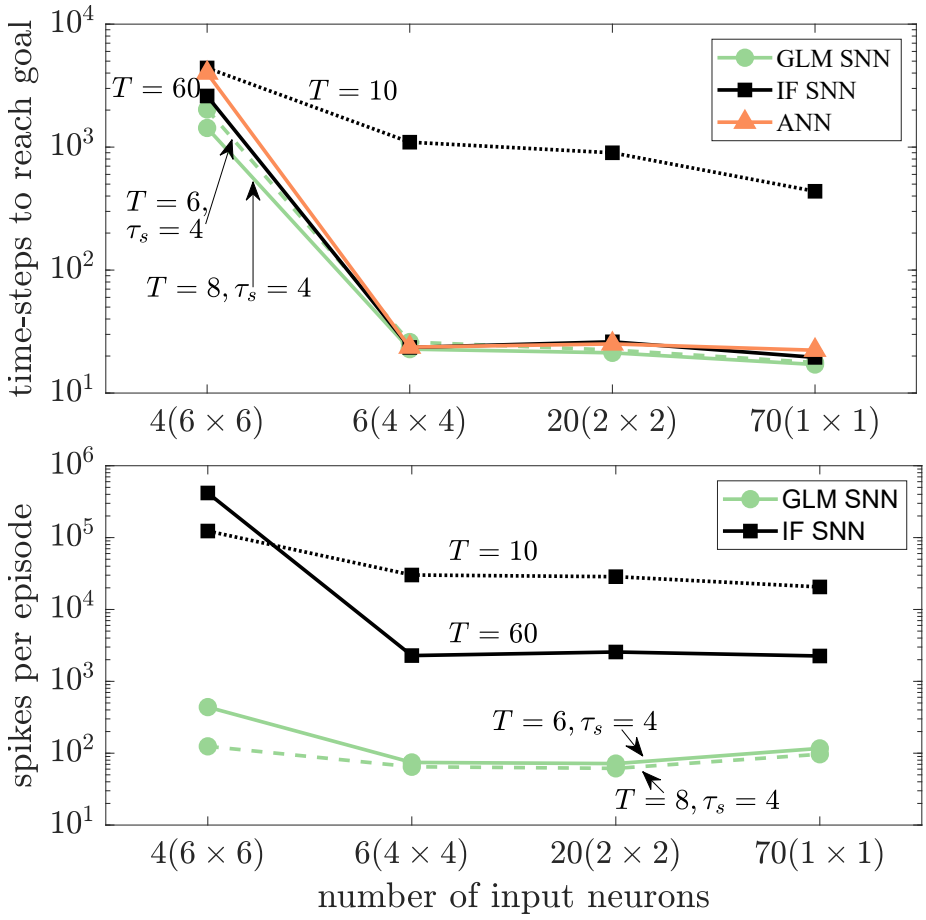


**Figure 3.5** Average number of time-steps to reach goal (top) and average number of total spikes per sample (bottom) in the test episodes as a function of the number of input neurons $N_x$ (also indicated is the size of the $W \times W$ encoding window) for GLM SNN and IF SNN.

episodes, the SNN learns to exploit optimal actions, hence reducing the number of observed spikes necessary to fire the neuron corresponding to the optimized action.

We now turn to the performance evaluated after training. Here we consider also the performance of the conventional IF SNN trained offline as described in Section 3.4. We first analyze the impact of using coarser state encodings as defined by the encoding window size $W$. Considering only test episodes, Figure 3.5 plots the number of time-steps to reach the goal (top) and the total number of spikes per episode across the network (bottom), as a function of the number of input neurons, or equivalently of $W$. For all schemes, it is seen that, as long as the window size is no larger than

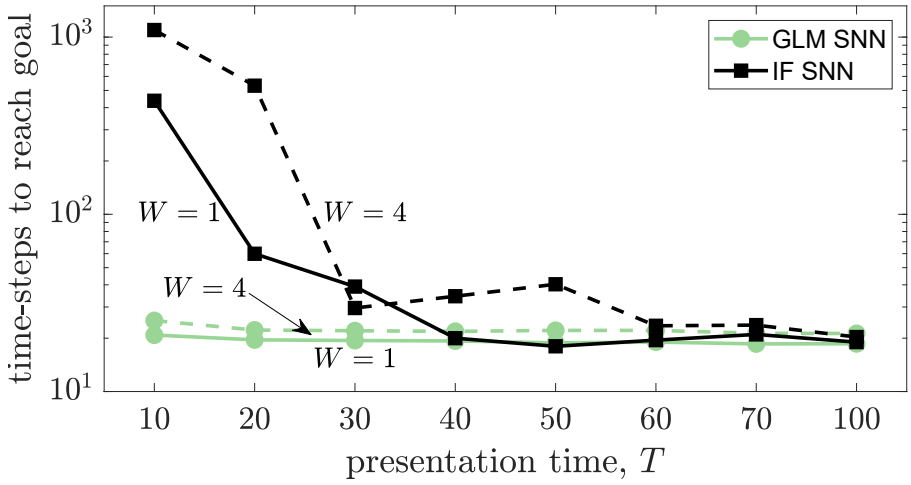


**Figure 3.6** Average number of time-steps to reach goal versus the presentation time $T$ for GLM SNN and IF SNN ($\tau_w = 6, K_a = 1$).

$W = 4$ and $T$ is large enough for the SNN-based strategies, no significant increase of time-steps to reach the goal is incurred. Importantly, the IF SNN is observed to require $10\times$ the presentation time and more than $10\times$ the number of spikes per episode of the first-to-spike SNN to achieve the same performance.

The test performance comparison between first-to-spike SNN and IF SNN is further studied in Figure 3.6, which varies the presentation time $T$. In order to discount the advantages of the first-to-spike SNN due to its larger number of synaptic parameters, we set here $K_a = 1$, thus reducing the number of synaptic parameters to 1 as for the IF SNN. Figure 3.6 shows that the gains of the proposed policy are to be largely ascribed to its decision rule learned based on first-to-spike decoding. In contrast, the IF SNN uses conventional rate decoding, which requires a larger value of $T$ in order to obtain a sufficiently good estimate of the value of each state via the spiking rates of the corresponding output neurons.

## 3.6 Conclusion

In this chapter we have proposed a policy gradient-based online learning strategy for a first-to-spike stochastic SNN. As compared to a conventional approach based on

offline learning and conversion of a second generation ANN to an integrate-and-fire SNN with rate decoding, the proposed approach was seen to yield a reduction in presentation time and number of spikes by more than $10\times$ in a standard windy grid world example. Thanks to the larger number of trainable parameters associated with each synapse, which enables optimization of the synaptic kernels, performance gains were also observed with respect to a conventional ANN with the same architecture that was trained online using policy gradient.

# CHAPTER 4

# ONLINE-WITHIN-ONLINE META LEARNING

The previous chapter highlighted the power efficiency of an SNN trained using the first to spike decision rule. In this chapter we turn to meta-learning as a method by which to increase the learning efficiency of the SNN by decreasing the number of training iterations required to achieve profficiency on a given task. This feature is especially relevant to SNNs because of its application to the online adaptation of a learned model to individualized data on personal, resource constrained, edge devices. While the stochastic SNN adopted in Chapter 3 included only neurons with observed spiking outputs, in this problem, it is expanded to include hidden neurons with latent output variables and the variational learning methodology described in Section 2.2.3 is applied. These concepts are developed in the following sections that are adapted from my previously published paper [102].

## 4.1 Introduction

*Context and motivation.* The standard "train offline-then-deploy" approach underlies most applications of machine learning for tasks as different as facial recognition, natural language processing, and health monitoring. This framework yields rigid solutions that can produce inaccurate predictions and decisions when the data encountered after deployment presents statistical differences with respect to the training data. As an example, consider a natural language processor run by a user with a specific speech impairment or accent. Since mobile machine learning solutions are used for highly personalized tasks, there is a need to fine tune the models to the unique features of individuals and local environments after deployment in order to avoid unfair and inefficient results across different adopters of the technology [65].

Meta-learning, or learning to learn, addresses this problem by eschewing the identification of a "universal" model to be fixed at deployment time, and focusing instead on determining an adaptation procedure that is able to adjust the model based on limited local data [7, 121, 54, 32]. This can be generally done in one of two ways – by meta-learning shared sub-models, e.g., feature extractors [131, 61, 92], or by meta-learning shared training procedures operating over local data. As an example of the latter, it is possible to meta-learn local iterative rules [81, 87], initializations [34, 93, 82], or learning rates [15].

*Online vs offline meta-learning.* In the classical formulation of meta-learning [7, 57], it is assumed that there exists a family of tasks with similarly distributed data. Tasks may, for instance, correspond to personalizations for different users. Meta-training data is obtained by sampling tasks and per-task data sets from a given, unknown, distribution. The meta-learned solution is then "meta-tested" on new tasks sampled from the same underlying distribution in order to evaluate its performance and adaptability. Meta-learning algorithms follow a *nested loop* structure: The outer loop updates the hyperparameters that define the shared sub-models or learning procedures, and the inner loop carries out per-task learning based on limited data.

We can distinguish different types of meta-learning problems depending on whether either loop is run using offline or online procedures. We can specifically have *online/offline-within-online/offline* settings, in which the former selection applies to the outer loop and the latter to the inner loop. An offline procedure is one that uses a static data set with data points fully accessible by the algorithms, while online (meta-)learning uses streaming data that is sequentially processed by the algorithm [26].

*Main contributions.* The general formulation of meta-learning is in line with research in neuroscience that has shown how biological brains can learn broader concepts on a slower timescale, allowing faster adaptation to specific activities or tasks

[59, 78]. In light of this observation, this chapter focuses on the development of an *online-within-online* meta-learning algorithm for biologically inspired SNNs, termed OWOML-SNN. Unlike prior work to be reviewed in teh following section, the derived rules do not require an offline pre-deployment stage and the use of backpropagation (BP). Rather, they rely on a *three-factor nested local learning rule* [38] that is derived by following a principled approach grounded in the use of probabilistic Generalized Linear Models (GLMs) for the spiking neurons and variational inference [51, 53]. The online within-task updates encompass a pre-synaptic term that implements an eligibility trace on recent activity on the pre-synaptic neuron; a post-synaptic term that can be interpreted based on the principles of *predictive coding* [83] as an error between desired or realized post-synaptic output and the probabilistic model's prediction; and a global feedback signal. The online meta-update follows the form of the first-order meta-gradient introduced in [93].

## 4.2 Related Work

In the most common meta-learning scenario, hyperparameters are updated in the outer loop in an offline manner via a meta-gradient over data from batches of tasks selected from a data distribution [42, 92, 34]. Batch-within-batch algorithms implement within-task learning in an offline manner as well [34, 93, 42, 81], while batch-within-online strategies use within-task data in a streaming fashion [92, 87]. The online-within-online setting assumed here is most similar to that adopted in [35, 26], where an online stream of task datasets is assumed in the outer loop. In our case, as in [26], the within-task adaptation assumes an online stream of data, while [35] implements batch learning in the inner loop via standard BP. Aside from the similarity in formulation, reference [26] approaches the solution using a static deterministic linear model under a convex loss function while we study a dynamic (spiking) probabilistic model with a non-convex loss function.

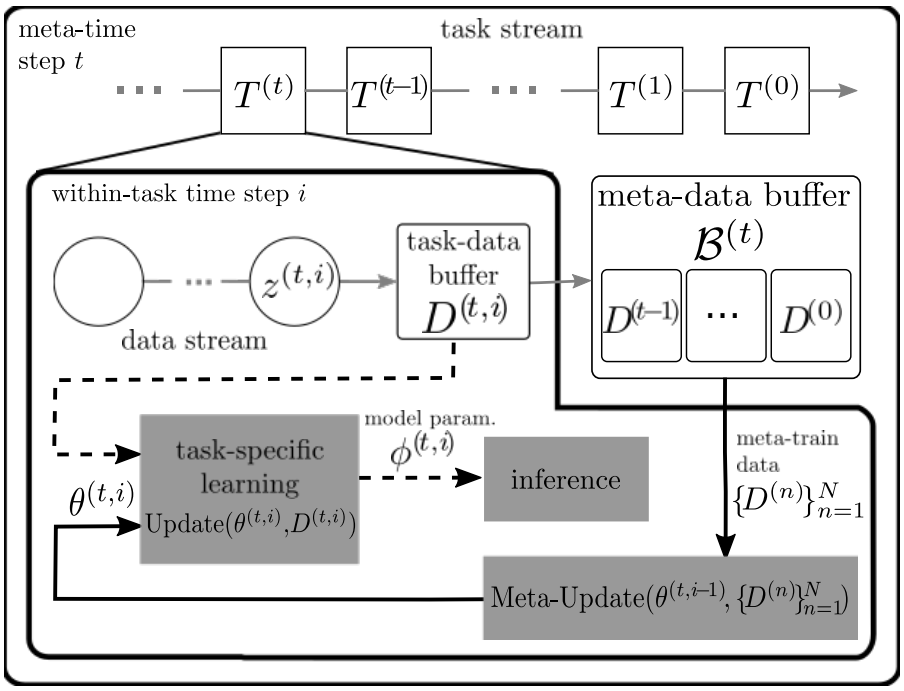


**Figure 4.1** Online-within-online meta-learning: Tasks $T^{(t)}$ are drawn from family $\mathcal{F}$ and presented sequentially to the meta-learner over timescale $t$ (denoted in the top left corner). Within-task data are also observed sequentially (bold inset box), with a new batch $z^{(t,i)}$ added to the task-data buffer $D^{(t,i)}$ at each within task time $(t, i)$. After all within-task data has been processed, the task-data buffer is added to the meta-data buffer $\mathcal{B}^{(t)}$. At each time-step $(t, i)$ the meta-learner seeks to improve online inference by learning task-specific parameter $\phi^{(t,i)}$ using the updated task-data buffer and hyper-parameter $\theta^{(t,i)}$ as the initialization (dashed arrows). Concurrently, the meta-learner makes a meta-update to the hyperparameter, yielding the next iterate $\theta^{(t,i+1)}$. As part of the meta-update, data from $N$ different previously seen tasks are sampled from the buffer $\mathcal{B}(t)$ and task-specific parameters for $N$ parallel models are learned starting from the hyper-parameter initialization $\theta^{(t,i)}$ (solid arrows).

All the works summarized so far assume standard ANN models. SNNs are also being explored for meta-learning due to their capability to implement within-task learning via online, local rules [125, 108, 124, 15, 9]. In all cases, meta-training is implemented offline or pre-deployment. The SNN models adopted in prior work require the use of surrogate gradients to approximate BP, and they rely on the use of the deterministic leaky integrate and fire (LIF) neuron model. In contrast we adopt the probabilistic Generalized Linear Model (GLM) neuron model, allowing a direct derivation of local rules based on maximum likelihood [51, 53].

### 4.3 Online-Within-Online Meta-Learning

In this section, we outline the general operation of online within-online meta-learning via meta-gradient descent. Meta-learning assumes the presence of a family $\mathcal{F}$ of tasks that share common statistical properties. Specifically, it assumes that a common hyper-parameter $\theta$ can be identified that yields efficient learning when applied separately for each task in $\mathcal{F}$. Following the current dominant approach [35, 93], we will take hyper-parameter $\theta$ to represent the initialization to be used for the within-task training iterative procedure.

In the considered online-within-online formulation adapted from [35], the meta-learner seeks to improve its *online inference* capabilities over a series of tasks drawn from family $\mathcal{F}$. For each new task, it aims to quickly learn a task-specific model using streaming within-task data. To this end, the meta-learner runs an underlying meta-learning process to update the hyper-parameter $\theta$ by using data observed from previous tasks in the series. The hyper-parameter $\theta$ is then used as a within-task model initialization that enables efficient within-task training for the new task.

To support *online inference and meta-learning*, two data buffers are maintained. The *task-data buffer* collects streaming within-task data used for within-task learning, While the *meta-data buffer* holds data from a number of previous tasks to be used by the meta-training process. As illustrated in Figure 4.1, a stream of data sets $\mathcal{D}^{(t)}$, each corresponding to a task $T^{(t)} \in \mathcal{F}$, is presented to the meta-learner sequentially at $t = 1, 2, ....$ Within each *meta-time step* $t$, samples from data set $\mathcal{D}^{(t)}$ are also presented sequentially, so that at each *within-task time step* $i$, a batch $z^{(t,i)} = \{(x^j, y^j)\}_{j=1}^{B} \subseteq \mathcal{D}^{(t)}$ of $B$ training examples for task $T^{(t)}$ is observed, and added to the task-data buffer as $D^{(t,i)} = D^{(t,i-1)} \cup z^{(t,i)}$ with $D^{(t,0)} = \emptyset$. Once all within-task data for task $T^{(t)}$ has been processed, the final task-data buffer $D^{(t,i)}$ is added to a meta-data buffer $\mathcal{B}^{(t)}$.

The within-task training and meta-training processes take place concurrently at each time $(t, i)$. As a new batch of within-task data is observed for the current task $T^{(t)}$, the meta-learner uses it, along with the entire current task-data buffer $D^{(t,i)}$, to learn a better task-specific model parameter $\phi^{(t,i)}$ and thus improve inference on a held-out test data set. The task-specific parameter is initialized with the current hyper-parameter $\theta^{(t,i)}$ and is updated via an iterative within task training process $\phi^{(t,i)} = \text{Update}(\theta^{(t,i)}, D^{(t,i)})$. Concurrently, the meta-learner improves the hyper-parameter initialization for the next round of within-task training by making a single gradient update to $\theta$. This update can be written as $\theta^{(t,i+1)} \leftarrow \theta^{(t,i)} + \mu \nabla_\theta F(\theta^{(t,i)}, \mathcal{B}^{(t)})$ for some meta-learning rate $\mu \geq 0$, where $F$ is the meta-learning objective function. The meta-learning objective evaluates the performance of the initialization $\theta^{(t,i)}$ on data from previous tasks stored in the meta-data buffer $\mathcal{B}^{(t)}$.

Specifically, in order to evaluate the meta-learning objective function $F$, task-specific parameters for a number of previous tasks need to be learned. To this end, $N$ tasks $T^{(n)},\ n = 1, ..., N$, are drawn from the meta-data buffer $\mathcal{B}^{(t)}$ and a small data-set, $D^{(n)}$, is drawn as a subset of the stored data for each task. The within-task iterative training process is applied to learn task-specific parameters $\phi^{(n)} = \text{Update}(\theta^{(t,i)}, D^{(n)})$ using $N$ parallel models, each initialized with the hyper-parameter $\theta^{(t,i)}$.

The update function $\text{Update}(\theta, D)$ addresses the problem of maximizing the likelihood of data over model parameters $\phi$. Specifically, the data set $D^{(n)}$ is split into distinct data sets $D^{(n)}_{meta}$ for the meta optimization and $D^{(n)}_{task}$ for the within-task maximization [34], and the update function tackles the problem $\max_\phi \log p(D^{(n)} \mid \phi)$ via stochastic gradient descent (SGD) starting from initialization $\theta$. The meta-objective is then defined as

$$F\left(\theta, \mathcal{B}^{(t)}\right) = \sum_{n=1}^{N} \log p\left(D^{(n)}_{meta} \middle| \text{Update}\left(\theta, D^{(n)}_{task}\right)\right). \tag{4.1}$$

The meta-gradient $\nabla_\theta F(\theta, \mathcal{B}^{(t)})$ requires the computation of the second order gradient of the training losses used in the task-specific learning function $\text{Update}(\theta, D_{task}^{(n)})$ [34]. In this work, we make use of the first-order REPTILE approximation for the gradient

$$\nabla_\theta \log p\left(D_{meta}^{(n)} \middle| \phi^{(n)}\right) \approx \theta^{(t,i)} - \phi^{(n)}, \tag{4.2}$$

which has been shown to have properties similar to the true gradient in a number of benchmark tasks [93]. This yields the meta-update function

$$\text{Meta-Update}(\theta^{(t,i)}, \{D^{(n)}\}_{n=1}^{N}) = \theta^{(t,i)} + \mu \sum_{n=1}^{N} \left(\theta^{(t,i)} - \text{Update}(\theta^{(t,i)}, D_{task}^{(n)})\right), \tag{4.3}$$

where $\mu \geq 0$ is the meta-learning rate.

## 4.4 SNN Model

We adopt a recurrent SNN based on the GLM spiking neuron described in Section 2.1.1 for optimization via the online-within-online meta-learning framework described above. The network includes a set of visible neurons $\mathcal{V}$ whose outputs are specified by data during training, as well as a set of hidden neurons $\mathcal{H}$ whose outputs are auxiliary to the operation of the visible neurons, as discussed in Section 2.2.3. Arbitrary connections are defined between neurons within each set and across sets, such that the post-synaptic spike of any neuron may be a pre-synaptic input to any other neuron in the network (see Figure 4.2).

## 4.5 Per-task Learning

We assume the data in the task-data buffer to be in the form of spike sequences $(x_{\leq \mathcal{T}}, y_{\leq \mathcal{T}})$ of length $\mathcal{T}$. They may be natively neuromorphic signals or natural signals that are converted to the spike domain via the application of an encoder. The per-task training function, $\text{Update}(\phi, D)$ concatenates all the examples $(x_{\leq \mathcal{T}}, y_{\leq \mathcal{T}}) \in$

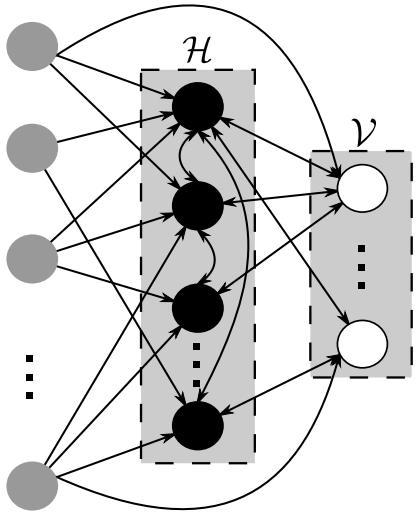


**Figure 4.2** An example of an SNN with an arbitrary topology. Black circles are the hidden neurons, in set $\mathcal{H}$, and white circles are the visible neurons in set $\mathcal{V}$, while gray circles represent exogenous inputs. Synaptic links are shown as directed arrows, with the post-synaptic spikes of the source neuron being integrated as inputs to the destination neuron. A bi-directional arrow represents two individual connections between the two neurons concerned, one in each direction.

$D^{(t,i)}$ along the time dimension to obtain a single stream of data $(x_{\leq S}, y_{\leq S})$ with $S = M\mathcal{T}$. The function processes this data over time with the goal of addressing the maximum likelihood problem $\max_\theta \mathcal{L}_{\upsilon_{\leq S}}(\phi) = \log p_\phi(\upsilon_{\leq S}, h_{\leq S})$ with initialization $\theta$. We approach this problem using the variational learning method reviewed in Section 2.2.3, by maximizing the ELBO given in Equation (2.9) with the variational distribution $q_\theta(h_{\leq T})$ modeled by the distribution over the latent spikes $p_\phi(h_{\leq S})$ defined by the hidden neurons [51].

The SNN model parameter vector $\phi$, initialized by the hyperparameter $\theta$, is iteratively updated according to the gradients detailed in Section 2.2.3, namely Equations (2.13a) and (2.13b), along with the local gradients (Equation 2.8) calculated at every time $\tau$. The learning signal $\ell_{\phi^V}$ is given by the log-likelihood of the observed spikes estimated via Monte Carlo sampling of the hidden spikes $[h_{i,\tau} \sim p_\phi(h_{i,\tau} \mid u_{i,\tau})]_{i\in\mathcal{H}}$ at every discrete time-instant $\tau$ which allows the computation of the new membrane potentials $[u_{i,\tau+1}]_{i\in\mathcal{V},\mathcal{H}}$ according to Equation (2.1). The

derivatives are accumulated for $\Delta s$ discrete time-steps and added to an eligibility trace that is maintained as an exponentially decaying average of previous gradients to update the SNN model parameters. The overall operation of the Update$(\theta, D)$ function, including a sparsity-inducing regularizer [51], is summarized in Algorithm 2.

### 4.6 SNN Meta-Learning

We now describe OWOML-SNN, an online-within-online meta-learning algorithm for probabilistic SNNs that builds on the framework described in Section 4.3. The overall algorithm is described in Algorithm 3, and is detailed next. The meta-learner is defined as an SNN model whose weights define the hyperparameters $\theta^{(t,i)}$. At each within-task time-step $(t, i)$, $N + 1$ SNN models are instantiated with initial weight given by $\theta^{(t,i)}$. One SNN is used to carry out inference on the current task $T^{(t)}$, while the remaining $N$ SNN models are used to enable the meta-update.

**Algorithm 2** Maximum Likelihood Optimization via SGD for Meta Learning Within Task Update, Update($\theta, D$)

**Input:** $\theta, D$, hyperparameters $\kappa, \Delta s, \eta$

initialize $\phi \leftarrow \theta$, $l_0 \leftarrow 0$, $e_{i,0} \leftarrow 0$
**for** $\tau = 1$ **to** $S$ **do**
    compute membrane potentials $u_{i,\tau}$ as in (Eq. 2.1)
    **for** $i \in \mathcal{H}$ **do**
        Monte Carlo sampling of $h_{i,\tau}$
    **end for**
    accumulate learning signal $l_\tau = l_{\tau-1} + \ell_\tau$ (Eq. 2.12)
    accumulate local gradients
$e_{i,\tau} = e_{i,\tau} + \nabla_{w_{j,i}^\alpha} \log p(\upsilon_{i,\tau} \mid u_{i,\tau})$ (Eq. 2.8)
    **if** $\tau$ is a multiple of $\Delta s$ **then**
        compute learning signal trace
$l = \kappa l + (1 - \kappa) l_\tau$ (4.4)
        **for** neurons $i \in \{\mathcal{V}, \mathcal{H}\}$ **do**
            compute local gradient traces
            update parameters $e_i = \kappa e_i + (1 - \kappa) e_{i,\tau}$ (4.5)

$$\phi_i \leftarrow \phi_i + \eta \begin{cases} e_i \text{ if } i \in \mathcal{V} \\ l e_i \text{ if } i \in \mathcal{H} \end{cases}$$

        **end for**
        reset $l_\tau \leftarrow 0$, $e_{i,\tau} \leftarrow 0$
    **end if**
**end for**
**return:** $\phi$

To elaborate, at every meta-time step $t$, a task $T^{(t)} \in \mathcal{F}$ is drawn, and the task-data buffer is initialized as $D^{(t,i)} = \emptyset$. Within-task data is added to the current task-data buffer at every within-task time step $i$ in batches of $B$ training examples $z^{(t,i)} = \{(x^j_{\leq \mathcal{T}}, y^j_{\leq \mathcal{T}})\}_{j=1}^B$. The inference SNN implements online learning for the current task via the update function Update($\theta^{(t,i)}, D^{(t,i)}$) using the hyperparameter initialization $\theta^{(t,i)}$ and data in the task-data buffer.

To enable the meta update, the mentioned $N$ SNN models are trained online using $N$ data-sets sampled from the meta-data buffer $\{D^{(n)}\}_{n=1}^N \in \mathcal{B}^{(t)}$. Each of the data-sets includes $M$ training examples that are a subset of the data-set of a previously seen task such that $D^{(n)} = \{(x^j_{\leq \mathcal{T}}, y^j_{\leq \mathcal{T}})\}_{j=1}^M$. The hyperparameter is updated via the

function Meta-Update $\left(\theta^{(t,i)}, \{D^{(n)}\}_{n=1}^{N}\right)$ which includes within-task training on the $N$ sampled data-sets.

The inference accuracy of the within-task parameter $\phi^{(t,i)}$ is tested on a held out test data set for the current task. Online meta-learning is successful if the inference SNN is able to obtain satisfactory accuracy levels by using fewer examples for online adaptation.

## 4.7 Experiments

In this section, we compare the performance of OWOML-SNN to conventional per-task training and joint training (a standard benchmark for meta-learning [142, 3, 34]). Under conventional training, the meta-update function in Algorithm 3 is disabled and the hyperparameter $\theta^{(t,0)}$ is randomly re-initialized for any new task $T^{(t)}$. Under joint training, the meta-training function Meta-Update$(\theta, \mathcal{B})$, is replaced by training across all tasks whose data is in buffer $\mathcal{B}^{(t)}$. This amounts to applying function Update$(\theta^{(t,i)}, D)$, where $D$ includes examples sampled from $\mathcal{B}^{(t)}$ as for the meta-update function.

**Algorithm 3** OWOML-SNN

**Input:** $\mathcal{B}^{(0)}$, hyperparameters $N, \mu$

**repeat**
  $t \leftarrow t+1$
  sample $T^{(t)} \in \mathcal{F}$
  initialize $D^{(t,0)} \leftarrow \emptyset$
  **while** data available for $T^{(t)}$ **do**
    $i \leftarrow i+1$
    $D^{(t,i)} \leftarrow D^{(t,i)} \cup z^{(t,i)}$
    $\phi^{(t,i)} \leftarrow \text{Update}\left(\theta^{(t,i)}, D^{(t,i)}\right)$
    inference using $\phi^{(t,i)}$
    sample meta-train data $\{D^{(n)}\}_{n=1}^{N} \in \mathcal{B}^{(t)}$
    $\theta^{(t,i+1)} \leftarrow \text{Meta-Update}\left(\theta^{(t,i)}, \{D^{(n)}\}_{n=1}^{N}\right)$
  **end while**
  $\mathcal{B} = \mathcal{B} \cup D^{(t,i)}$
**until** convergence

**function** Meta-Update($\theta$, $\{D^{(n)}\}_{n=1}^{N}$)
  **for** $n = 1$ **to** $N$ parallel models **do**
    within-task training $\phi^{(n)} \leftarrow \text{Update}(\theta, D^{(n)})$
  **end for**
  $\theta \leftarrow \theta + \mu \sum_{n=1}^{N} \left(\theta - \phi^{(n)}\right)$
  **return** $\theta$
**end function**

We first consider the family of omniglot 2-way classification tasks [66], which is often used to test the within-task generalization capabilities of meta-learners [34, 108, 131, 61]. We follow the more complex definition of the problem in which the two characters to be classified in each task may be from different alphabets. Each task dataset includes 14 examples from each class, while 6 examples from each class are reserved for the test dataset. The figures are downsized to $26 \times 26$ pixels and rate encoded to obtain examples of the form $(x_{\leq\mathcal{T}}, y_{\leq\mathcal{T}})$ where $y_{\leq\mathcal{T}}$ is a one hot encoded label and $\mathcal{T} = 80$.

We train a fully connected GLM-based SNN with 4 hidden neurons and 2 visible neurons with additional lateral connections between the hidden neurons and feedback connections from the visible neurons to the hidden neurons. After 15 meta-time

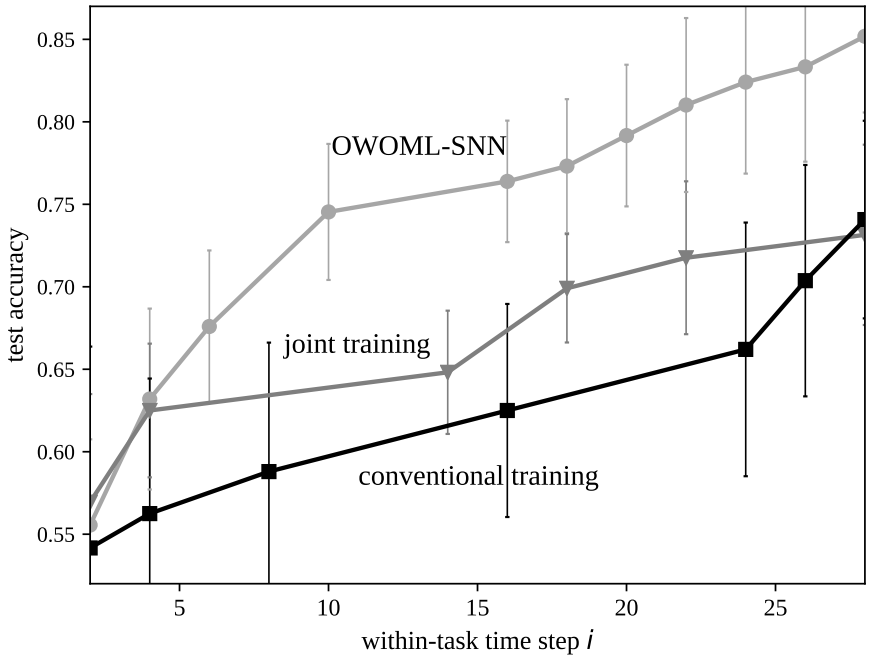


**Figure 4.3** Test accuracy of within-task training after $t = 15$ meta-time steps for the omniglot 2-way classification task family. Lines show an average over 6 new 2-digit classification tasks with half standard deviation error bars. 5-shot training is completed after $i = 10$ within-task time steps

steps, i.e., at $t = 15$, we test the hyperparameter initialization on the next 6 unseen tasks without further meta training updates. We set $M = 2, N = 5$, and $\Delta s = 5$. The results in Figure 4.3 show that after training on five examples from each class, OWOML-SNN provides an increase in accuracy over the baseline of conventional training that is about 18% larger than joint training. After further training, we observe that OWOML-SNN enables the SNN to achieve on average a 15% higher accuracy overall than the two benchmarks.

We now show that the fast adaptation capability of OWOML-SNN extends to the case of a continuous input stream with data encoded in the spike timing, by considering 2-way classification on the the MNIST-DVS data-set [110]. MNIST-DVS is a dataset of MNIST images captured by a neuromorphic DVS camera that generates localized events based on changes in individual pixels over time. The tasks are sampled from the set of permutations of the digits between 0 and 6 with 900

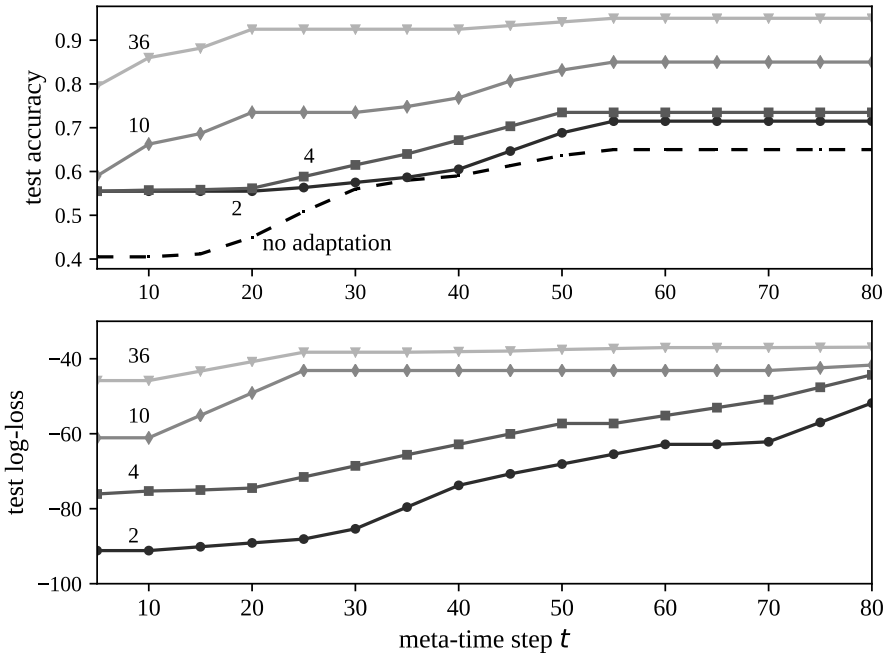


**Figure 4.4** Within-task test accuracy (top) and loss (bottom) on the current task $T^{(t)}$ over online-within-online meta-training for the MNIST-DVS 2-way classification task family after the full within-task dataset $D^{(t,i)}$ has been processed. The dashed line represents no within-task adaptation. Each solid line represents test results after within-task training on the number of training examples labeled. Due to variability in the hardness of the tasks, the lines show best accuracy seen so far, giving an envelope of the overall behavior.

examples from each class used for training and 100 held out for test. Each image is cropped to the $26 \times 26$ pixel size and the events are downsampled to create a sequence of length $\mathcal{T} = 50$ [118].

We train an SNN with the same recurrent architecture as described in the previous experiment with hyperparameters $M = 4, N = 5$, and $\Delta s = 20$. We examine the performance of the SNN instantiated for task-specific training on the current task $T^{(t)}$ as the hyperparameter initialization is improved over meta-training. We show results for online adaptation after the full dataset $D^{(t,i)}$ for the current task has been processed i.e., at the maximum value of within-task time step $i$. The results in Figure 4.4 confirm that OWOML-SNN enables fast online adaptation that continuously

improves as the number of meta-time steps $t$ increases and more meta-updates are applied, requiring fewer examples to achieve within-task accuracy targets.

## 4.8 Conclusion

The OWOML-SNN proposed is an online-within-online meta-learning algorithm that builds on online variational learning algorithm for probabilistic SNNs. We have demonstrated the performance benefits of OWOML-SNN when adapting to new tasks observed sequentially in an online fashion. OWOML-SNN is based on a first-order approximation of the meta-gradient, yielding a local meta-learning rule that follows the principles of predictive coding, with no reliance on back-propagation of gradients. This makes the approach a prime candidate for fast on-chip adaptation to new tasks, and in particular for tasks that involve streaming input data. While the first order approximation yields an efficient local update rule, it also generally reduces the capacity of adaptation to new tasks [93]. As future work, it would be interesting to investigate more accurate approximations of the meta-gradient that retain the property of locality, as well as convergence properties.

# CHAPTER 5

# SPIKING GENERATIVE ADVERSARIAL NETWORKS

In this chapter we move away from supervised learning algorithms for SNNs to address SNN learning without the definition of rigid target output patterns. We highlight the fact that this allows us to train an SNN to output signals with diverse spatial and temporal correlations. We adopt the adversarial learning framework which offers a guided learning process for the SNN that centers around the similarity of the distribution of the spiking outputs to a target distribution, as mediated by an auxiliary discriminative model. As a result of this, the SNNs considered here include only hidden neurons whose spiking outputs are in the latent variable space, in contrast to the previous studies in which some or all of the spiking neuron outputs are observed as defined by a target output sequence.

Learning efficiency and model adaptability are important considerations for adversarial learning, especially in the context of on chip learning which is our overarching goal in developing local learning algorithms for SNNs. Towards this end, we apply the meta-learning framework discussed in Chapter 4 to decrease the number of training iterations required to train an SNN using adversarial learning.

These topics are further developed in the rest of the chapter, reproduced from my paper [104].

## 5.1 Introduction

In existing SNN learning solutions, inputs and outputs are prescribed spike patterns, which may be obtained from a neuromorphic data set – e.g., from a Dynamic Vision Sensor (DVS) camera or a Dynamic Audio Sensor (DAS) recorder [109, 72] – or converted from natural signals using a fixed encoding strategies such as rate or temporal encoding [94]. This approach may lead to an overly rigid and narrow

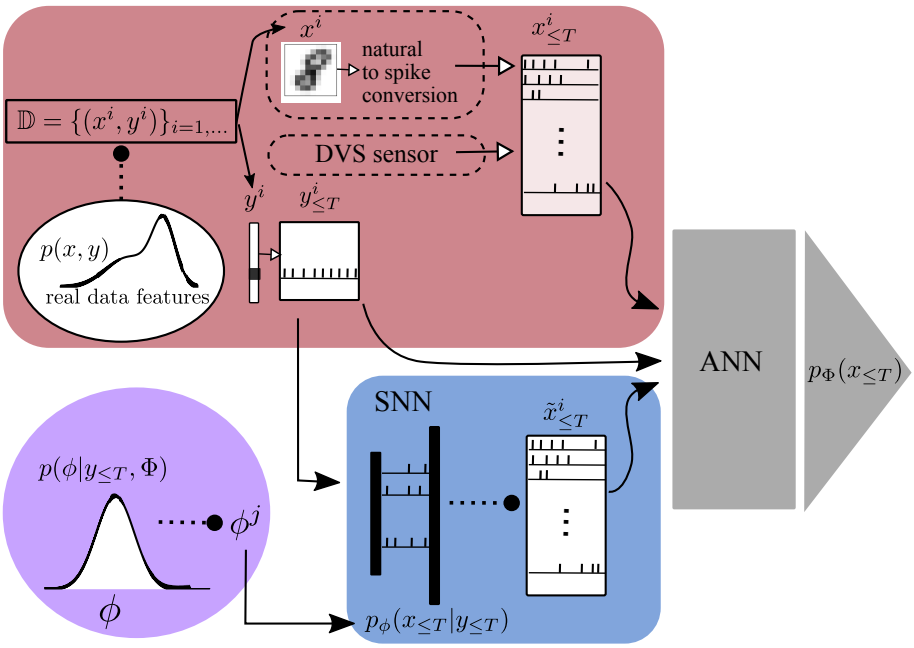


**Figure 5.1** Block diagram of the proposed hybrid SNN-ANN SpikeGAN architecture. Sampling from the data set to obtain example $(x^i_{\leq T}, y^i_{\leq T})$ is indicated by dotted lines in the red box, along with a fixed conversion strategy from natural to spiking signals for the case of natural real data. For neuromorphic data (e.g., collected from a DVS sensor), there is no need for natural-to-spike conversion. The SNN generator, shown in the blue box, produces a sample $\tilde{x}^i_{\leq T}$. Real data and synthetic data are processed by an ANN discriminator that evaluates the likelihood $p_\Phi(x_{\leq T})$ that the input data is from the real data. In the case of Bayes-SpikeGAN, the model parameter $\phi^j$ of the generator are drawn from a posterior distribution, which is shown in the purple circle.

definition of the desired input-output behavior that does not fully account for the expressivity of spiking signals. Spiking signals can in fact encode the same natural signal in different ways by making use of coding in the spike times [94].

In this study, we address this issue by redefining the learning problem away from the approximation of specific input-output *patterns*, and towards the matching of the *distribution* of the SNN outputs with a target distribution, broadening the scope of possible output spike patterns. To this end, we propose an adversarial learning rule for SNNs, and explore its extensions to a Bayesian framework as well as to meta-learning.

### 5.1.1 Hybrid SNN-ANN Generative Adversarial Networks

The proposed adversarial learning approach follows the general architecture of generative adversarial networks (GANs) [43, 107]. A GAN architecture involves not only the target model, whose goal is generating samples approximately distributed according to the given desired "real" distribution, but also a discriminator. The role of the discriminator is to provide feedback to the generator during training concerning the discriminator's attempts to distinguish between real and synthetic samples produced by the generator. Generator and discriminator are optimized in an adversarial fashion, with the former aiming at decreasing the performance of the latter as a classifier of real against synthetic samples.

Unlike prior work on GANs, as illustrated in Figure 5.1, the key novel elements of the proposed architecture are that: (*i*) the generator is a *probabilistic SNN* tasked with the goal of reproducing spatio-temporal distributions in the space of spiking signals; and (*ii*) the discriminator is implemented via a standard artificial neural networks (ANNs). Concerning point (*i*), while in a conventional GAN model the randomness of the generator is due to the presence of stochastic, Gaussian, inputs to the generator, in this work we leverage the randomness produced by stochastic spiking neurons following the generalized linear model (GLM) [98, 51].

As for the novel item (*ii*), the adoption of a *hybrid SNN-ANN* architecture allows us to leverage the flexibility and power of ANN-based classifiers in order to enhance the training of the spiking generator. Optimization methods for ANNs are already well established, and ANN classifiers are known to achieve high accuracy on a variety of data sets, while classification using SNNs is still an active field of research. We therefore hypothesize that an ANN discriminator can complement the SNN generator by providing an accurate and adaptable learning signal. It is also worthwhile to explore the use of an SNN as the discriminator so as to enable online updates to the generator. This may provide the additional benefit of better credit

assignment over time. We focus here on developing and examining the hybrid SNN-ANN architecture, and leave the design of a pure spiking GAN as a direction of future investigation. It is emphasized that, once training is complete, the SNN acts as a standalone generator model, and the ANN-based discriminator can be discarded.

Once trained, the SNN can serve as a generator model to produce synthetic spiking data with the same statistical properties of real data. This data can be used to augment neuromorphic data sets that are limited in size, or as a generative replay for SNN learning. Furthermore, since the proposed model consists of a *conditional* generator, the trained SNN can also be used directly as a stochastic input-output mapping implementing supervised learning tasks without the need to hard-code spiking targets.

As an implementation note, in neuromorphic hardware, one may envision the ANN to be implemented on the same chip as the SNN in case the deployment requires continual, on-chip, learning; or to be part of auxiliary peripheral circuitry, possibly on an external device, in case the system is to be deployed solely for inference without requiring continual training.

### 5.1.2 Bayesian Spiking GANs

The randomness entailed by the presence of probabilistic spiking neurons, in much the same way as its conventional counterpart given by Gaussian inputs, may be insufficient to produce sufficiently *diverse* samples that cover real multi-modal distributions [106]. A way around this problem is to allow the model parameters to be stochastic too, such that new model parameters are drawn for each sample generation. This setting can be naturally modelled within a Bayesian framework, in which the model weights are given prior distributions that can be updated to produce posterior distributions during training as depicted by the distribution over generator weight $\phi$ in Figure 5.1. In order to enhance the diversity of the output samples, we investigate for the

first time the use of Bayesian spiking GANs, and demonstrate their potential use in reproducing biologically motivated spiking behavior [133].

### 5.1.3 Continual Meta-learning for Spiking GANs

In the continual learning setting, the statistics of the input data change over time, and a generative SNN must adapt to generate useful synthetic data. A generator model that is able to adapt based on few examples from the changed statistics is especially useful for augmenting the small data sets to be used in downstream applications such as an expert policy generator in reinforcement learning [132].

In Chapter 4, we presented a continual meta learning framework for online SNN learning applied to classification problems. Here, that framework is adapted to the problem of adversarial learning for spiking GANs. We follow the same general process shown in Figure 4.1, in which a hyperparameter initialization is optimized and then used to enable faster optimization on future tasks. In the proposed meta learning framework for spiking GANs, a joint initialization is learned for the adversarial network pair made up of the SNN generator and the ANN discriminator that improves the spiking GANs efficiency in learning to generate useful synthetic data as the statistics of the population distribution vary.

## 5.2 Related Work

The only, recent, paper that has proposed a spiking GAN is [62]. In it, the authors have introduced an adversarial learning rule based on temporal backpropagation with surrogate gradients by assuming a spiking generator and discriminator. The work focuses solely on encoded real-valued images, without consideration for neuromorphic data. Aside from the inclusion of Bayesian and continual meta-learning for the spiking GAN, our work explores the use of target distributions with temporal attributes, and adopts local learning rules based on probabilistic spiking models. Additionally, the

probabilistic SNN model that we have adopted includes natural stochasticity that facilitates generating multi-modal synthetic data, while in [62], the randomness is artificially injected via exogenous inputs sampled from a uniform distribution and time encoded.

Several recent works [117, 96, 135, 122, 119] have explored some form of hybrid SNN-ANN networks that combine SNNs and ANNs to capitalize on the low-power usage of SNNs and gain in accuracy from the broad range of processing capabilities and effective training algorithms for ANNs. Some examples of the applications are high-speed object tracking [135], classification [96, 117], gesture recognition [122] and robotic control [96]. While not the main focus of our work, the SNN generator that we propose is capable of learning a temporal embedding for natural signals similar to the SNN encoder in [119] with the key difference that in [119] the read-out signals are a compressed encoding of real data while in our case the signals are trained to emulate the real data. Both [117] and [96] propose chip designs to implement inference for such hybrid networks, and report increased classification accuracy for hybrid networks over pure SNN deployment with minimal increase in power usage. The Tianjic chip [96] also showcases the ability to deploy ANNs and SNNs that process simultaneously to achieve combined inference on a complex automated bicycle control task. In this case, a convolutional neural network (CNN) is used to extract features from large images while the SNN is responsible for processing auditory time series. In this work, we derive a similar benefit for the novel task of training a generative spiking model: the ANN is chosen as the discriminator to extract features of the real and synthetic data, while the SNN is responsible for modeling a temporal distribution.

## 5.3 Hybrid SNN-ANN Spiking GAN

In this section, we first describe the proposed hybrid SNN-ANN setting and then introduce the resulting training problem within a standard frequentist adversarial formulation.

### 5.3.1 Setting

Unlike prior work focused on the generation of static natural signals, we are concerned with generating spatio-temporal spiking data which consist of a sequence $x_{\leq T} = (x_{i,1}, ..., x_{i,\tau}, ...x_{i,T})_{i=1}^{N_x}$, of $N_x \times 1$ binary vectors $\{x_\tau\}_{\tau=1}^{T}$ over the time index $\tau = 1, ..., T$. The sequence is drawn from some unknown underlying population distribution $p(x_{\leq T}, y_{\leq T}) = p(y_{\leq T})p(x_{\leq T}|y_{\leq T})$, jointly with another spatio-temporal spiking signal $y_{\leq T} = (y_{i,1}, ..., y_{i,\tau}, ...y_{i,T})_{i=1}^{N_y}$ with $N_y$ binary vector $\{y_\tau\}_{\tau=1}^{T}$. The signal $y_{\leq T}$ may be made available to the generator $\mathcal{G}_\phi$ as an exogenous input to guide the generating mechanism. This auxiliary input signal can be useful to ensure that the generated data $x_{\leq T}$ cover a particular region of the data space, such as a specific class of spiking signals $x_{\leq T}$.

Since the generator SNN takes as input and produces as output spiking data, in case the actual data are defined over real-valued, or non-binary discrete alphabets, an encoding, or decoding, scheme can be used to either convert between the original data format and the binary time series processed by the spiking generator, or to convert the spiking output of the generator to the format of the real data.

The SNN generator $\mathcal{G}_\phi$ models the population distribution via a parameterized distribution $p_\phi(x_{\leq T}|y_{\leq T})$ that describes the statistics of the output of the $N_x$ read out neurons given the exogenous inputs $y_{\leq T}$. As detailed in Section 2.1.1, the parameter vector $\phi$ of the SNN includes synaptic weights and biases, with the latter playing the role of firing thresholds.

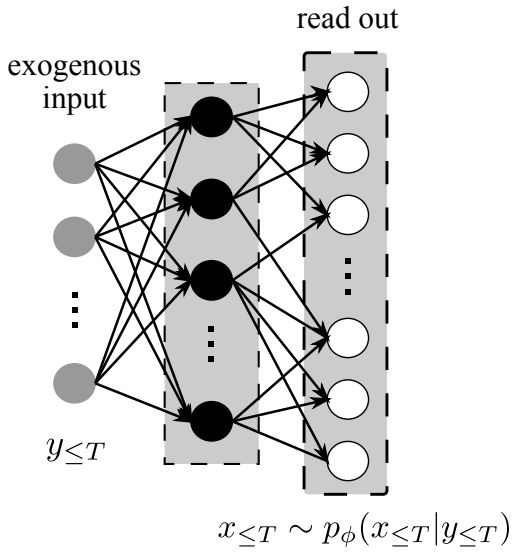


**Figure 5.2** An example of a generator SNN, $\mathcal{G}_\phi$, with a layered topology. Black circles are the hidden neurons, in set $\mathcal{H}$, and white circles are the read-out neurons in set $\mathcal{R}$, while gray circles represent exogenous inputs. Synaptic links are shown as directed arrows, with the post-synaptic spikes of the source neuron being integrated as inputs to the destination neuron.

The architecture of the ANN depends on whether the output of the SNN is directly fed to the ANN, which is the case when the original data are spiking signals, or if it is first converted back to a natural signal which is the case when the original data is static. The discriminator implements a classifier $\mathcal{D}_\Phi(x_{\leq T}) = p_\Phi(\xi = 1|x_{\leq T}, y_{\leq T})$ giving the probability that the input $(x_{\leq T}, y_{\leq T})$ is drawn from the real data distribution – an event indicated by the binary variable $\xi = 1$.

### 5.3.2 Training Problem Formulation

During training, real data pairs $(x_{\leq T}, y_{\leq T})$ are sampled from the data set. Recall that these are spiking signals, possibly converted from natural signals. The real data pair $(x_{\leq T}, y_{\leq T})$ along with a synthetic data pair $(\tilde{x}_{\leq T}, y_{\leq T})$, where $\tilde{x}_{\leq T}$ is the output of the generator SNN in response to input $y_{\leq T}$, are then fed as inputs to train the ANN discriminator $\mathcal{D}_\Phi$. Specifically, during training, the pair of SNN and ANN models are optimized jointly, with the discriminator's parameters $\Phi$ trained to

classify between the real and synthetic data, while the generator's parameters $\phi$ are updated to undermine the classification at the ANN.

Let us denote as $z_{\leq T} = (x_{\leq T}, y_{\leq T}) \sim p(x_{\leq T}, y_{\leq T})$ an input-output pair drawn from the underlying population distribution and $z_{\leq T} = (\tilde{x}_{\leq T}, y_{\leq T}) \sim p_\phi(\tilde{x}_{\leq T}, y_{\leq T})$ an input-output pair with $y_{\leq T} \sim p(y_{\leq T})$, drawn from the marginal population distribution, and $\tilde{x}_{\leq T} \sim p_\phi(\tilde{x}_{\leq T}|y_{\leq T})$ obtained from the SNN generator. In the single task frequentist setting studied in Section 5.5, the adversarial training objective is described by the min-max optimization problem [41]

$$\min_{\phi} \max_{\Phi} \mathbb{E}_{z_{\leq T} \sim p}[\psi_1(\mathcal{D}_\Phi(z_{\leq T}))] + \mathbb{E}_{z_{\leq T} \sim p_\phi}[\psi_2(\mathcal{D}_\Phi(z_{\leq T}))], \tag{5.1}$$

where $\psi_1(\cdot)$ is an increasing function and $\psi_2(\cdot)$ is a decreasing function. In Equation (5.1), the first expectation is over real data sampled from the true population distribution, while the second expectation is over synthetic data generated by the SNN. Following the original GAN formulation [41], we set $\psi_1 = \log(x)$ and $\psi_2 = \log(1 - x)$ so that the inner maximization in Equation (5.1) evaluates to the Jensen-Shannon divergence between the two distributions when no constraints are imposed on the discriminator.

As we detail in Section 5.5, in the proposed solution, stochastic gradient updates are made in a parallel fashion with the discriminator taking a gradient step to address the inner maximization in Equation (5.1), and the generator taking a gradient step to tackle the outer minimization in Equation (5.1).

In Section 5.6, the adversarial learning problem (Equation 5.1) is generalized to account for Bayesian SNNs in which the weight vector $\phi$ is allowed to have a posterior distribution, so that sample generation entails a preliminary step of sampling from the weight distribution [106]. Furthermore, in Section 5.7, the framework described in this section is extended to continual meta-learning, in which case the population distribution varies over time.

### 5.4 Probabilistic Spiking Neuronal Network Generator Model

In this section we describe the conditional probability distribution $p_\phi(x_{\leq T}|y_{\leq T})$ that is followed by the samples generated by the SNN. As illustrated in Figure 5.2, this distribution is realized by a general SNN architecture with probabilistic spiking neurons implementing generalized linear models [98, 55, 51] as detailed in Section 2.1.1. The generator SNN $\mathcal{G}_\phi$ processes data in the form of binary signals (spikes) over processing time $\tau = 1, 2, ...,$ with each neuron $i$ producing an output spike, $s_{i,\tau} = 1$, or no output spike, $s_{i,\tau} = 0$, at any time $\tau$. The network includes a layer of read-out neurons $\mathcal{R}$ and a set of hidden neurons $\mathcal{H}$, with respective spiking outputs denoted as $s_{i,\tau} = x_{i,\tau},\ i \in \mathcal{R}$ and $s_{i,\tau} = h_{i,\tau},\ i \in \mathcal{H}$. Both the read-out and hidden signals of the generator are latent variables in that no target output is observed. As depicted in Figure 5.2, the topology of the SNN is defined so as not to include any loops except for the individual neuron feedback signals.

Given the probabilistic expression of the output of GLM neurons in Equation (2.2), the membrane potentials of the read-out neurons and the hidden neurons define the joint likelihood of a sequence of read-out spikes $x_{\leq T} = \{[x_{i,0}, ..., x_{i,\tau}, ..., x_{i,T}]\}_{i\in\mathcal{R}}$ and hidden spikes $h_{\leq T} = \{[h_{i,0}, ..., h_{i,\tau}, ..., h_{i,T}]\}_{i\in\mathcal{H}}$. These sequences are sampled as a result of exogenous input sequence $y_{\leq T}$. Accordingly, the likelihood of the sequence of read-out spikes $x_{\leq T}$ is conditioned on some sequence of exogenous input spikes $y_{\leq T}$ and is defined as

$$\begin{aligned} p_\phi(x_{\leq T}, h_{\leq T}|y_{\leq T}) &= \prod_{\tau=1}^{T} \prod_{i\in\{\mathcal{R},\mathcal{H}\}} p_{\phi_i}(s_{i,\tau}|u_{i,\tau}) \\ &= \prod_{\tau=1}^{T} \prod_{i\in\{\mathcal{R},\mathcal{H}\}} \sigma(u_{i,\tau}). \end{aligned} \tag{5.2}$$

where $s_{i,\tau}$ refers to either a hidden spike signal $h_{i,\tau}$ or a read-out spike signal $x_{i,\tau}$ depending on which set the neuron $i$ that it is sampled from belongs to.

## 5.5 Adversarial Training for SNN: SpikeGAN

As described in Section 5.3, the proposed SpikeGAN model for adversarial training includes an SNN generator $\mathcal{G}_\phi$ with parameter vector $\phi$ and an ANN discriminator $\mathcal{D}_\Phi$ with parameter vector $\Phi$. As illustrated in Figure 5.1, the SNN generator processes exogenous inputs $y_{\leq T}$ in a sequential manner, mapping each $N_y \times 1$ input vector $y_\tau$ at time $\tau$ to an $N_x \times 1$ output vector $x_\tau$ for $\tau = 1, ..., T$. The SNN mapping is causal and probabilistic, with an output distribution $p_\phi(x_{\leq T}, h_{\leq T}|y_{\leq T})$ defined in Section 5.4. The discriminator $\mathcal{D}_\Phi$ is implemented as an ANN with a binary classification output. In this section we propose a method, referred to as SpikeGAN, to address the training problem in Equation (5.1).

### 5.5.1 Algorithm Overview

Consider the population distribution $p(x_{\leq T}, y_{\leq T})$ with side information $y_{\leq T}$ and data $x_{\leq T}$ that underlies the generation of a data set $\mathbb{D}$. At each training step, a batch of $B$ examples $(x^i_{\leq T}, y^i_{\leq T})$, for $i = 1, ..., B$ are drawn from the data set $\mathbb{D}$ for training. Additionally, a batch of synthetic data $\tilde{x}^i_{\leq T}$ is sampled from the spiking generator as the output spikes of the $N_x$ read-out neurons given the corresponding $N_y$ exogenous inputs $y^i_{\leq T}$, for $i = 1, ..., B$.

**Algorithm 4** SpikeGAN

**Input:** Data set $\mathbb{D} = \{(x^i_{\leq T}, y^i_{\leq T})\}_{i=1,2,...}$, learning rates $\mu_\Phi, \mu_\phi$

**repeat**
  sample a batch of real data samples $X = [x^i_{\leq T}]^B_{i=1}$ from the data set $\mathbb{D}$
  initialize synthetic data cache $\tilde{X} = \emptyset$
  initialize generator gradient cache $g = \emptyset$
  **for** $i = 1, ..., B$ **do**
    $x^i_{\leq T}, g^i_\phi \leftarrow$ SNN procedure (see below)
    cache sample $\tilde{X} = \tilde{X} \cup \{x^i_{\leq T}\}$
    cache gradients $g = g \cup \{g^i_\phi\}$
  **end for**
  evaluate classification probability $\mathcal{D}_\Phi(X)$ and $\mathcal{D}_\Phi(\tilde{X})$
  evaluate reward signal $r = [\psi_2\left(\mathcal{D}_\Phi(x^i_{\leq T})\right)]_{i=1,...,B}$
  Update $\Phi := \Phi + \mu_\Phi \frac{1}{B} \sum^B_{i=1} \nabla_\Phi \psi_1\left(\mathcal{D}_\Phi(x^i_{\leq T})\right) + \nabla_\Phi \psi_2\left(\mathcal{D}_\Phi(\tilde{x}^i_{\leq T})\right)$
  Update $\phi := \phi - \mu_\phi \frac{1}{B} \sum^B_{i=1} r^i g^i_\phi$
**until** convergence
**procedure** SNN
  initialize traces $h_{j,\tau} = 0$ for all neurons $j$ at time $\tau = 1$
  initialize gradients $g_{\phi_j} = 0$ for all pre-synaptic connection weights to neuron $j$
  **for** $\tau = 1, ..., T$ **do**
    **for** $j \in \mathcal{H}$ in order of connectivity **do**
      compute $u_{j,\tau}$ according to Equation 2.1
      sample $h_{j,\tau} = (h_{s,j,\tau}, h_{r,j,\tau})$
      accumulate gradients $g_{\phi_j} += \nabla_{\phi_j} p(h_{j,\tau} | u_{j,\tau})$ (Equation 2.8)
    **end for**
  **end for**
  **return** $h_{r,\leq T}, g_\phi$
**end procedure**

The SNN operates on the local discrete time scale defined by index $\tau = 1, ..., T$, which runs over the temporal dimension of each exogenous input sequence $y_{\leq T}$. At each time $\tau$ it processes a batch of $B$ input vectors $y^i_\tau$ with $i = 1, ..., B$, mapping them in parallel through the full network topology to sample a batch of $B$ corresponding output vectors $\tilde{x}^i_\tau$, with $i = 1, ..., B$ from the $N_x$ read-out neurons. As detailed in Section 4.4, this involves computing batches of instantaneous membrane potentials $u_{j,\tau}$ using Equation (2.1) and sampling output spikes using Equation (2.2) by following the order defined by the underlying computational graph.

The gradients with respect to the generator parameter $\phi$ for the learning criterion in Equation (5.1) are computed using a *local, three-factor, rule* that includes the gradient of the log likelihood of the joint output distribution (Equation 5.2), and accumulated as the output spikes of each neuron are sampled over time $\tau$. After the full sequence has been processed, the local gradients $g^i_\phi$, with $i = 1, ..., B$ are cached for use in the outer minimization in Equation (5.1).

The discriminator processes the batch of real data examples $\{(x^i_{\leq T}, y^i_{\leq T})\}^B_{i=1}$ and the batch of synthetic data samples $\{(\tilde{x}^i_{\leq T}, y^i_{\leq T})\}^B_{i=1}$ to approximate the expectations in the objective function given by Equation (5.1). For each example, the input to the discriminator includes both the data signal $x_{\leq T}$ and the feature signal $y_{\leq T}$. To enable the ANN to process the time series data, the series is either compressed to a fixed smaller-dimensional embedding, or the ANN includes convolutions over the time dimension to automatically optimize suitable embeddings. The gradient of the objective function with respect to the discriminator parameter $\Phi$ is evaluated using standard backpropagation.

### 5.5.2 Derivation

The GAN objective function given by Equation (5.1) is optimized via SGD updates with respect to the discriminator parameter vector $\Phi$ and generator parameter vector

$\phi$. To update the discriminator, the gradient of the expected values in Equation (5.1) are estimated by the described batches of $B$ examples drawn from the training data and from the generator as

$$\nabla_\Phi \mathbb{E}_{z_{\leq T}\sim p}\left[\psi_1\left(\mathcal{D}_\Phi(z_{\leq T})\right)\right] + \mathbb{E}_{z_{\leq T}\sim p_\phi}\left[\psi_2\left(\mathcal{D}_\Phi(z_{\leq T})\right)\right] \approx \tag{5.3}$$
$$\approx \frac{1}{B}\sum_{i=1}^{B} \nabla_\Phi \psi_1\big(\mathcal{D}_\Phi(z^i_{\leq T})\big) + \nabla_\Phi \psi_2\big(\mathcal{D}_\Phi(\tilde{z}^i_{\leq T})\big)\,,$$

where $z^i_{\leq T}$ is the $i$-th example sampled from the training data and $\tilde{z}^i_{\leq T}$ is the $i$-th example sampled from the generator. The derivatives are easily computed via the standard backpropagation algorithm.

Taking the gradient of the outer expression to update the generator model, we have

$$\begin{aligned}\nabla_\phi \mathbb{E}_{z_{\leq T}\sim p}\left[\psi_1\left(\mathcal{D}_\Phi(z_{\leq T})\right)\right] + \mathbb{E}_{\tilde{z}_{\leq T}\sim p_\phi}[\psi_2(\mathcal{D}_\Phi(\tilde{z}_{\leq T}))] &= \\ = \nabla_\phi \mathbb{E}_{\tilde{z}_{\leq T}\sim p_\phi}[\psi_2(\mathcal{D}_\Phi(\tilde{z}_{\leq T}))] &\\ = \sum_{x_{\leq T}, h_{\leq T}} \log\left(1 - \mathcal{D}_\Phi(\tilde{z}_{\leq T})\right) \nabla_\phi p_\phi(x_{\leq T}, h_{\leq T} \mid y_{\leq T}), &\end{aligned} \tag{5.4}$$

where the derivative of the first term evaluates to zero [41] and the gradient of the second term is expanded to highlight the average over synthetic data $\tilde{x}_{\leq T}$, which is jointly distributed with the hidden spike signals $h_{\leq T}$, weighted by the likelihood of each possible value. The REINFORCE gradient is applied to estimate this average by sampling from the SNN. For the case of a single sample, this gives the approximate gradient

$$\sum_{\tilde{x}_{\leq T}, h_{\leq T}} \log\left(1 - \mathcal{D}_\Phi(\tilde{z}_{\leq T})\right) \nabla_\phi p_\phi(\tilde{x}_{\leq T}, h_{\leq T} \mid y_{\leq T}) \approx$$
$$\approx \log\big(1 - \mathcal{D}_\Phi(\tilde{z}^i_{\leq T})\big)\, \nabla_\phi \log p_\phi(\tilde{x}^i_{\leq T}, h^i_{\leq T} \mid y^i_{\leq T}) \tag{5.5}$$

in which the gradient of the log likelihood of spiking signals sampled from the SNN is given as in [51] by Equation (2.8). This yields a local, three-factor rule [38] which follows the general form

$$w_{j,i} \leftarrow w_{j,i} + \eta(r^i \cdot g^i_{j,i}) \tag{5.6}$$

for the update to the synaptic weights $w_{j,i}$ where $0 < \eta < 1$ is the learning rate, $r^i$ is the global reward signal for the $i$-th sample from the SNN generator and $g_{j,i}$ is the local neuron gradient that depends on the filtered input and output spikes. The key point of SpikeGAN is that the global reward signal $r^i = \psi_2(\mathcal{D}_\Phi(\tilde{x}^i_{\leq T}))$ is given by the classification certainty of the discriminator. This makes intuitive sense in that SNN connection strength is decreased if the generated outputs $x^i_{\leq T}$ are likely to be synthetic data according to the discriminator, and they are enforced in the opposite case.

In specified experiments in Section 5.9, to avoid vanishing gradients early in training, we adopt the commonly used alternative generator optimization objective $\max_\phi \mathbb{E}_{z_{\leq T} \sim p_\phi} [\log (\mathcal{D}_\Phi(z_{\leq T}))]$, in which the generator parameter $\phi$ is updated to maximize the log likelihood that the synthetic data is mis-classified as real data [41]. The resulting gradient has the same general form as Equation (5.4).

### 5.5.3 Comparison with Other Methods

Maximum likelihood (ML) learning for SNNs optimizes the likelihood that the output signals of the read-out neurons match a target binary sequence. This can be implemented using the framework outlined in Section 2.2. A major benefit of adversarial training for SNNs as compared to ML learning is its potential to better capture the different modes of the population distribution. In contrast, ML tends to produce inclusive approximations that overestimate the variance of the population distribution. A practical advantage of ML learning, as detailed in [51] is that it

enables online, incremental, learning. The proposed GAN training algorithm applies an episodic rule in which the global learning signal can only be evaluated after a full sequence of outputs has been sampled from the SNN due to the choice of an ANN as the discriminator. An online learning variant could be also devised by defining the discriminator as a recurrent network [137], but we leave this topic to future research.

## 5.6 Bayes-SpikeGAN

In the Section 5.5, we have explored frequentist adversarial training for SNNs. By forcing the choice of a single model parameter vector $\phi$ for the generator $\mathcal{G}_\phi$, the approach may fail at reproducing the diversity of multi-modal population distribution [106]. As an example, it has been shown that tailored synaptic filters and spiking thresholds are necessary to induce specific temporal patterns at the output of a spiking neuron [133], which are incompatible with the choice of a single parameter vector $\phi$. In this section we explore the application of Bayesian adversarial learning to address this problem.

### 5.6.1 Generalized Posterior

The Bayes-SpikeGAN assumes a prior distribution, $p(\phi)$, over the generator parameter vector $\phi$, and, rather than optimizing over a single parameter vector, it obtains a generalized posterior distribution on $\phi$ given observed real data. For a fixed ANN discriminator, the posterior distribution can be defined as [106]

$$p(\phi|y_{\leq T}, \Phi) \propto p(\phi)\mathbb{E}_{\tilde{x}_{\leq T}\sim p_\phi}\left[\mathcal{D}_\Phi(\tilde{x}_{\leq T})\right], \tag{5.7}$$

where the expectation is over synthetic data $\tilde{x}_{\leq T}$ sampled from the distribution $p_\phi(x_{\leq T})$ defined by the generator $\mathcal{G}_\phi$. In Equation (5.7), the average confidence of the discriminator $\mathcal{D}_\Phi(\tilde{x}_{\leq T})$ plays the role of likelihood of the current generator parameter $\phi$ given the observed real data used to optimize discriminator parameter vector $\Phi$.

**Algorithm 5** Bayes-SpikeGAN

**Input:** Data set $\mathbb{D} = \{(x^i_{\leq T}, y^i_{\leq T})\}_{i=1,2,...}$, learning rates $\mu_\Phi, \mu_\phi$

initialize $J$ SNN generators $\mathcal{G}_\phi = \{\mathcal{G}_{\phi^j}\}_{j=1}^J$ each with parameter $\phi^j$

initialize CNN discriminator $D_\Phi$

**repeat**

  sample a batch of real data samples $X = [x^i_{\leq T}]_{i=1}^B$ from the data set $\mathbb{D}$

  **for** each SNN generator $G_{\phi^j}$ **do**

    initialize synthetic data cache $\tilde{X} = \emptyset$

    initialize generator gradient cache $g = \emptyset$

    **for** $i = 1, ..., B$ **do**

      $x^i_{\leq T}, g^i_{\phi^j} \leftarrow$ SNN procedure (see Algorithm 4)

      cache sample $\tilde{X} = \tilde{X} \cup \{x^i_{\leq T}\}$

      cache gradients $g = g \cup \{g^i_{\phi^j}\}$

    **end for**

    evaluate reward signal $r = [-\log\big(\mathcal{D}_\Phi(x^i_{\leq T})\big)]_{i=1}^B$

    cache gradients w.r.t. all weights in $\phi^j$

$$\nabla_{\phi^j} \mathbb{E}_{\tilde{z}_{\leq T} \sim p_{\phi^j}}[-\log(\mathcal{D}_\Phi(\tilde{z}_{\leq T}))] = \frac{1}{B} \sum_{i=1}^{B} r^i g^i_\phi$$

    evaluate and cache classification probability $\mathcal{D}_\Phi(X)$ and $\mathcal{D}_\Phi(\tilde{X})$

  **end for**

  **for** each SNN generator parameter $\phi^j$ **do**

    Update $\phi^j$ following Eq. (5.10) using cached gradients w.r.t. all generator parameters $\{\phi^j\}_{j=1}^J$

  **end for**

  Update discriminator parameter:

$$\Phi := \Phi + \mu_\Phi \frac{1}{JB} \sum_{j=1}^{J} \nabla_\Phi \Big[\psi_1\big(\mathcal{D}_\Phi(X^j_{\leq T})\big) + \psi_2\Big(\mathcal{D}_\Phi(\tilde{X}^j_{\leq T})\Big)\Big]$$

**until** convergence

### 5.6.2 Training Objective

Computing the generalized posterior $p(\phi)$ in Equation (5.7) is generally intractable, and hence we approximate it with a variational distribution $q(\phi|y_{\leq T}, \Phi)$. The variational posterior $q(\phi|y_{\leq T}, \Phi)$ is optimized by addressing the problem of minimizing the free energy metric

$$\min_{q(\phi)} -\log\left(\mathbb{E}_{\tilde{x}_{\leq T}\sim q(\phi)}[\mathcal{D}_\Phi(\tilde{x}_{\leq T})]\right) - \mathrm{KL}(q(\phi)||p(\phi)), \tag{5.8}$$

where $\mathrm{KL}(q(\phi)||p(\phi)) = \mathbb{E}_{\phi\sim q(\phi)}[\log(q(\phi)/p(\phi))]$ is the Kullback-Liebler (KL) divergence. If no constraints are imposed on the distribution $q(\phi)$, the optimal solution of the problem in Equation (5.8) is exactly Equation (5.7). We further apply Jensen's inequality to obtain the more tractable objective

$$\min_{q(\phi)} -\mathbb{E}_{\tilde{x}_{\leq T}\sim q(\phi)}\left[\log \mathcal{D}_\Phi(\tilde{x}_{\leq T})\right] - \mathrm{KL}(q(\phi)||p(\phi)). \tag{5.9}$$

In order to address this problem, we parametrize the variational posterior with a set of $J$ parameter vectors $\phi = \{\phi^j\}_{j=1}^J$, also known as particles. This effectively defines $J$ SNN generators $\{\mathcal{G}_{\phi^j}\}_{j=1}^J$. Samples from the generator are then obtained by randomly and uniformly selecting one particle from the set of $J$ particles, and then using the selected sample $\phi^j$ to run the SNN generator $\mathcal{G}_{\phi^j}$.

As explained next, in order to optimize the set of particles with the goal of minimizing the free energy metric in Equation (5.8), we leverage Stein variational gradient descent (SVGD) [71].

### 5.6.3 Bayes-SpikeGAN

Following SVGD, for a fixed discriminator parameter vector $\Phi$, the particles are updated simultaneously at each iteration as

$$\phi_{i+1}^j = \phi_i^j - \eta\sum_{j'=1}^{J}\left\{\kappa(\phi_i^j, \phi_i^{j'})\Big(-\nabla_{\phi^{j'}}\mathbb{E}_{\tilde{x}_{\leq T}\sim p_{\phi^{j'}}}[\log\left(p(\phi)\mathcal{D}_\Phi(\tilde{x}_{\leq T})\right)]\Big) - \nabla_{\phi^{j'}}\kappa(\phi_i^j, \phi_i^{j'})\right\} \tag{5.10}$$

where $\kappa(\phi^j, \phi^{j'}) = \exp(-||\phi^j - \phi^{j'}||^2)$ is the Gaussian kernel function. The gradient with respect to the generator parameter $\phi_i^{j'}$ can be computed as in Equation (5.4), which is estimated via the REINFORCE gradient as $r^i \cdot g_{i,j}^i$ as in Equation (5.6).

In each iteration, the discriminator parameter, $\Phi$, is updated via SGD to optimize the standard GAN loss function given in Equation (5.1) by taking an average of the losses calculated for data sampled from each of the $J$ generators.

## 5.7 Continual Meta-Learning for Spiking GANs: Meta-SpikeGAN

We have so far defined two adversarial training methods for SNNs, namely SpikeGAN (based on frequentist learning) and Bayes-SpikeGAN to train an SNN to generate data that follows a single population distribution. We now focus on the general continual meta-training framework introduced in Chapter 4 that can be combined with SpikeGAN adversarial training in order to enable the SpikeGAN to efficiently, and sequentially, learn to generate data from a range of similar population distributions.

Specifically, we assume that a common hyperparameter $\theta$ can be identified, representing a parameter initialization for the SNN that yields efficient learning when applied separately for each task in a family of statistically similar tasks, $\mathcal{F}$. The hyperparameter initialization should improve the learning efficiency of the within-task training in terms of the total updates necessary to obtain a useful within-task model. This problem is approached with the aim of improving the across task generalization capability of the hyperparameter as new tasks arrive in a streaming fashion, while maintaining the ability to quickly recover the within task parameter learned for previous tasks.

### 5.7.1 Meta-SpikeGAN

The continual meta-learning framework detailed in Section 4.3 is adapted to the SpikeGAN architecture described in Section 5.5 – a system we will refer to as

meta-SpikeGAN. The meta-SpikeGAN follows the timeline introduced in Section 4.3 with the stream of tasks over meta-time steps defined as $t = 0, 1, ...$ and the stream of within-task data over within-task time steps as $i = 0, 1, ...$, such that each within-task time step is associated with a tuple $(t, i)$. We introduce two meta-models, a discriminator ANN $\mathcal{D}_\Theta$, and a spiking generator $\mathcal{G}_\theta$, whose weights $\Theta$ and $\theta$ respectively, define the hyperparameters $\boldsymbol{\theta}^{(t,i)} = \{\Theta^{(t,i)}, \theta^{(t,i)}\}$ that will be updated during the meta learning process. The two hyperparameter vectors must be learned synchronously in order to maintain the balance between the discriminator and the generator in the min-max process of adversarial learning described by Equation (5.1). In particular, it is important that the discriminator learn to differentiate real and synthetic data quickly, based on few examples from the new task, in order to provide a meaningful learning signal to the spiking generator. While the derivation here follows a frequentist framework, extensions to Bayesian solutions could be obtained by following the approach detailed in the previous section.

In meta-SpikeGAN, the within-task iterative update function Update$(\theta, D)$ refers to the adversarial training process described in Section 5.5 in which both models are updated to learn within task parameters $\Phi$ and $\phi$ starting from initializations $\Theta$ and $\theta$ respectively. At each within-task time-step $(t, i)$, $N+1$ adversarial model pairs are instantiated with initial weight given by $\boldsymbol{\theta}^{(t,i)}$. One pair is trained using data from the current task $T^{(t)}$ to generate within task synthetic data, while the remaining $N$ adversarial network pairs are used to enable the meta-update. We note that the notation $T$ is overloaded here to represent the tasks in the meta-learning framework (annotated by the task index $t$ in the superscript) in addition to the SNN sample length.

The meta-objective, following the classic MAML formulation, is to optimize the min-max adversarial training objective over the hyperparameter initialization $\boldsymbol{\theta}$, given the learned within-task parameters across multiple previously seen tasks. Specifically,

the objective is defined as an average over $N$ tasks with data sets $D^{(n)}$ stored in the meta-data buffer as

$$\min_{\theta} \max_{\Theta} \sum_{n=1}^{N} \sum_{i=1}^{B} \log\Big(\mathcal{D}_{\Phi^{(n)}}(x_{\leq T}^{(n),i})\Big) + \sum_{i=1}^{B} \log\Big(1 - \mathcal{D}_{\Phi^{(n)}}(\tilde{x}_{\leq T}^{(n),i})\Big)\, p_{\phi^{(n)}}(\tilde{x}_{\leq T}^{(n),i} \,|\, y_{\leq T}^{(n),i}) \quad (5.11)$$

where the real data is sampled from the data set $D^{(n)}$ and the synthetic data is sampled from the generator $G_{\phi^{(n)}}$ trained via within-task adversarial training for that task.

To implement the meta-update function Meta-Update $\Big(\boldsymbol{\theta}^{(t,i)}, \{D^{(n)}\}_{n=1}^{N}\Big)$, the mentioned $N$ adversarial network pairs are trained in parallel according to the Update$(\boldsymbol{\theta}, D)$ using $N$ data-sets sampled from the meta-data buffer $\{D^{(n)}\}_{n=1}^{N} \in \mathcal{B}^{(t)}$. Each of the data-sets includes $M$ training examples from the real data that are a subset of the data set of a previously seen task such that $D^{(n)} = \{(x_{\leq \mathcal{T}}^{j}, y_{\leq \mathcal{T}}^{j})\}_{j=1}^{M}$. Once the within task parameters for both networks ($\mathcal{D}_{\Phi^{(n)}}$ and $\mathcal{G}_{\phi^{(n)}}$) for each of the $N$ tasks are learned, the hyperparameters $\Theta^{(t,i)}$ and $\theta^{(t,i)}$ are each updated individually following the first-order REPTILE approximation for the gradient of Equation (5.11) as

$$\Theta^{(t,i+1)} = \Theta^{(t,i)} - \Phi^{(n)} \quad (5.12)$$

$$\theta^{(t,i+1)} = \theta^{(t,i)} - \phi^{(n)}. \quad (5.13)$$

## 5.8 Experimental Methods

In this section, we describe the experimental set-up we have adopted to evaluate the performance of SpikeGAN, Bayes-SpikeGAN, and meta-SpikeGAN.

### 5.8.1 Data Sets, Encoding, and Decoding

We consider three different data sets: 1) handwritten digits [29]; 2) simulated spike-domain handwritten digits; and 3) synthetic temporal data [133]. These data sets have been selected to present a range of spatial and temporal correlations, posing different

challenges to the training of a generative model. The handwritten digits data set represents a population distribution with exclusively spatial correlations, as there is no temporal aspect to the real data. The simulated spike domain handwritten digits data set incorporates some temporal correlations by using a spike code to convert the handwritten digit data into the spike domain. Lastly, the synthetic temporal data has a dimension of $N_x = 1$ and thus includes only strong temporal features and no spatial correlations. The data sets are detailed in the next section.

For the first data set, the SNN outputs at the read-out layer are compressed to match the domain of the real data, using rate decoding or time surface decoding [122]. Rate decoding computes the ratio $\sum_{t=1}^{T} h_{r,i,t}/T$ of the number of output spikes at any read-out neuron $i$ to example length $T$. Alternatively, time surface decoding [122] convolves an exponentially decaying kernel over the time dimension of the read-out spike sequence, and outputs the last sample of the convolution. For the second data set, the real data is rate-encoded as a $T$-length time sequence by sampling from a Bernoulli process with probability $p$ corresponding to the pixel value [40]. For the third data set, there are exogenous inputs to the generator to encourage diverse samples from the distribution, as we will discuss.

For the first data set, the discriminator is a $74 \times 128 \times 1$ feedforward ANN with ReLU activation functions. For the second and third data sets, the discriminator includes several one-dimensional convolutional layers that filter over the time dimension of the data to extract temporal features and learn a natural embedding. The number of layers, as well as the attributes of each layer (number of channels, kernel width, and stride length) are chosen to best match each data set and are detailed in Section 5.9. The output of the temporal filter is flattened and processed through a linear layer. The approach is adapted from [60].

### 5.8.2 Benchmarks and Performance Metrics

In order to evaluate how well the output of the SNN generator approximates the underlying population distribution for the data, the following measures are used.

1. *Train on synthetic – Test on real (TSTR)* [31]: A classifier is trained over data sampled from the conditional SNN generator for all classes, and test accuracy is evaluated on data sampled from a held-out test set of the real data set. The resulting TSTR error measure provides insight into how well the attributes of the data that are important to distinguish the different data classes have been modeled by the distribution of the SNN outputs. It specifically captures how fully the SNN samples represent the sample space of the true distribution: If there are outlying portions of the sample space that are not well covered by the sample distribution, the corresponding real data samples may be missclassified, yielding a large TSTR error.

2. *Train on real – Test on synthetic (TRTS)* [31]: A classifier is trained over data from the real data set, and test accuracy is evaluated on synthetic data sampled from the conditional SNN generator for all classes. This measure highlights how well the samples of the synthetic data distribution stay within the bounds of the real data distribution – or how realistic the samples are.

3. *Principal component analysis (PCA)*: Extract the principal components of the real data set and compare the projection of a synthetic data set sampled from the SNN generator into that space to the projection of the real data. Plotting the principal component projections gives a visual representation of how well the sample space of the synthetic distribution matches that of the true distribution [31].

As a benchmark, we consider deep adversarial belief networks (DBNs) [49]. DBNs output a single binary sample and hence they can be used only when the data is a vector as for the first data set. They serve as a useful baseline comparison to the proposed SpikeGAN for the problem of generating real valued handwritten digit images (first data set) in that, like the proposed SNN model, they also implement probabilistic neurons with binary processing capabilities. However, importantly, they lack the capacity to process information encoded over time.

For temporal real data, i.e., for the second and third data sets, we consider maximum likelihood (ML) learning for a spiking variational auto encoder as detailed in [51] as a benchmark.

In order to evaluate the similarity between the generated temporal data and the real temporal data of the third data set, we use the van Rossum metric [129]. This metric measures the dissimilarity between two spike trains by first filtering the spike trains to convert them to continuous-valued time series, and then computing the Euclidean distance between them. Following [47], we use the exponential kernel $\exp(-(t-t_s)/\tau_R)$ for filtering, where $t_s$ is the time of a spike and the time constant that sets the width of the filter $\tau_R$ is set to 12 ms.

## 5.9 Results and Discussion

In this section, we present our main results by discussing separately the three data sets mentioned in the previous section. We first evaluate single-task performance, and then provide examples also for the continual meta-learning setting. We will mostly focus on the frequentist SpikeGAN approach detailed in Section 5.3, but we will also elaborate on the potential advantages of Bayes-SpikeGAN in the context of the third data set.

### 5.9.1 Handwritten Digits

For this first experiment, the UCI handwritten digits data set [29] is considered as defining the real data distribution $p(x|y)$ conditioned on the class label $y$ for $y \in \{0, 1, ..., 9\}$ as a one hot vector. We encode label $y$ as a time sequence input $y_{\leq T}$ for the SNN using rate encoding. The SNN's task is to learn a temporal distribution $p_\phi(x_{\leq T}|y_{\leq T})$ such that the distribution of the synthetic samples $x_{\leq T}$, processed via a fixed decoding scheme, approximates the samples drawn from the real distribution $p(x|y)$. As explained in the previous section, both standard rate decoding and time

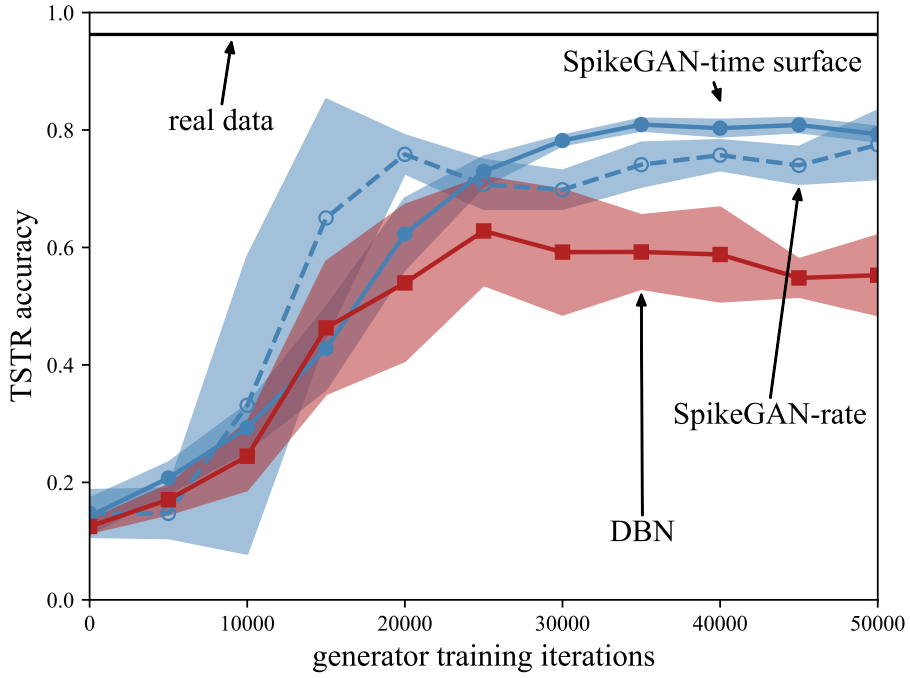


**Figure 5.3** TSTR classification accuracy for synthetic data sampled from the SNN generator during training on the handwritten digits dataset. The black line is the ideal test accuracy for a classifier trained with real data. The blue lines are results from SpikeGAN with outputs converted to images using rate (blue dashed) or time surface (blue solid) decoding, while the red line represents the performance of the DBN.

surface decoding are considered for the SNN output sequences $x_{\leq T}$. As an application of the approach, after training, the synthetic data may be considered a temporal representation of the true data and can be used as a neuromorphic data set.

The real data samples $x$ are $8 \times 8$ grayscale images with values in the range $[0, 1]$. The SNN generator includes 10 exogenous inputs $y_{\leq T}$, $H_s = 100$ supplementary neurons, and $H_r = 64$ read-out neurons producing output $x_{\leq T}$; and the SNN has a fully connected topology. An exponential decay basis function $\exp(-\tau/\tau_f)$, with $\tau = 0, 1, ..., \tau_w$, filter length $\tau_w = 5$, and decay rate parameter $\tau_f = 2$, is adopted for both the pre- and post-synaptic filters $\alpha$ and $\beta$ under rate decoding; while a set of two raised cosine basis functions [98] is used under time surface decoding.

The TSTR classification accuracy metric is first evaluated by using a $64 \times 100 \times 100 \times 10$ non-spiking ANN classifier with ReLU activation functions that achieves a

baseline of 96% test accuracy when trained on real data, and the results are shown in Figure 5.3 as a function of the training iterations for the generator. SpikeGAN approaches this ideal accuracy level, while far outperforming the DBN. In this regard, it is noted that, while SpikeGAN can generate grayscale data when paired with a decoder, the DBN can only generate binary data.

We have also considered the same experiment in a more challenging setting in terms of image size and sample diversity. Here, the data set images are $10 \times 10$ handwritten digit images obtained from the MNIST data set. The architecture of the SNN is updated to accommodate the larger image size by including $H_r = 100$ read-out neurons. All hyper-parameters are the same as in the last experiment and images are decoded via rate-decoding. The TRTS classification accuracy metric is evaluated by training the previously described ANN classifier over the real $10 \times 10$ MNIST images and testing its accuracy on the rate-decoded synthetic images. Our experiments show that the classifier achieves an accuracy of 90.6% when tested on the synthetic images, as compared to a baseline test accuracy of 98% when tested on real images.

Next we compare the SpikeGAN and DBN GAN in terms of robustness to noise on the UCI digits data set. The noisy data set is constructed by adding uniform noise to a fraction of the pixels in each image selected at random. For the DBN, the images are first binarized as in [49] in order to improve the performance, which was otherwise found to be too low in this experiment. The extent to which the digits can be distinguished from the noise in the resulting noisy synthetic images is evaluated using the TRTS accuracy measure. A classifier with the same architecture as in the previous experiment is trained on the uncorrupted real handwritten digit data set and tested on the noisy synthetic data with a baseline comparison to the classifier tested on noisy real data.

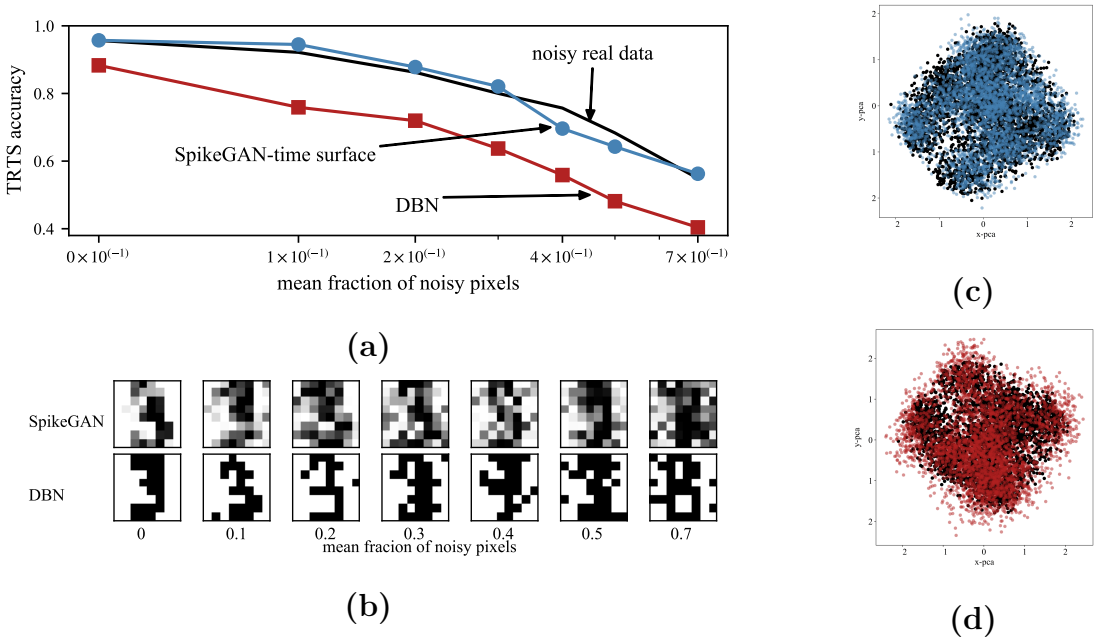


**Figure 5.4** (a) Handwritten digit classification accuracy for test data sampled from a generator trained using a noisy real data set. The fraction of pixels per image corrupted by additive uniform noise is increased on a log scale. Test data is sampled from either SpikeGAN (blue), DBN GAN (red) or the noisy real data set (black) as a baseline. (b) Synthetic data sampled from SpikeGAN with time surface decoding (top) and DBN GAN (bottom) trained over real data disrupted by additive uniform noise. The fraction of noisy pixels per image in the real data increases from left to right as labeled. (c,d) PCA projections of SpikeGAN synthetic data (top) and DBN GAN synthetic data (bottom) onto the real data principal components. The real data projection is shown by the black dots in both figures.

As reported in Figure 5.4a, the SpikeGAN noisy synthetic data, using time surface decoding, is classified more accurately than the DBN generated noisy synthetic data and maintains an accuracy close to the baseline obtained by testing as the fraction of noisy pixels is increased. This suggests that the capacity of the SNN to generate grayscale images is instrumental in enabling the classifier to distinguish the digits from the noise. This interpretation is corroborated by the samples of synthetic images from SNNs and DBNs shown in Figure (5.4b). Even for the case of zero pixels with added noise, the SpikeGAN synthetic data is seen to be more realistic than the binary DBN synthetic data. A more quantitative support to this observation is supported by the PCA projections in Figures 5.4d and 5.4c. The SpikeGAN projection

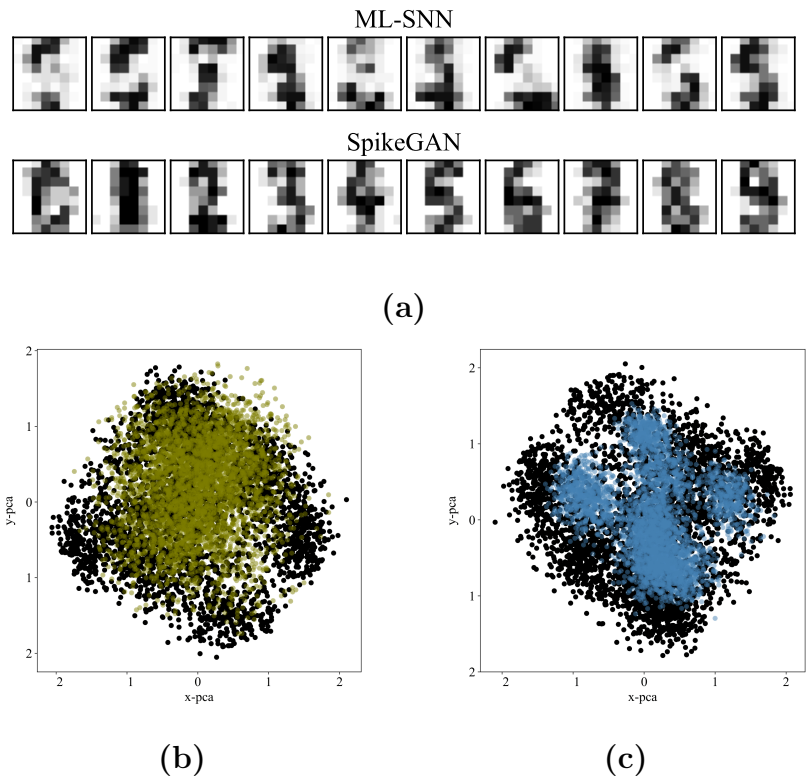


**Figure 5.5** (a) Rate-decoded outputs sampled from an ML-trained SNN [51] (top) and SpikeGAN with a CNN discriminator (bottom). (b, c) PCA projections of a large set of samples drawn from the SpikeGAN and from an ML-SNN, respectively (black - real data)

follows the shape of the data projection, while the DBN projection has many points outside of it.

### 5.9.2 Simulated Neuromorphic Handwritten Digits

In this second experiment, we move beyond the problem of generating time domain embeddings of real valued data sets by considering the problem of generating synthetic data that matches a spatio-temporal distribution. To simulate a spike domain data set, the UCI handwritten digits data set is encoded via rate encoding to produce the inputs $x_{\leq T}$. The label for each example $y$ that is used as the conditional input, is also encoded using rate encoding as $y_{\leq T}$ before being processed by the discriminator. The discriminator is defined as c128k4s2xc1k4s1x1 (c(number of channels)k(kernel width)s(stride)) with leaky ReLU activation functions, while the SNN generator

architecture is the same as in the previous experiment. The key difference between this experiment and the previous is that here the output of the SNN generator is not converted into a real vector before being fed to the discriminator, since the goal is to reproduce the spatio-temporal distribution of the input spiking data set.

As a benchmark, DBN is not relevant since it cannot generate temporal data, and an SNN trained vial ML as in [51] is used as a reference. Synthetic images are shown by decoding the spiking generator output using the reverse of the encoding scheme applied to the real data, here rate decoding, in Figure 5.5, in order to provide a qualitative idea of how well the spatio-temporal distribution has been matched for SpikeGAN and ML training. The PCA projections show that SpikeGAN can represent the mulit-modal structure of the true data distribution more accurately that ML, which is know to be support covering and inclusive [84, 68, 116].

To evaluate the quality of the synthetic data as a neuromorphic data set we now train an SNN classifier using the ML approach in [51] based on the synthetic data, and report the TSTR accuracy metric in Table 5.1. The SNN classifier processes 64 exogenous inputs which are the flattened input image, and includes 256 hidden neurons and 10 visible neurons in a fully connected topology. The visible neurons are clamped to the class labels $y_{\leq T}$ that the synthetic data was conditioned on. The table shows that the SpikeGAN that is trained with a CNN discriminator so that the output directly reproduces a spiking data set enables a better classifier than the SpikeGAN that is trained using a fixed decoder (whether a time surface decoder or rate decoder) to match a real dataset. The ML-SNN does not generate images that match the class label $y_{\leq T}$ that the sample is conditioned on (see Figure 5.5) which leads to a poor classifier. The CNN discriminator SpikeGAN approaches the baseline performance reported for the SNN classifier trained over rate encoded real data.

As a more challenging data set in this setting, inspired by [123], we consider the Double UCI digits data set, in which the classes are defined by the indices of

**Table 5.1** TSTR Accuracy of SNN Classifier Trained via ML for Simulated Neuromorphic Handwritten Digits

| Training Data | Real Data Test Accuracy |
|---|---|
| Rate encoded real data | 0.85 |
| SpikeGAN (CNN discriminator) | 0.82 |
| SpikeGAN time surface decode | 0.8 |
| SpikeGAN rate decode | 0.8 |
| ML-SNN | 0.12 |

two digits, and each sample image is a concatenation of two single UCI digit images which creates an $8 \times 16$ image. The class label for each example includes two one hot vectors, one for each digit. The class labels are rate encoded to be used as the conditional input to the generator. The discriminator and generator architectures are both updated to accommodate the larger data size with shape c128k4s2c128k4s1x1 and 20x100x128 respectively. The quality of the synthetic data is evaluated using an SNN classifier similar to the one described above, with the only differences being the number of inputs (now 128) and the number of visible neurons (now 20). The same class label encoding strategy is applied to clamp the visible neurons of the SNN classifier to the class labels during training. Our results show that the SNN classifier achieves 80% accuracy when trained using the synthetic data, as compared to a baseline of 87% accuracy when trained on the rate encoded real data.

### 5.9.3 Synthetic Temporal Data

For the last SpikeGAN experiment, a synthetic temporal data set is constructed by taking inspiration from biological neuronal behavior [133]. The goal is to assess whether adversarial unsupervised training can reproduce some of the diversity shown by neuronal activity in the brain. We specifically consider two biologically inspired neural spike modes, namely tonic spiking and burst spiking. In a manner similar to [133], we define burst spiking as periods of five consecutive spikes followed by a

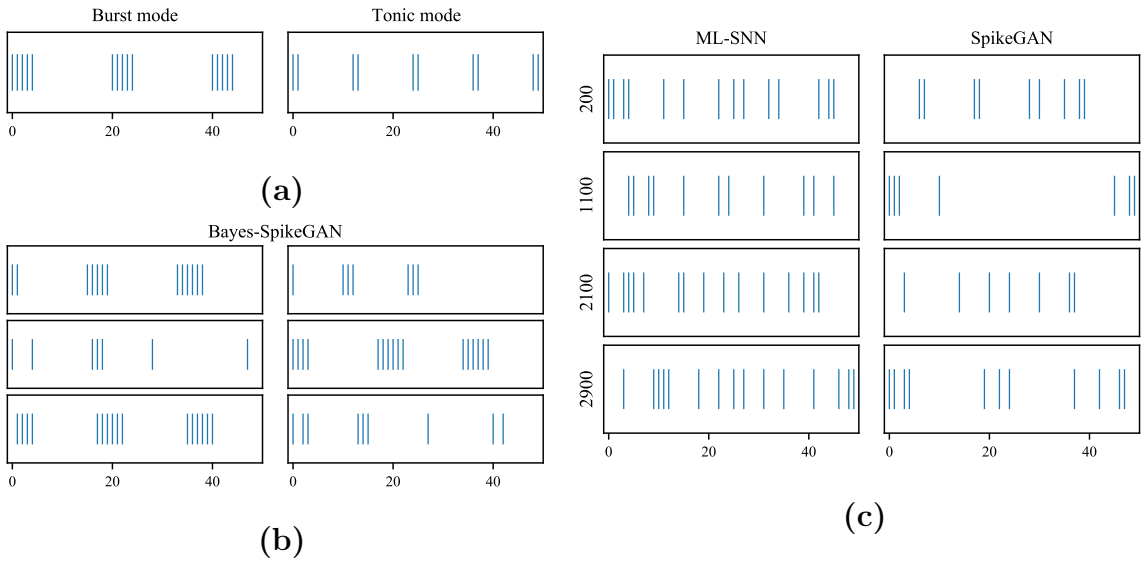


**Figure 5.6** (a) 'Real' examples of the two modes found in the synthetic temporal data set (b) Time sequence samples drawn from the Bayes-SpikeGAN (right) after training is completed. (c) Time sequence samples drawn from the ML-SNN (left) and SpikeGAN (right). Samples are taken after the model has been trained for the number of iterations indicated by the label on the left.

non-spiking period of 15 time steps while tonic spiking includes two consecutive spikes with a 10 time step non-spiking period as displayed in Figure 5.6a.

As shown in [133], a single neuron with tailored synaptic filters is sufficient to approximate either one of these modes well. Both the synaptic filter and spiking threshold (bias) of the neuron need to be carefully optimized to maintain this behavior. We generate a data set of 10,000 burst and tonic spiking sequences of length $T = 50$. We compare the performance of the SpikeGAN, and Bayes-SpikeGAN with an SNN trained via ML as the baseline. Specifically, we train the ML-based SNN and the SpikeGAN each with a single neuron with $T = 50$ and synaptic filter memory $\tau_w = 30$ that is stimulated by an exogenous step function input, as well as Bayes-SpikeGAN with $J = 5$ of the single neuron SNN generators to approximate the posterior distribution over $\phi$. For the Bayes-SpikeGAN we make the choice of an improper constant prior for $p(\phi)$ in the SVGD update rule (Equation 5.10).

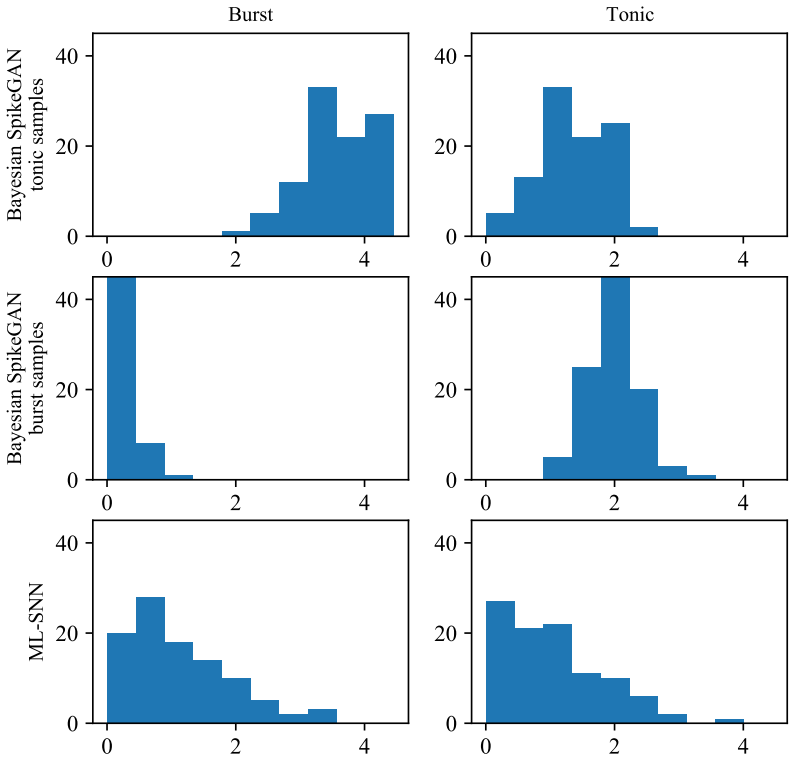


**Figure 5.7** Distribution of van Rossum distances between 100 generated samples and a real temporal data sample. The column labels "Burst" and "Tonic" denote which mode of temporal data the generated data is compared to. For each plot, the generated samples were drawn from the model labeled on the left hand side. The Bayesian SpikeGAN includes a set of models to sample from, which have learned to generate different modes of the data. The models generating data most similar to the tonic mode have been sampled separately from the models generating data most similar to the burst mode, as indicated by the labels "tonic samples" and "burst samples".

As seen in Figure 5.6c, the ML-trained SNN generates outputs that are a blend of the two modes while the SpikeGAN oscillates between them as training proceeds. In contrast, as seen in Figure 5.6b, Bayes-SpikeGAN is able to learn a set of generators whose combined outputs cover both modes simultaneously.

The mode of the samples generated by each generator in the Bayesian-SpikeGAN can be visually identified as either "burst" or "tonic", and thus an example of a chosen mode can be sampled by choosing the correct generator model from the set of learned models. The generators are sampled from in this way, and the van Rossum distance between the generated samples and the real temporal data is

evaluated for each mode individually. Specifically, the following four distances are evaluated: 1) tonic generated data vs. burst real data; 2) tonic generated data vs. tonic real data; 3) burst generated data vs. burst real data; 4) burst generated data vs. tonic real data. The distance should be small between like modes, and large between different modes. The histogram plots in the top two rows of Figure 5.7 show that the distance is small when the mode of the samples generated by the SpikeGAN matches the mode of the real data (top right and middle left); and the large in the opposite case. In contrast, with conventional ML training of the SNN [51] the distribution of the distance between the generated samples and real data from either mode is seen to be wide, with no distinct spikes to indicate a strong similarity or dissimilarity for any samples. This supports the hypothesis that, unlike SpikeGAN, ML has learned to generate samples that are a mean of the two modes.

### 5.9.4 Continual Meta-Learning

We now evaluate the ability of meta-SpikeGAN to continually improve its efficiency in generating useful time domain embeddings by focusing on the real-valued handwritten digits data set studied in Section 5.9.1. The real data set that defines each task $T^{(t)}$ is chosen as the subset of the UCI handwritten digits data set obtained by selecting the combination of two digits from among digits 0-6, along with a randomly sampled rotation of 90° applied to each digit. Digits 7-9 are reserved for testing. The class labels $y \in \{0, 1\}$ are applied to the pair of rotated digits in each new task. As in the previous handwritten digits experiment, the SNN generator is conditioned on the class label as a one hot vector encoded as time sequence $y_{\leq T}$ using rate encoding with $T = 5$. The output of the SNN generator $x_{\leq T}$ is decoded back to a natural signal using rate decoding before being fed to the discriminator.

The SNN generator includes two exogenous inputs $y_{\leq T}$, $H_s = 100$ supplementary neurons and $H_r = 64$ read-out neurons producing output $x_{\leq T}$ and has a

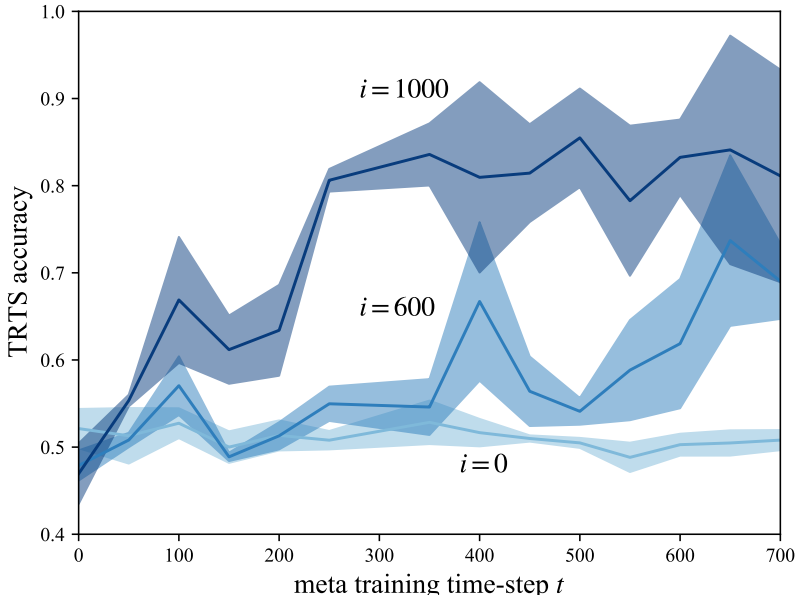


**Figure 5.8** Within task TRTS classification accuracy for hybrid adversarial network pair using the hyperparameter initialization learned over epochs of continual meta training ($t$). Lines are labeled with the number of within-task adversarial training iterations ($i$), and show the average over three tasks drawn from a held out set of digits (shading shows half standard deviation spread).

fully connected topology. The same exponential decay synaptic filters are used as described in Section 5.9.1. We implement the Meta-Update($\boldsymbol{\theta}^{(t,i)}, \{D^{(n)}\}_{n=1}^{N}$) function with $N = 10$ within-task data sets of $M = 5$ examples each and 10 within-task update steps.

The performance of SpikeGAN under the meta-SpikeGAN initialization $\boldsymbol{\theta}^{(t,i)} = (\Theta^{(t,i)}, \theta^{(t,i)})$ is evaluated by looking at the TRTS accuracy for synthetic data generated at intervals throughout within-task training. If training efficiency has been improved by the meta-SpikeGAN initialization, the TRTS accuracy will be higher after fewer within-task training updates $i$. The continual improvement of the meta-SpikeGAN hyperparameter initialization is measured by applying the initialization to a SpikeGAN model after every 50 meta-training time-steps $t$ and by evaluating the TRTS accuracy throughout within-task training on a new task.

We choose a new task as the combination of two digits from the set of held-out digits (digits 7-9) of the UCI handwritten digits data set and train the SpikeGAN

over mini-batches of $B = 100$ training examples. As shown in Figure 5.8, the TRTS accuracy improves significantly as the meta-SpikeGAN hyperparameter initialization is learned, with the accuracy after $i = 1000$ within-task updates increasing by 30% over a randomly initialized SpikeGAN ($t = 0$ meta training time-steps) as meta training progresses.

## 5.10 Conclusion

This chapter has introduced adversarial training methods for a novel hybrid SNN-ANN GAN architecture, termed SpikeGAN. The proposed approaches solve the problem of learning how to emulate a spatio-temporal distribution, while allowing for a flexible, distribution-based, definition of the target outputs that fully leverages the temporal encoding nature of spiking signals. Both frequentist and Bayesian formulations of the learning problem were considered, along with a generalization to continual meta-learning. The proposed SpikeGAN approach has been evaluated on a range of spatio-temporal data sets, and has been shown to outperform current baselines (DBN GANs and SNNs based on ML training) in all settings. Bayes-SpikeGAN is proven to be an important extension to the frequentist learning solution in the problem of emulating multi-modal data with large variations in specific temporal patterns, such as for biologically inspired spiking sequences.

The introduction of the SpikeGAN approach gives rise to many avenues of further research, including its application to alternative data sets, as well as changes to the architecture and development of additional learning rules. For instance, the proposed SpikeGAN may be used to generate colored data sets by the addition of channels to the generator SNN. Other interesting applications are the prediction and reconstruction of time series data such as biological spiking data, stock exchange data or audio signals. Time series processing is an area that has been explored in the context of ANN-based GAN models as reviewed in [16]. The proposed Meta-

SpikeGAN learning rule may be extended to include Bayesian Meta-learning, for example by leveraging the approach introduced in [136]. The architecture may be adapted to include an SNN as the discriminator to yield a fully-spiking model that would support the development of an online learning rule.

# CHAPTER 6

# CONCLUSION AND FUTURE OUTLOOK

In this dissertation we have developed a suite of machine learning algorithms for spiking neural networks (SNNs) based on principles of probabilistic learning that covers several machine learning frameworks. In all cases, local three-factor learning rules have been derived to support on-chip implementation in neuromorphic hardware. These probabilistic learning rules are enabled by the consideration of stochastic SNNs based on a generalized linear model neuron that implements a stochastic spiking process rather than a standard deterministic spiking process.

We have developed a policy gradient reinforcement learning algorithm in which an SNN models the policy distribution of an agent. The actions taken by the agent are stochastically sampled directly from the policy using the first-to-spike decision rule. We have demonstrated that the first-to-spike policy gradient algorithm converges more quickly than an equivalently sized ANN. We have also shown the power efficiency of the stochastic SNN policy in terms of spike frequency to be $10\times$ better than a baseline deterministic SNN policy with model weights set by conversion from an ANN. While this algorithm is on-policy and model free, it is episodic, which would require the intermediate local gradients and rewards to be cached in an on-chip implementation. An adaptation to accommodate online or intermittent online updates could be explored by reward function tuning.

Meta-learning has been explored as a method by which to improve the adaptability and learning efficiency of SNN models for personalized applications where learning on the edge is imperative. We have developed a first order online-within-online meta-learning algorithm and shown its ability to train a hyperparameter initialization for an SNN that improves the speed of model adaptation in terms

of number of samples need to achieve good accuracy on new tasks. We have demonstrated the applicability of this algorithm to neuromorphic data with information encoded in the spike timing. Our results have shown that the meta-learned hyperparameter initialization allows an SNN to achieve 80% accuracy for 5-shot, 2-way classification of the neuromorphic MNIST-DVS data set, representing a 20% increase over the accuracy achieved in the same task when the SNN is trained from a random initialization. The consideration of meta-learning based on second order derivatives is likely to further improve this performance.

The common practice of defining exact target sequences for SNNs may limit representations of information in the spike timing, a key characteristic of biological signals that is tied to low power, event based processing [128]. We have developed a hybrid ANN-SNN adversarial learning framework in which learning is guided by a measure of the divergence between the distribution of the spiking outputs and that of the target data. In our simulations we demonstrate that a generative SNN trained by this method can be used to augment a neuromorphic data set by generating naturally time encoded data. We have extended this to a Bayesian framework that learns a particle based approximation of the posterior distribution over the model parameters that is shown to capture temporally diverse modes of the data as evaluated by the van Rossum distance between spike sequences. We have also demonstrated the successful application of online meta-learning to adversarial SNN learning.

The natural extension of this work lies in emulation and implementation of the algorithms that we have developed in neuromorphic hardware. The literature shows that many small adjustments and sometimes larger approximations must be made to bridge the gap between simulation and reality [124, 64, 100]. It is also important to investigate the benefits and challenges associated with expanding the size of the models considered and the potential methods by which to do so successfully whether through alternative variational learning methods such as the reparameterization trick

(see, e.g., [116]) or by other methods. Furthermore, to truly capture the benefits of the naturally recurrent and event based processing of SNNs, their application to time-series problems (for example, prediction of sequences and reconstruction of noisy time based signals in the context of adversarial learning) should be explored.

## REFERENCES


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

[2] Maruan Al-Shedivat, Rawan Naous, Emre Neftci, Gert Cauwenberghs, and Khaled N Salama. Inherently stochastic spiking neurons for probabilistic neural computation. In *IEEE International Conference on Neural Engineering (NER)*, pages 356–359, 2015.

[3] Ron Amit and Ron Meir. Meta-learning by adjusting priors based on extended pac-bayes theory. In *Proceedings of Machine Learning Research International Conference on Machine Learning*, pages 205–214, 2018.

[4] Muhammad Arsalan, Avik Santra, and Vadim Issakov. Radarsnn: A resource efficient gesture sensing system based on mm-wave radar. *IEEE Transactions on Microwave Theory and Techniques*, 2022.

[5] Daniel Auge, Julian Hille, Etienne Mueller, and Alois Knoll. A survey of encoding techniques for signal processing in spiking neural networks. *Neural Processing Letters*, 53(6):4693–4710, 2021.

[6] Alireza Bagheri, Osvaldo Simeone, and Bipin Rajendran. Training probabilistic spiking neural networks with first-to-spike decoding. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2986–2990, 2018.

[7] Jonathan Baxter. Theoretical models of learning to learn. In *Learning to Learn*, pages 71–94. Boston, MA: Springer, 1998.

[8] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. *arXiv preprint arXiv:1711.05136*, 2017.

[9] Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. *Advances in Neural Information Processing Systems*, 31, 2018.

[10] Guillaume Bellec, Franz Scherr, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets. *arXiv preprint arXiv:1901.09049*, 2019.

[11] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[12] Guo-qiang Bi and Mu-ming Poo. Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *The Journal of Neuroscience*, 18(24):10464, 1998.

[13] Zhenshan Bing, Claus Meschede, Kai Huang, Guang Chen, Florian Rohrbein, Mahmoud Akl, and Alois Knoll. End to end learning of spiking neural network based on r-stdp for a lane keeping vehicle. In *IEEE International Conference on Robotics and Automation*, pages 4725–4732, 2018.

[14] Zhenshan Bing, Claus Meschede, Florian Röhrbein, Kai Huang, and Alois C. Knoll. A survey of robotics control based on learning-inspired spiking neural networks. *Frontiers in Neurorobotics*, 12, 2018.

[15] Thomas Bohnstingl, Franz Scherr, Christian Pehle, Karlheinz Meier, and Wolfgang Maass. Neuromorphic hardware learns to learn. *Frontiers in Neuroscience*, 13:483, 2019.

[16] Eoin Brophy, Zhengwei Wang, Qi She, and Tomas Ward. Generative adversarial networks in time series: A survey and taxonomy. *arXiv preprint arXiv:2107.11098*, 2021.

[17] Lars Buesing, Johannes Bill, Bernhard Nessler, and Wolfgang Maass. Neural dynamics as sampling: a model for stochastic computation in recurrent networks of spiking neurons. *PLoS Computational Biology*, 7(11), 2011.

[18] Karla Burelo, Georgia Ramantani, Giacomo Indiveri, and Johannes Sarnthein. A neuromorphic spiking neural network detects epileptic high frequency oscillations in the scalp eeg. *Scientific Reports*, 12(1), 2022.

[19] Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113(1):54–66, 2015.

[20] Natalia Caporale and Yang Dan. Spike timing–dependent plasticity: A hebbian learning rule. *Annual Review of Neuroscience*, 31(1):25–46, 2008.

[21] Chin-Jui Chang, Yu-Wei Chu, Chao-Hsien Ting, Hao-Kang Liu, Zhang-Wei Hong, and Chun-Yi Lee. Reducing the deployment-time inference control costs of deep reinforcement learning agents via an asymmetric architecture. In *IEEE International Conference on Robotics and Automation*, volume 2021-May, pages 4762–4768, 2021.

[22] Mingzhe Chen, Ursula Challita, Walid Saad, Changchuan Yin, and Mérouane Debbah. Machine learning for wireless networks with artificial intelligence: A tutorial on neural networks. *arXiv preprint arXiv:1710.02913*, 9, 2017.

[23] Wuhui Chen, Xiaoyu Qiu, Ting Cai, Hong-Ning Dai, Zibin Zheng, and Yan Zhang. Deep reinforcement learning for internet of things: A comprehensive survey. *IEEE Communications Surveys and Tutorials*, 23(3):1659–1692, 2021.

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

[25] Sophie Deneve. Bayesian spiking neurons i: inference. *Neural Computation*, 20(1):91–117, 2008.

[26] Giulia Denevi, Dimitris Stamos, Carlo Ciliberto, and Massimiliano Pontil. Online-within-online meta-learning. In *Advances in Neural Information Processing Systems*, 2019.

[27] Peter U Diehl et al. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *International Joint Conference on Neural Networks*, pages 1–8. IEEE, 2015.

[28] Meng Dong, Xuhui Huang, and Bo Xu. Unsupervised speech recognition through spike-timing-dependent plasticity in a convolutional spiking neural network. *PLoS One*, 13(11), 2018.

[29] Dheeru Dua and Casey Graff. University of California Irvine machine learning repository, 2017. Irvine, CA.

[30] Steven K. Esser, Paul A. Merolla, John V. Arthur, Andrew S. Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J. Berg, Jeffrey L. McKinstry, Timothy Melano, Davis R. Barch, Carmelo di Nolfo, Pallab Datta, Arnon Amir, Brian Taba, Myron D. Flickner, and Dharmendra S. Modha. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences of the United States of America*, 113(41):11441–11446, 2016.

[31] Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*, 2017.

[32] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. Personalized federated learning: A meta-learning approach. *arXiv preprint arXiv:2002.07948*, 2020.

[33] Haowen Fang, Amar Shrestha, Ziyi Zhao, and Qinru Qiu. Exploiting neuron and synapse filter dynamics in spatial temporal learning of deep spiking neural network. In *International Joint Conference on Artificial Intelligence*, volume 2021-January, pages 2799–2806, 2021.

[34] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[35] Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. Online meta-learning. *arXiv preprint arXiv:1902.08438*, 2019.

[36] Răzvan V Florian. Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Computation*, 19(6):1468–1502, 2007.

[37] Mohamed Said Frikha, Sonia Mettali Gammar, Abdelkader Lahmadi, and Laurent Andrey. Reinforcement and deep reinforcement learning for wireless internet of things: A survey. *Computer Communications*, 178:98–113, 2021.

[38] Nicolas Frémaux and Wulfram Gerstner. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in Neural Circuits*, 9, 2016.

[39] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge, U.K.: Cambridge University Press, 2002.

[40] Matthieu Gilson, Timothée Masquelier, and Etienne Hugues. Stdp allows fast rate-modulated coding with poisson-like spike trains. *PLoS Computational Biology*, 7(10):e1002231, 2011.

[41] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems*, 27, 2014.

[42] Keren Gu, Sam Greydanus, Luke Metz, Niru Maheswaranathan, and Jascha Sohl-Dickstein. Meta-learning biologically plausible semi-supervised update rules. *bioRxiv*, 2019.

[43] Jie Gui, Zhenan Sun, Yonggang Wen, Dacheng Tao, and Jieping Ye. A review on generative adversarial networks: Algorithms, theory, and applications. *arXiv preprint arXiv:2001.06937*, 2020.

[44] Nguyen-Dong Ho and Ik-Joon Chang. Tcl: An ann-to-snn conversion with trainable clipping layers. In *Design Automation Conference*, volume 2021-December, pages 793–798, 2021.

[45] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500, 1952.

[46] John Joseph Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

[47] Conor Houghton and Thomas Kreuz. On the efficient calculation of van rossum distances. *Network: Computation in Neural Systems*, 23(1-2):48–58, 2012.

[48] Ruihan Hu, Sheng Chang, Hao Wang, Jin He, and Qijun Huang. Efficient multispike learning for spiking neural networks using probability-modulated timing method. *Transactions on Neural Networks and Learning Systems*, 30(7):1984–1997, 2018.

[49] Yuming Huang, Ashkan Panahi, Hamid Krim, Yiyi Yu, and Spencer L Smith. Deep adversarial belief networks. *arXiv preprint arXiv:1909.06134*, 2019.

[50] Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

[51] Hyeryung Jang, Osvaldo Simeone, Brian Gardner, and Andre Gruning. An introduction to probabilistic spiking neural networks: Probabilistic models, learning rules, and applications. *IEEE Signal Processing Magazine*, 36(6):64–77, 2019.

[52] Hyeryung Jang, Nicolas Skatchkovsky, and Osvaldo Simeone. Spiking neural networks – part I: Detecting spatial patterns. *arXiv preprint arxiv:2010.14208*, 2020.

[53] Hyeryung Jang, Nicolas Skatchkovsky, and Osvaldo Simeone. VOWEL: A local online learning rule for recurrent networks of probabilistic spiking winner-take-all circuits. In *International Conference on Pattern Recognition*, 2020.

[54] Yihan Jiang, Jakub Konečnỳ, Keith Rush, and Sreeram Kannan. Improving federated learning personalization via model agnostic meta learning. *arXiv preprint arXiv:1909.12488*, 2019.

[55] Danilo Jimenez Rezende and Wulfram Gerstner. Stochastic variational learning in recurrent spiking networks. *Frontiers in Computational Neuroscience*, 8:38, 2014.

[56] Xin Jin, Alexander Rast, Francesco Galluppi, Sergio Davies, and Steve Furber. Implementing spike-timing-dependent plasticity on spinnaker neuromorphic hardware. In *International Joint Conference on Neural Networks*, pages 1–8, 2010.

[57] Sharu Theresa Jose and Osvaldo Simeone. Information-theoretic generalization bounds for meta-learning and applications. *arXiv preprint arXiv:2005.04372*, 2020.

[58] David Kappel et al. A dynamic connectome supports the emergence of stable computational function of neural circuits through reward-based learning. *eNeuro*, 5(2):ENEURO–0301, 2018.

[59] Avi Karni, Gundela Meyer, Christine Rey-Hipolito, Peter Jezzard, Michelle M Adams, Robert Turner, and Leslie G Ungerleider. The acquisition of skilled motor performance: fast and slow experience-driven changes in primary motor cortex. *Proceedings of the National Academy of Sciences*, 95(3):861–868, 1998.

[60] Wonjun Ko, Jeeseok Yoon, Eunsong Kang, Eunji Jun, Jun-Sik Choi, and Heung-Il Suk. Deep recurrent spatio-temporal neural network for motor imagery based bci. In *International Conference on Brain-Computer Interface*, pages 1–3. IEEE, 2018.

[61] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *International Conference on Machine Learning: Deep Learning Workshop*, volume 2, 2015.

[62] Vineet Kotariya and Udayan Ganguly. Spiking-gan: A spiking generative adversarial network using time-to-first-spike coding. *arXiv preprint arXiv:2106.15420*, 2021.

[63] Levin Kuhlmann, Michael Hauser-Raspe, Jonathan H Manton, David B Grayden, Jonathan Tapson, and André van Schaik. Approximate, computationally efficient online learning in bayesian spiking neurons. *Neural Computation*, 26(3):472–496, 2014.

[64] Shruti R Kulkarni and Bipin Rajendran. Spiking neural networks for handwritten digit recognition—supervised learning and network optimization. *Neural Networks*, 103:118–127, 2018.

[65] Viraj Kulkarni, Milind Kulkarni, and Aniruddha Pant. Survey of personalization techniques for federated learning. *arXiv preprint arXiv:2003.08673*, 2020.

[66] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[67] Chankyu Lee, Gopalakrishnan Srinivasan, Priyadarshini Panda, and Kaushik Roy. Deep spiking convolutional neural network trained with unsupervised spike-timing-dependent plasticity. *IEEE Transactions on Cognitive and Developmental Systems*, 11(3):384–394, 2019.

[68] Yingzhen Li and Richard E Turner. Rényi divergence variational inference. *arXiv preprint arXiv:1602.02311*, 2016.

[69] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7(1):1–10, 2016.

[70] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie Mellon University, 1993.

[71] Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. *arXiv preprint arXiv:1608.04471*, 2016.

[72] Shih-Chii Liu, Andre van Schaik, Bradley A Minch, and Tobi Delbruck. Asynchronous binaural spatial audition sensor with 2×64×4 channel output. *IEEE Transactions on Biomedical Circuits and Systems*, 8(4):453–464, 2013.

[73] Milton Llera-Montero, João Sacramento, and Rui Ponte Costa. Computational roles of plastic probabilistic synapses. *Current Opinion in Neurobiology*, 54:90–97, 2019.

[74] Andrew Lobbezoo, Yanjun Qian, and Hyock-Ju Kwon. Reinforcement learning for pick and place operations in robotics: A survey. *Robotics*, 10(3), 2021.

[75] Wolfgang Maass and Michael Schmitt. On the complexity of learning for spiking neurons with temporal coding. *Information and Computation*, 153(1):26–46, 1999.

[76] Alberto Marchisio, Giacomo Pira, Maurizio Martina, Guido Masera, and Muhammad Shafique. R-snn: An analysis and design methodology for robustifying spiking neural networks against adversarial attacks through noise filters for dynamic vision sensors. In *IEEE International Conference on Intelligent Robots and Systems*, 2021.

[77] Henry Markram, Joachim Lübke, Michael Frotscher, and Bert Sakmann. Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps. *Science*, 275(5297):213–215, 1997.

[78] Stephen J Martin, Paul D Grimwood, and Richard GM Morris. Synaptic plasticity and memory: an evaluation of the hypothesis. *Annual Review of Neuroscience*, 23(1):649–711, 2000.

[79] Timothée Masquelier and Simon J Thorpe. Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS Computational Biology*, 3(2):e31, 2007.

[80] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha. Artificial brains. a million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–73, 2014.

[81] Luke Metz, Niru Maheswaranathan, Brian Cheung, and Jascha Sohl-Dickstein. Meta-learning update rules for unsupervised representation learning. *arXiv preprint arXiv:1804.00222*, 2018.

[82] Thomas Miconi, Jeff Clune, and Kenneth O Stanley. Differentiable plasticity: training plastic neural networks with backpropagation. *arXiv preprint arXiv:1804.02464*, 2018.

[83] Beren Millidge, Alexander Tschantz, and Christopher L Buckley. Predictive coding approximates backprop along arbitrary computation graphs. *arXiv preprint arXiv:2006.04182*, 2020.

[84] Tom Minka et al. Divergence measures and message passing. Technical report, Microsoft Research, Citeseer, 2005.

[85] Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller, and Volodymyr. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[86] Abigail Morrison, Markus Diesmann, and Wulfram Gerstner. Phenomenological models of synaptic plasticity based on spike timing. *Biological Cybernetics*, 98(6):459–478, 2008.

[87] Elias Najarro and Sebastian Risi. Meta-learning through hebbian plasticity in random networks. *arXiv preprint arXiv:2007.02686*, 2020.

[88] Takashi Nakano, Makoto Otsuka, Junichiro Yoshimoto, and Kenji Doya. A spiking neural network model of model-free reinforcement learning with high-dimensional sensory input and perceptual ambiguity. *PloS One*, 10(3):e0115620, 2015.

[89] Emre O Neftci, Bruno U Pedroni, Siddharth Joshi, Maruan Al-Shedivat, and Gert Cauwenberghs. Stochastic synapses enable efficient brain-inspired learning machines. *Frontiers in Neuroscience*, 10:241, 2016.

[90] Emri O. Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019.

[91] Bernhard Nessler, Michael Pfeiffer, Lars Buesing, and Wolfgang Maass. Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity. *PLoS Computational Biology*, 9(4), 2013.

[92] Thy Nguyen, A Steven Younger, Emmett Redd, and Tayo Obafemi-Ajayi. Meta-learning related tasks with recurrent networks: Optimization and generalization. In *International Joint Conference on Neural Networks*, pages 1–8. IEEE, 2018.

[93] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[94] Zihan Pan, Jibin Wu, Malu Zhang, Haizhou Li, and Yansong Chua. Neural population coding for effective temporal classification. In *International Joint Conference on Neural Networks*, pages 1–8. IEEE, 2019.

[95] Dejan Pecevski, Lars Buesing, and Wolfgang Maass. Probabilistic inference in general graphical models through sampling in stochastic networks of spiking neurons. *PLoS Computational Biology*, 7(12), 2011.

[96] Jing Pei, Lei Deng, Sen Song, Mingguo Zhao, Youhui Zhang, Shuang Wu, Guanrui Wang, Zhe Zou, Zhenzhi Wu, Wei He, et al. Towards artificial general intelligence with hybrid tianjic chip architecture. *Nature*, 572(7767):106–111, 2019.

[97] Jonathan W Pillow, Liam Paninski, Valerie J Uzzell, Eero P Simoncelli, and EJ Chichilnisky. Prediction and decoding of retinal ganglion cell responses with a probabilistic spiking model. *Journal of Neuroscience*, 25(47):11003–11013, 2005.

[98] Jonathan W Pillow, Jonathon Shlens, Liam Paninski, Alexander Sher, Alan M Litke, EJ Chichilnisky, and Eero P Simoncelli. Spatio-temporal correlations and visual signalling in a complete neuronal population. *Nature*, 454(7207):995, 2008.

[99] Steven D Pyle, Ramtin Zand, Shadi Sheikhfaal, and Ronald F Demara. Subthreshold spintronic stochastic spiking neural networks with probabilistic hebbian plasticity and homeostasis. *Journal on Exploratory Solid-State Computational Devices and Circuits*, 5(1):43–51, 2019.

[100] Bipin Rajendran, Abu Sebastian, and Evangelos Eleftheriou. Building next-generation ai systems: Co-optimization of algorithms, architectures, and nanoscale memristive devices. In *IEEE International Memory Workshop*, 2019.

[101] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[102] Bleema Rosenfeld, Bipin Rajendran, and Osvaldo Simeone. Fast on-device adaptation for spiking neural networks via online-within-online meta-learning. In *Data Science and Learning Workshop*, 2021.

[103] Bleema Rosenfeld, Osvaldo Simeone, and Bipin Rajendran. Learning first-to-spike policies for neuromorphic control using policy gradients. In *IEEE International Workshop on Signal Processing Advances in Wireless Communications*, pages 1–5. IEEE, 2019.

[104] Bleema Rosenfeld, Osvaldo Simeone, and Bipin Rajendran. Spiking generative adversarial networks with a neural network discriminator: Local training, bayesian models, and continual meta-learning. *arXiv preprint arXiv:2111.01750*, 2021.

[105] Bodo Rueckauer and Shih-Chii Liu. Conversion of analog to spiking neural networks using sparse temporal coding. In *IEEE International Symposium on Circuits and Systems*, volume 2018-May, 2018.

[106] Yunus Saatci and Andrew Wilson. Bayesian gans. In *Advances in Neural Information Processing Systems*, pages 3624–3633, 2017.

[107] Divya Saxena and Jiannong Cao. Generative adversarial networks (gans) challenges, solutions, and future directions. *ACM Computing Surveys*, 54(3):1–42, 2021.

[108] Franz Scherr, Christoph Stöckl, and Wolfgang Maass. One-shot learning with spiking neural networks. *bioRxiv*, 2020.

[109] Teresa Serrano-Gotarredona and Bernabé Linares-Barranco. A 128×128 1.5% contrast sensitivity 0.9% fpn 3 $\mu$s latency 4 mw asynchronous frame-free dynamic vision sensor using transimpedance preamplifiers. *IEEE Journal of Solid-State Circuits*, 48(3):827–838, 2013.

[110] Teresa Serrano-Gotarredona and Bernabé Linares-Barranco. Poker-DVS and MNIST-DVS. their history, how they were made, and other details. *Frontiers in Neuroscience*, 9:481, 2015.

[111] Jiangrong Shen, Jian K. Liu, and Yueming Wang. Dynamic spatiotemporal pattern recognition with recurrent spiking neural network. *Neural Computation*, 33(11):2971–2995, 2021.

[112] Myung Seok Shim and Peng Li. Biologically inspired reinforcement learning for mobile robot collision avoidance. In *International Joint Conference on Neural Networks*, pages 3098–3105. IEEE, 2017.

[113] Ajay Shrestha and Ausif Mahmood. Review of deep learning algorithms and architectures. *IEEE Access*, 7:53040–53065, 2019.

[114] Amar Shrestha, Haowen Fang, Qing Wu, and Qinru Qiu. Approximating back-propagation for a biologically plausible local learning rule in spiking neural networks. In *International Conference on Neuromorphic Systems*, pages 1–8, 2019.

[115] Sumit B. Shrestha and Garrick Orchard. Slayer: Spike layer error reassignment in time. In *Advances in Neural Information Processing Systems*, volume 2018-December, pages 1412–1421, 2018.

[116] Osvaldo Simeone. A brief introduction to machine learning for engineers. *Foundations and Trends in Signal Processing*, 12(3-4):200–431, 2018.

[117] Sonali Singh, Anup Sarma, Nicholas Jao, Ashutosh Pattnaik, Sen Lu, Kezhou Yang, Abhronil Sengupta, Vijaykrishnan Narayanan, and Chita R Das. Nebula: a neuromorphic spin-based ultra-low power architecture for snns and anns. In *ACM International Symposium on Computer Architecture*, pages 363–376. IEEE, 2020.

[118] Nicolas Skatchkovsky, Hyeryung Jang, and Osvaldo Simeone. Federated neuromorphic learning of spiking neural networks for low-power edge intelligence. In *International Conference on Acoustics, Speech and Signal Processing*, pages 8524–8528. IEEE, 2020.

[119] Nicolas Skatchkovsky, Osvaldo Simeone, and Hyeryung Jang. Learning to time-decode in spiking neural networks through the information bottleneck. In *Advances in Neural Information Processing Systems*, 2021.

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

[121] David Stavens, Gabriel Hoffmann, and Sebastian Thrun. Online speed adaptation using supervised learning for high-speed, off-road autonomous driving. In *International Joint Conference on Artifical Intelligence*, pages 2218–2224, 2007.

[122] Kenneth Stewart, Andreea Danielescu, Lazar Supic, Timothy Shea, and Emre Neftci. Gesture similarity analysis on event data using a hybrid guided variational auto encoder. *arXiv preprint arXiv:2104.00165*, 2021.

[123] Kenneth Stewart and Emre Neftci. Meta-learning spiking neural networks with surrogate gradient descent. *arXiv preprint arXiv:2201.10777*, 2022.

[124] Kenneth Stewart, Garrick Orchard, Sumit Bam Shrestha, and Emre Neftci. On-chip few-shot learning with surrogate gradient descent on a neuromorphic processor. In *IEEE International Conference on Artificial Intelligence Circuits and Systems*, pages 223–227. IEEE, 2020.

[125] Kenneth Stewart, Garrick Orchard, Sumit Bam Shrestha, and Emre Neftci. Online few-shot gesture learning on a neuromorphic processor. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 10(4):512–521, 2020.

[126] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, second edition: An Introduction*. MIT Press, Cambridge, MA, USA, 2018.

[127] Amirhossein Tavanaei and Anthony S Maida. Studying the interaction of a hidden markov model with a bayesian spiking neural network. In *International Workshop on Machine Learning for Signal Processing*, pages 1–6. IEEE, 2015.

[128] Simon Thorpe, Arnaud Delorme, and Rufin Van Rullen. Spike-based strategies for rapid processing. *Neural Networks*, 14(6):715–725, 2001.

[129] Mark CW van Rossum. A novel spike distance. *Neural Computation*, 13(4):751–763, 2001.

[130] Eleni Vasilaki, Nicolas Frémaux, Robert Urbanczik, Walter Senn, and Wulfram Gerstner. Spike-based reinforcement learning in continuous state and action space: when policy gradient methods fail. *PLoS Computational Biology*, 5(12):e1000586, 2009.

[131] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016.

[132] Dongzi Wang, Bo Ding, and Dawei Feng. Meta reinforcement learning with generative adversarial reward from expert knowledge. In *IEEE International Conference on Information Systems and Computer Aided Education*, pages 1–7. IEEE, 2020.

[133] Alison I. Weber and Jonathan W. Pillow. Capturing the Dynamical Repertoire of Single Neurons with Generalized Linear Models. *Neural Computation*, 29(12):3260–3289, 2017.

[134] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in Neuroscience*, 12, 2018.

[135] Zheyu Yang, Yujie Wu, Guanrui Wang, Yukuan Yang, Guoqi Li, Lei Deng, Jun Zhu, and Luping Shi. Dashnet: A hybrid artificial and spiking neural network for high-speed object tracking. *arXiv preprint arXiv:1909.12942*, 2019.

[136] Jaesik Yoon, Taesup Kim, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn. Bayesian model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[137] Jinsung Yoon, Daniel Jarrett, and Mihaela Van der Schaar. Time-series generative adversarial networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[138] Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. *Neural Computation*, 30(6):1514–1541, 2018.

[139] Wenrui Zhang and Peng Li. Information-theoretic intrinsic plasticity for online unsupervised learning in spiking neural networks. *Frontiers in Neuroscience*, 13:31, 2019.

[140] Nan Zheng and Pinaki Mazumder. Hardware-friendly actor-critic reinforcement learning through modulation of spike-timing-dependent plasticity. *IEEE Transactions on Computers*, 66(2):299–311, 2017.

[141] Yajing Zheng, Shanshan Jia, Zhaofei Yu, Tiejun Huang, Jian K Liu, and Yonghong Tian. Probabilistic inference of binary markov random fields in spiking neural networks through mean-field approximation. *Neural Networks*, 2020.

[142] Fengwei Zhou, Bin Wu, and Zhenguo Li. Deep meta-learning: Learning to learn in the concept space. *arXiv preprint arXiv:1802.03596*, 2018.

[143] Chenglong Zou, Xiaoxin Cui, Yisong Kuang, Kefei Liu, Yuan Wang, Xinan Wang, and Ru Huang. A scatter-and-gather spiking convolutional neural network on a reconfigurable neuromorphic hardware. *Frontiers in Neuroscience*, 15, 2021.