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

PROBABILISTIC SPIKING NEURAL NETWORKS: SUPERVISED, UNSUPERVISED AND ADVERSARIAL TRAININGS

by
Alireza Bagheri

Spiking Neural Networks (SNNs), or third-generation neural networks, are networks of computation units, called neurons, in which each neuron with internal analogue dynamics receives as input and produces as output spiking, that is, binary sparse, signals. In contrast, second-generation neural networks, termed as Artificial Neural Networks (ANNs), rely on simple static non-linear neurons that are known to be energy-intensive, hindering their implementations on energy-limited processors such as mobile devices. The sparse event-based characteristics of SNNs for information transmission and encoding have made them more feasible for highly energy-efficient neuromorphic computing architectures. The most existing training algorithms for SNNs are based on deterministic spiking neurons that limit their flexibility and expressive power. Moreover, the SNNs are typically trained based on the back-propagation method, which unlike ANNs, it becomes challenging due to the non-differentiability nature of the spike dynamics. Considering these two key issues, this dissertation is devoted to develop probabilistic frameworks for SNNs that are tailored to the solution of supervised and unsupervised cognitive tasks. The SNNs utilize rich model, flexible and computationally tractable properties of Generalized Linear Model (GLM) neuron. The GLM is a probabilistic neural model that was previously considered within the computational neuroscience literature. A novel training method is proposed for the purpose of classification with a first-to-spike decoding rule, whereby the SNN can perform an early classification decision once spike firing is detected at an output neuron. This method is in contrast with conventional classification rules for SNNs that operate offline based on the number of

output spikes at each output neuron. As a result, the proposed method improves the accuracy-inference complexity trade-off with respect to conventional decoding. For the first time in the field, the sensitivity of SNNs trained via Maximum Likelihood (ML) is studied under white-box adversarial attacks. Rate and time encoding, as well as rate and first-to-spike decoding, are considered. Furthermore, a robust training mechanism is proposed that is demonstrated to enhance the resilience of SNNs under adversarial examples. Finally, unsupervised training task for probabilistic SNNs is studied. Under generative model framework, multi-layers SNNs are designed for both encoding and generative parts. In order to train the Variational Autoencoders (VAEs), the standard ML approach is considered. To tackle the intractable inference part, variational learning approaches including doubly stochastic gradient learning, Maximum A Posterior (MAP)-based, and Rao-Blackwellization (RB)-based are considered. The latter is referred as the Hybrid Stochastic-MAP Variational Learning (HSM-VL) scheme. The numerical results show performance improvements using the HSM-VL method compared to the other two training schemes.

PROBABILISTIC SPIKING NEURAL NETWORKS: SUPERVISED, UNSUPERVISED AND ADVERSARIAL TRAININGS

by
Alireza Bagheri

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 Electrical Engineering

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

May 2019

APPROVAL PAGE

PROBABILISTIC SPIKING NEURAL NETWORKS: SUPERVISED, UNSUPERVISED AND ADVERSARIAL TRAININGS

Alireza Bagheri

Dr. Osvaldo Simeone, Dissertation Co-advisor — Date
Professor of Electrical and Computer Engineering, NJIT

Dr. Bipin Rajendran, Dissertation Co-advisor — Date
Associate Professor of Electrical and Computer Engineering, NJIT

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

Dr. Ali Abdi, Committee Member — Date
Professor of Electrical and Computer Engineering, NJIT

Dr. Zhi Wei, Committee Member — Date
Associate Professor of Computer Science, NJIT

BIOGRAPHICAL SKETCH

Author: Alireza Bagheri

Degree: Doctor of Philosophy

Date: May 2019

Undergraduate and Graduate Education:

- Doctor of Philosophy in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2019
- Master of Science in Electrical Engineering,
  Semnan University, Semnan, Semnan, Iran, 2013
- Bachelor of Science in Electrical Engineering,
  Islamic Azad University, Karaj, Alborz, Iran, 2009

Major: Electrical Engineering

Presentations and Publications:

A. Bagheri, O. Simeone, and B. Rajendran, "Adversarial Training for Probabilistic Spiking Neural Networks," *IEEE Int. Wksh. Signal Process. Adv. Wireless Commun. (SPAWC)*, June 2018.

A. Bagheri, O. Simeone, and B. Rajendran, "Training Probabilistic Spiking Neural Networks with First-to-spike Decoding," *IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, Apr. 2018.

A. AL-Shuwaili, O. Simeone, **A. Bagheri** and G. Scutari, "Joint Uplink/Downlink Optimization for Backhaul-Limited Mobile Cloud Computing with User Scheduling," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 3, issue 4, Dec. 2017.

A. N. Al-Shuwaili, **A. Bagheri** and O. Simeone, "Joint uplink/downlink and offloading optimization for mobile cloud computing with limited backhaul," *IEEE Conf. Inf. Sci. Syst. (CISS)*, Mar. 2016.

Dedicated to my family and friends.

ACKNOWLEDGMENT

I am very lucky to have so many people to thank. First and foremost, I would like to express my sincere gratitude to my advisors Prof. Osvaldo Simeone and Prof. Bipin Rajendran, for all of their guidances, supports and inspirations throughout my PhD studies. None of this would have been possible without their helps.

Second, I would like to thank my committee members Prof. Alexander Haimovich, Prof. Ali Abdi, and Prof. Zhi Wei for spending their valuable time and constructive comments on my dissertation.

Third, I would like to express my thanks to my fellow group members who either through discussions or collaborations have helped me in my graduate career. Amongst many others, this includes Ali Najdi Al-Shuwaili, Shruti R Kulkarni, Anakha V Babu, Bleema Rosenfeld, and Sarah Ali Obead.

Fourth, I would like to thank Ms. Kathleen Bosco and Ms. Angela Retino for always being kind and ready to help me at the Elisha Yegal Bar-Ness Center for Wireless Information Processing (CWiP), New Jersey Institute of Technology (NJIT).

Fifth, I would like to thank Dr. Durgamadhab (Durga) Misra, Ms. Teri Bass, Ms. Clarisa Gonzalez-Lenahan and all the staff of the Department of Electrical and Computer Engineering (ECE), the Office of Graduate Studies, and the Office of Global Initiatives (OGI) at NJIT for their help and support with administrative matters during my PhD studies.

Sixth, I would like to have special thanks on NSF, Ross and PhoneTel Fellowships that entirely supported my PhD studies. My research works have supported by the U.S. NSF under grants 1525629 and ECCS #1710009. O. Simeone has also received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation program (grant agreement #725731).

Last but not least, I am deeply thankful to my parents, family and friends for their constant support and encouragement, for always being in touch with me.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

MOTIVATION AND OVERVIEW

In this dissertation, we develop biology-inspired learning methods for probabilistic neural networks. The overall research goal is the establishment of a theoretical framework to enable the design of flexible spike-domain learning algorithms that are tailored to the solution of supervised and unsupervised cognitive tasks. This work has centered around developing probabilistic frameworks for energy-efficient learning and inference on third generation of neural networks, also referred to Spiking Neural Networks (SNNs). SNNs are networks of computation units, called neurons, in which each neuron with internal analogue dynamics receives as input and produces as output spiking, that is, binary sparse, signals. The sparse event-driven characteristics of SNNs have led them more feasible for highly energy-efficient neuromorphic computing architectures. Some examples of such hardware implementations are the Loihi from Intel, TrueNorth from IBM, SpiNNaker from the University of Manchester. We refer to [2] for an overview of existing neuromorphic architectures and applications.

1.1 Organization and Contributions

In this section, the organization and contributions of the dissertation are outlined.

Chapter 2: In this chapter, the problem of training a two-layer SNN under a probabilistic neuron model is studied, for the purpose of classification. We use the flexible and computationally tractable Generalized Linear Model (GLM) that was introduced in the context of computational neuroscience. Conventional decoding in SNNs operates offline by selecting the output neuron, and hence the corresponding class, with the largest number of output spikes. In contrast, we study here a first-to-spike decoding rule, whereby the SNN can perform an early classification decision once a spike firing is detected at an output neuron. This generally reduces decision

latency and complexity during the inference phase. We have demonstrated that the proposed method improves the accuracy-inference complexity trade-off with respect to conventional decoding. The material in this chapter has been reported in [3].

Chapter 3: Classifiers trained using conventional empirical risk minimization or maximum likelihood methods are known to suffer dramatic performance degradations when tested over examples adversarially selected based on knowledge of the classifier's decision rule. Due to the prominence of Artificial Neural Networks (ANNs) as classifiers, their sensitivity to adversarial examples, as well as robust training schemes, have been recently the subject of intense investigation. In this chapter, for the first time, the sensitivity of SNNs, or third-generation neural networks, to adversarial examples is studied. The study considers rate and time encoding, as well as rate and first-to-spike decoding. Furthermore, a robust training mechanism is proposed that is demonstrated to enhance the performance of SNNs under white-box attacks. The material in this chapter has been reported in [4].

Chapter 4: This chapter presents unsupervised training for probabilistic SNNs based on GLM neurons. We consider the problem of training a Variational Autoencoder (VAE) in which the encoder and decoder probabilistic mappings are both modeled with multiple hidden layers of SNNs. In order to capture the best representation of data, we follow generative model framework. In this chapter, for the first time, the Maximum A Posterior (MAP) and Hybrid Stochastic-MAP Variational Learning (HSM-VL) training techniques developed for SNNs. Numerical results present performance gains using the HSM-VL scheme for both feature learning and data representation tasks.

CHAPTER 2

TRAINING PROBABILISTIC SPIKING NEURAL NETWORKS WITH FIRST-TO-SPIKE DECODING

2.1 Introduction

Most current machine learning methods rely on second-generation neural networks, which consist of simple static non-linear neurons. In contrast, neurons in the human brain are known to communicate by means of sparse spiking processes. As a result, they are mostly inactive, and energy is consumed sporadically. Third-generation neural networks, or Spiking Neural Networks (SNNs), aim at harnessing the energy efficiency of spike-domain processing by building on computing elements that operate on, and exchange, spikes [5]. SNNs can be natively implemented on neuromorphic chips that are currently being developed within academic projects and by major chip manufacturers. Proof-of-concept implementations have shown remarkable energy savings by multiple orders of magnitude with respect to second-generation neural networks (see, e.g., [6, 7]).

Notwithstanding the potential of SNNs, a significant stumbling block to their adoption is the dearth of flexible and effective learning algorithms. Most existing algorithms are based on variations of the unsupervised mechanism of Spike-Timing Dependent Plasticity (STDP), which updates synaptic weights based on local input and output spikes, and supervised variations that leverage global feedback [8, 9]. Another common approach is to convert trained second-generation networks to SNNs [10, 11]. Among the learning methods that attempt to directly maximize a spike-domain performance criterion, most techniques assume deterministic Spike Response Model (SRM) neurons, and propose various approximations to cope with the non-differentiability of the neurons' outputs (see [12, 13] and references therein).

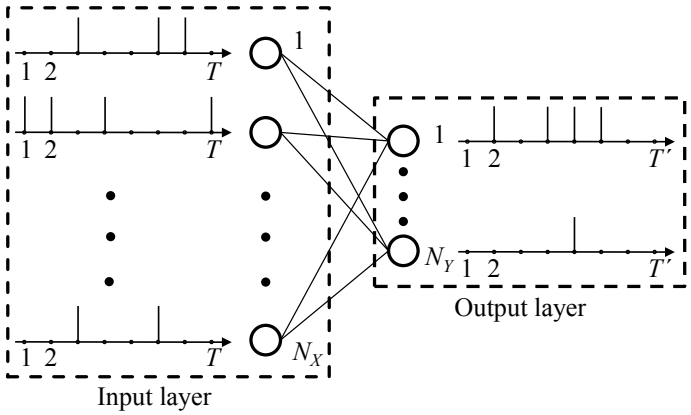


Figure 2.1 Two-layer SNN for supervised learning.

While the use of probabilistic models for spiking neurons is standard in the context of computational neuroscience (see, e.g., [14]), probabilistic modeling has been sparsely considered in the machine learning literature on SNNs. This is despite the known increased flexibility and expressive power of probabilistic models [15, 16]. In the context of SNNs, as an example, probabilistic models have the capability of learning firing thresholds using standard gradient based methods, while in deterministic models these are instead treated as hyperparameters and set by using heuristic mechanisms such as homeostasis [17]. The state of the art on supervised learning with probabilistic models is set by [18] that considers Stochastic Gradient Descent (SGD) for Maximum Likelihood (ML) training, under the assumption that there exist given desired output spike trains for all output neurons.

In this chapter, we study the problem of training the two-layer SNN illustrated in Figure 2.1 under a probabilistic neuron model, for the purpose of classification. Conventional decoding in SNNs operates offline by selecting the output neuron, and hence the corresponding class, with the largest number of output spikes [18]. In contrast, here we study a first-to-spike decoding rule, whereby the SNN can perform an early classification decision once a spike firing is detected at an output neuron. This generally reduces decision latency and complexity during the inference phase. The first-to-spike decision method has been investigated with temporal, rather than

rate, coding and deterministic neurons in [19, 20, 21, 22], but no learning algorithm exists under probabilistic neural models.

To fill this gap, we first propose the use of the flexible and computationally tractable Generalized Linear Model (GLM) that was introduced in [1] in the context of computational neuroscience (Section 2.3). Under this model, we then derive a novel SGD-based learning algorithm that maximizes the likelihood that the first spike is observed at the correct output neuron (Section 2.4). Finally, we present numerical results that bring insights into the optimal parameter selection for the GLM neuron and on the accuracy-complexity trade-off performance of conventional and first-to-spike decoding rules.

2.2 Spiking Neural Network with GLM Neurons

In this section, we describe the architecture of the two-layer SNN under study and then we present the proposed GLM neuron model.

2.2.1 Architecture

We consider the problem of classification using a two-layer SNN. As shown in Figure 2.1, the SNN is fully connected and has N_X presynaptic neurons in the input, or sensory layer, and N_Y neurons in the output layer. Each output neuron is associated with a class. In order to feed the SNN, an input example, e.g., a gray scale image, is converted to a set of N_X discrete-time spike trains, each with T samples, through rate encoding. The input spike trains are fed to the N_Y postsynaptic GLM neurons, which output discrete-time spike trains. A decoder then selects the image class on the basis of the spike trains emitted by the output neurons.

2.2.2 Rate Encoding

With the conventional rate encoding method, each entry of the input signal, e.g., each pixel for images, is converted into a discrete-time spike train by generating an

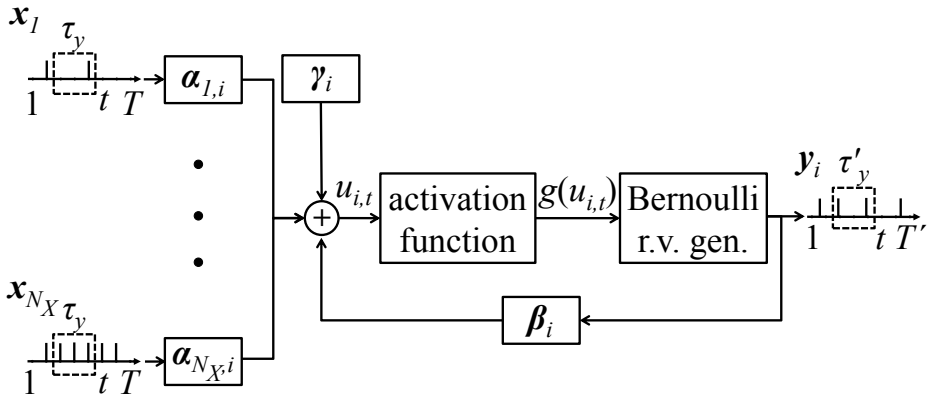


Figure 2.2 GLM neuron model.

independent and identically distributed (i.i.d.) Bernoulli vectors. The probability of generating a "1", i.e., a spike, is proportional to the value of the entry. In the experiments in Section 2.5, we use gray scale images with pixel intensities between 0 and 255 that yield a spike probability between 0 and 1/2.

2.2.3 GLM Neuron Model

The relationship between the input spike trains from the N_X presynaptic neurons and the output spike train of any postsynaptic neuron i follows a GLM, as illustrated in Figure 2.2. To elaborate, we denote as $x_{j,t}$ and $y_{i,t}$ the binary signal emitted by the j-th presynaptic and the i-th postsynaptic neurons, respectively, at time t. Also, we let $\mathbf{x}_{j,a}^b = (x_{j,a}, ..., x_{j,b})$ be the vector of samples from spiking process of the presynaptic neuron j in the time interval $[a, b]$. Similarly, the vector $\mathbf{y}_{i,a}^b = (y_{i,a}, ..., y_{i,b})$ contains samples from the spiking process of the neuron i in the interval $[a, b]$. As seen in Figure 2.2, the output $y_{i,t}$ of postsynaptic neuron i at time t is Bernoulli distributed, with firing probability that depends on the past spiking behaviors $\{\mathbf{x}_{j,t-\tau_y}^{t-1}\}$ of the presynaptic neurons $j = 1, ..., N_X$ in a window of duration τ_y samples, as well as on the past spike timings $\mathbf{y}_{i,t-\tau'_y}^{t-1}$ of neuron i in a window of duration τ'_y samples. Mathematically, the membrane potential of postsynaptic neuron i at time t is given

by

$$u_{i,t} = \sum_{j=1}^{N_X} \boldsymbol{\alpha}_{j,i}^T \mathbf{x}_{j,t-\tau_y}^{t-1} + \boldsymbol{\beta}_i^T \mathbf{y}_{i,t-\tau_y'}^{t-1} + \gamma_i, \tag{2.1}$$

where $\boldsymbol{\alpha}_{j,i} \in \mathbb{R}^{\tau_y}$ is a vector that defines the *Synaptic Kernel* (SK) applied on the $\{j,i\}$ synapse between presynaptic neuron j and postsynaptic neuron i; $\boldsymbol{\beta}_i \in \mathbb{R}^{\tau_y'}$ is the *Feedback Kernel* (FK); and γ_i is a bias parameter. The vector of variable parameters $\boldsymbol{\theta}_i$ includes the bias γ_i and the parameters that define the SK and FK filters, which are discussed below. Accordingly, the log-probability of the entire spike train $\mathbf{y}_i = [y_{i,1}, ..., y_{i,T}]^T$ conditioned on the input spike trains $\mathbf{x} = \{\mathbf{x}_j\}_{j=1}^{N_X}$ can be written as

$$\log p_{\boldsymbol{\theta}_i}(\mathbf{y}_i \,|\mathbf{x}) = \sum_{t=1}^{T} [y_{i,t} \log g\left(u_{i,t}\right) + \bar{y}_{i,t} \log \bar{g}\left(u_{i,t}\right)], \tag{2.2}$$

where $g(\cdot)$ is an activation function, such as the sigmoid function $g(x) = \sigma(x) = 1/\left(1+\exp(-x)\right)$, and we defined $\bar{y}_{i,t} = 1 - y_{i,t}$ and $\bar{g}\left(u_{i,t}\right) = 1 - g\left(u_{i,t}\right)$.

Unlike prior work on SNNs with GLM neurons, we adopt here the parameterized model introduced in [1] in the field of computational neuroscience. Accordingly, the SK and FK filters are parameterized as the sum of fixed basis functions with learnable weights. To elaborate, we write the SK $\boldsymbol{\alpha}_{j,i}$ and the FK $\boldsymbol{\beta}_i$ as

$$\boldsymbol{\alpha}_{j,i} = \mathbf{A}\mathbf{w}_{j,i}, \text{ and } \boldsymbol{\beta}_i = \mathbf{B}\mathbf{v}_i, \tag{2.3}$$

respectively, where we have defined the matrices $\mathbf{A} = [\mathbf{a}_1, ..., \mathbf{a}_{K_{\boldsymbol{\alpha}}}]$ and $\mathbf{B} = \left[\mathbf{b}_1, ..., \mathbf{b}_{K_{\boldsymbol{\beta}}}\right]$ and the vectors $\mathbf{w}_{j,i} = [w_{j,i,1}, ..., w_{j,i,K_{\boldsymbol{\alpha}}}]^T$ and $\mathbf{v}_i = \left[v_{i,1}, ..., v_{i,K_{\boldsymbol{\beta}}}\right]^T$; $K_{\boldsymbol{\alpha}}$ and $K_{\boldsymbol{\beta}}$ denote the respective number of basis functions; $\mathbf{a}_k = \left[a_{k,1}, ..., a_{k,\tau_y}\right]^T$ and $\mathbf{b}_k = \left[b_{k,1}, ..., b_{k,\tau_y'}\right]^T$ are the basis vectors; and $\{w_{j,i,k}\}$ and $\{v_{i,k}\}$ are the learnable weights for the kernels $\boldsymbol{\alpha}_{j,i}$ and $\boldsymbol{\beta}_i$, respectively. This parameterization generalizes previously studied models for machine learning application. For instance, as a special case, if we set $K_{\boldsymbol{\alpha}} = K_{\boldsymbol{\beta}} = 1$, set $\mathbf{a}_1$ and $\mathbf{b}_1$ as in [18, equations (4) and (5)], and

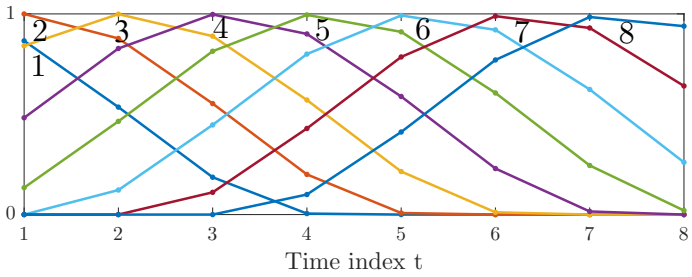


Figure 2.3 Basis functions used in Section 2.5 ($a = 7500$ and $c = 1$ in [1, Section Methods]).

fix the weights $v_{i,1} = 1$, equation (2.2) yields a discrete-time approximation of the model considered in [18]. As another example, if we set $K_{\boldsymbol{\alpha}} = \tau_y$, $K_{\boldsymbol{\beta}} = \tau_y'$, $\mathbf{a}_k = \mathbf{1}_k$, $\mathbf{b}_k = \mathbf{1}_k$, where $\mathbf{1}_k$ is the all-zero vector except for a one in position k, equation (2.1) yields the unstructured GLM model considered in [23]. For the experiments discussed in Section 2.5, we adopt the time-localized raised cosine basis functions introduced in [1], which are illustrated in Figure 2.3. Note that this model is flexible enough to include the learning of synaptic delays [24, 25].

2.3 Training with Conventional Decoding

In this section, we briefly review ML training based on conventional rate decoding for the two-layer SNN. During the inference phase, decoding is conventionally carried out in post-processing by selecting the output neuron with the largest number of spikes. In order to facilitate the success of this decoding rule, in the training phase, the postsynaptic neuron corresponding to the correct label $c \in \{1, ..., N_Y\}$ is typically assigned a desired output spike train $\mathbf{y}_c$ with a number of spikes, while a zero output is assigned to the other postsynaptic neurons $\mathbf{y}_i$ with $i \neq c$.

Using the ML criterion, one hence maximizes the sum of the log-probabilities equation (2.2) of the desired output spikes $\mathbf{y}(c) = \{\mathbf{y}_1(c), ..., \mathbf{y}_{N_Y}(c)\}$ for the correct label c given the N_X input spike trains $\mathbf{x} = \{\mathbf{x}_1, ..., \mathbf{x}_{N_X}\}$, i.e.,

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{N_Y} \log p_{\boldsymbol{\theta}_i}(\mathbf{y}_i(c) | \mathbf{x}). \tag{2.4}$$

The sum is further extended to all examples in the training set. The parameter vector $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{V}, \boldsymbol{\gamma}\}$ includes the parameters $\mathbf{W} = \{\mathbf{W}_i\}_{i=1}^{N_Y}$, $\mathbf{V} = \{\mathbf{v}_i\}_{i=1}^{N_Y}$ and $\boldsymbol{\gamma} = \{\gamma_i\}_{i=1}^{N_Y}$. The negative log-likelihood $-L(\boldsymbol{\theta})$ is convex with respect to $\boldsymbol{\theta}$ and can be minimized via SGD. For a given input $\mathbf{x}$, the gradients of the log-likelihood function $L(\boldsymbol{\theta})$ in equation (2.4) for conventional decoding are given as

$$\nabla_{\boldsymbol{w}_{j,i}} L(\boldsymbol{\theta}) = \sum_{t=1}^{T} e_{i,t} \rho_{i,t} \mathbf{A}^T \mathbf{x}_{j,t-\tau_y}^{t-1}, \tag{2.5}$$

$$\nabla_{\boldsymbol{v}_i} L(\boldsymbol{\theta}) = \sum_{t=1}^{T} e_{i,t} \rho_{i,t} \mathbf{B}^T \mathbf{y}_{i,t-\tau_y'}^{t-1}, \tag{2.6}$$

and

$$\nabla_{\gamma_i} L(\boldsymbol{\theta}) = \sum_{t=1}^{T} e_{i,t} \rho_{i,t}, \tag{2.7}$$

where

$$e_{i,t} = y_{i,t} - g(u_{i,t}), \tag{2.8}$$

is the error signal, and $\rho_{i,t}$ is given as

$$\rho_{i,t} = \frac{g'(u_{i,t})}{g(u_{i,t})\, \bar{g}(u_{i,t})}, \tag{2.9}$$

where $g'(u_{i,t}) \triangleq \frac{dg(u_{i,t})}{du_{i,t}}$.

2.4 Training with First-to-spike Decoding

In this section, we introduce the proposed learning approach based on GLM neurons and first-to-spike decoding.

During the inference phase, with first-to-spike decoding, a decision is made once a first spike is observed at an output neuron. In order to train the SNN for this classification rule, we propose to follow the ML criterion by maximizing the probability to have the first spike at the output neuron corresponding to the correct

label c. The logarithm of this probability for a given example $\mathbf{x}$ can be written as

$$L(\boldsymbol{\theta}) = \log\left(\sum_{t=1}^{T} p_t(\boldsymbol{\theta})\right), \tag{2.10}$$

where

$$p_t(\boldsymbol{\theta}) = \prod_{i=1, i\neq c}^{N_Y} \prod_{t'=1}^{t} \bar{g}(u_{i,t'}) g(u_{c,t}) \prod_{t'=1}^{t-1} \bar{g}(u_{c,t'}), \tag{2.11}$$

is the probability of having the first spike at the correct neuron c at time t. In equation (2.11), the potential $u_{i,t}$ for all i is obtained from equation (2.1) by setting $y_{i,t} = 0$ for all i and t. The log-likelihood function $L(\boldsymbol{\theta})$ in equation (2.10) is not concave, and we tackle its maximization via SGD.

To this end, the gradients of the log-likelihood function for a given input $\mathbf{x}$ can be computed after some algebra as (see Appendix A for details)

$$\begin{aligned} &\nabla_{\mathbf{w}_{j,i}} L(\boldsymbol{\theta}) \\ &= \begin{cases} -\sum_{t=1}^{T} \rho_{i,t} h_t g(u_{i,t}) \mathbf{A}^T \mathbf{x}_{j,t-\tau_y}^{t-1} & i \neq c \\ -\sum_{t=1}^{T} \rho_{c,t} \left(h_t g(u_{c,t}) - q_t\right) \mathbf{A}^T \mathbf{x}_{j,t-\tau_y}^{t-1} & i = c \end{cases}, \end{aligned} \tag{2.12}$$

for the weights and

$$\nabla_{\gamma_i} L(\boldsymbol{\theta}) = \begin{cases} -\sum_{t=1}^{T} \rho_{i,t} h_t g(u_{i,t}) & i \neq c \\ -\sum_{t=1}^{T} \rho_{c,t} \left(h_t g(u_{c,t}) - q_t\right) & i = c \end{cases}, \tag{2.13}$$

for the biases, where we have defined

$$\rho_{i,t} = \frac{g'(u_{i,t})}{g(u_{i,t})\, \bar{g}(u_{i,t})}, \tag{2.14}$$

and

$$h_t = \sum_{t'=t}^{T} q_{t'} = 1 - \sum_{t'=1}^{t-1} q_{t'}, \tag{2.15}$$

with

$$q_t = \frac{p_t(\boldsymbol{\theta})}{\sum_{t'=1}^{T} p_{t'}(\boldsymbol{\theta})}. \tag{2.16}$$

Note that we have $\rho_{i,t} = 1$ when g is the sigmoid function.

Based on equation (2.12), the resulting SGD update can be considered as a neo-Hebbian rule [26], since it multiplies the contributions of the presynaptic neurons and of the postsynaptic activity, where the former depends on $\mathbf{x}$ and the latter on the potential $u_{i,t}$. Furthermore, in equation (2.12)−equation (2.13), the probabilities $g(u_{i,t})$ and $g(u_{c,t})$ of firing at time t are weighted by the probability h_t in equation (2.15). By equation (2.16), this is the probability that the correct neuron is the first to spike and that it fires at some time $t' \geq t$, given that it is the first to spike at some time in the interval $[1, 2, ..., T]$.

As a practical note, in order to avoid vanishing values in calculating the weights equation (2.16), we compute each probability term $p_t(\boldsymbol{\theta})$ in the log-domain, and normalize all the resulting terms with respect to the minimum probability as $q_t = \exp(a_t) / \sum_{t'=1}^{T} \exp(a_{t'})$, where $a_t = \ln(p_t) - \min_t(\ln(p_t))$.

2.5 Numerical Results

In this section, we numerically study the performance of the probabilistic SNN in Figure 2.1 under conventional and first-to-spike decoding rules. We use the standard MNIST dataset [27] as the input data. As a result, we have $N_X = 784$, with one input neuron per pixel of the $\mathbf{x}$ images. Following [7], we consider different number of classes, or digits, namely, the two digits $\{5, 7\}$, the four digits $\{5, 7, 1, 9\}$ and all 10 digits $\{0, ..., 9\}$, and we use 1000 samples of each class for training and the same number for test set. We use a desired spike train with one spike after every three zeros for training the conventional decoding. SGD with minibatch size of one with 200 training epochs is used for both schemes. Ten-fold cross-validation is applied for

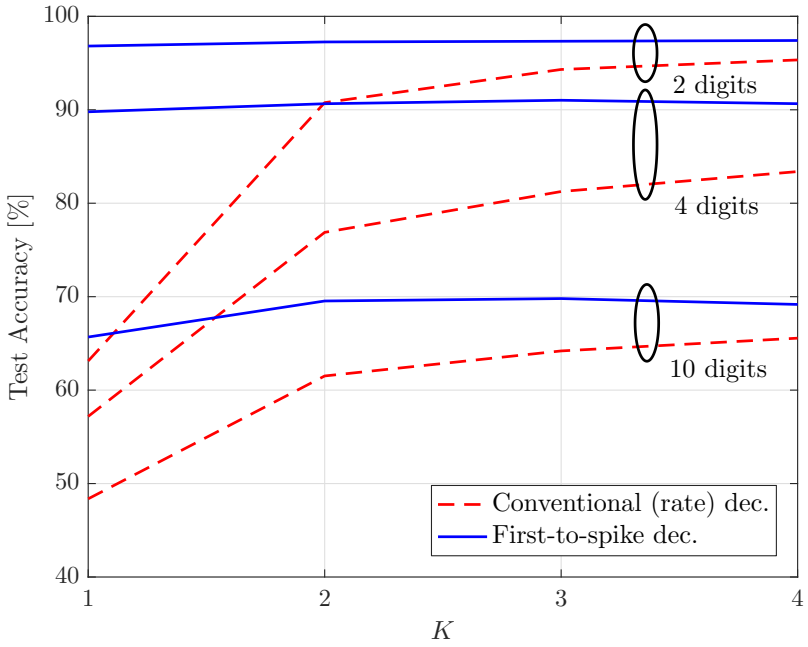


Figure 2.4 Test accuracy versus the number K of basis functions for both conventional (rate) and first-to-spike decoding rules when $T = 4$.

selecting between 10^{-3} or 10^{-4} for the constant learning rates. The model parameters $\boldsymbol{\theta}$ are randomly initialized with uniform distribution between -1 and 1.

We evaluate the performance of the schemes in terms of classification accuracy in the test set and of inference complexity. The inference complexity is measured by the total number of elementary operations, namely additions and multiplications, for input image that are required by the SNN during inference. The number of arithmetic operations needed to calculate the membrane potential equation (2.1) of neuron i at time instant t is of the order of $\mathcal{O}\left(N_X \tau_y + \tau_y'\right)$. As a result, in the conventional decoding method, the inference complexity per output neuron, or per class, is of the order $\mathcal{O}\left(T\left(N_X s_{\mathbf{x}} + s_{\mathbf{y}}\right)\right)$, where $s_{\mathbf{x}}$ and $s_{\mathbf{y}}$ are the fraction of spikes in $\mathbf{x}$ and $\mathbf{y}$, respectively. In contrast, with the first-to-spike decoding rule, the SNN can perform an early decision once a single spike is detected, and hence its complexity order is $\mathcal{O}\left(t\left(N_X s_{\mathbf{x}} + s_{\mathbf{y}}\right)\right)$, where $1 \leq t \leq T$ is the (random) decision time.

We first consider the test classification accuracy as a function of the number K of basis functions in the GLM neural model. The basis functions are numbered as in

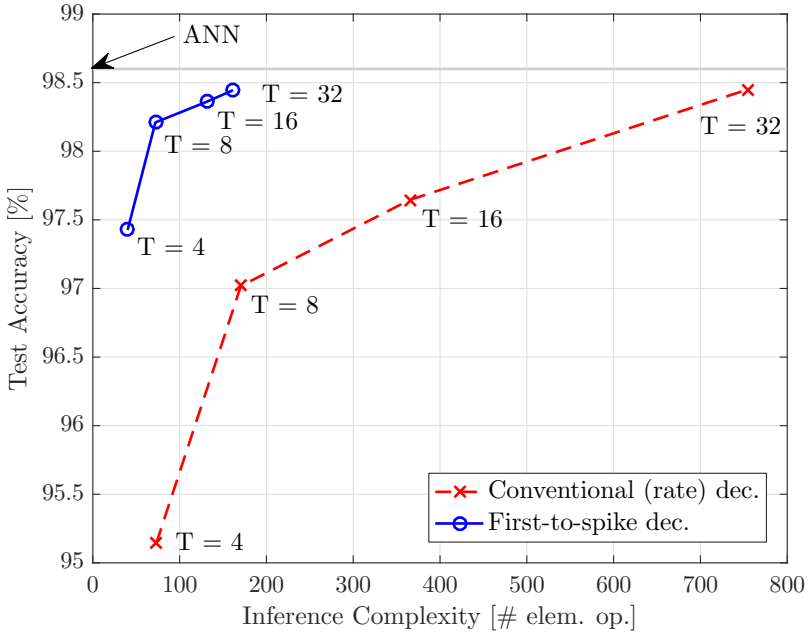


Figure 2.5 Test accuracy versus per-class inference complexity for both conventional (rate) and first-to-spike decoding rules.

Figure 2.3, and we set $T = 4$. From Figure 2.4, we observe that conventional decoding requires a large number K in order to obtain its best accuracy. This is due to the need to ensure that the correct output neuron fires consistently more than the other neurons in response to the input spikes. This, in turn, requires a larger temporal reception field, i.e., a larger K, to be sensitive to the randomly located input spikes. We note that for small values of T, such as $T = 4$, first-to-spike decoding obtains better accuracies than conventional decoding.

Figure 2.5 depicts the test classification accuracy versus the inference complexity for both conventional and first-to-spike decoding rules for two digits when $K = T$. The classification accuracy of a conventional two-layer artificial neural network (ANN) with logistic neurons is added for comparison. From the figure, first-to-spike decoding is seen to offer a significantly lower inference complexity, thanks to its capability for early decisions, without compromising the accuracy. For instance, when the classification accuracy equals to 98.4%, the complexity of the conventional decoding

method is five times larger than the first-to-spike method. Note also that conventional decoding generally requires large values of T to perform satisfactorily.

2.6 Conclusions

In this chapter, we have proposed a novel learning method for probabilistic two-layer SNN that operates according to the first-to-spike learning rule. We have demonstrated that the proposed method improves the accuracy-inference complexity trade-off with respect to conventional decoding. Additional work is needed in order to generalize the results to multi-layer networks.

CHAPTER 3

ADVERSARIAL TRAINING FOR PROBABILISTIC SPIKING NEURAL NETWORKS

3.1 Introduction

The classification accuracy of Artificial Neural Networks (ANNs) trained over large data sets from the problem domain has attained super-human levels for many tasks including image identification [28]. Nevertheless, the performance of classifiers trained using conventional empirical risk minimization or Maximum Likelihood (ML) is known to decrease dramatically when evaluated over examples adversarially selected based on knowledge of the classifier's decision rule [29]. To mitigate this problem, robust training strategies that are aware of the presence of adversarial perturbations have been shown to improve the accuracy of classifiers, including ANNs, when tested over adversarial examples [29, 30, 31].

ANNs are known to be energy-intensive, hindering their implementation on energy-limited processors such as mobile devices. Despite the recent industrial efforts around the production of more energy-efficient chips for ANNs [5], the gap between the energy efficiency of the human brain and that of ANNs remains significant [6, 32]. A promising alternative paradigm is offered by Spiking Neural Networks (SNNs), in which synaptic input and neuronal output signals are sparse asynchronous binary spike trains [5]. Unlike ANNs, SNNs are hybrid digital-analog machines that make use of the temporal dimension, not just as a neutral substrate for computing, but as a means to encode and process information [32].

Training methods for SNNs typically assume deterministic non-linear dynamic models for the spiking neurons, and are either motivated by biological plausibility, such as the spike-timing-dependent plasticity (STDP) rule [5, 8], or by an attempt to mimic the operation of ANNs and associated learning rules (see, e.g., [33] and

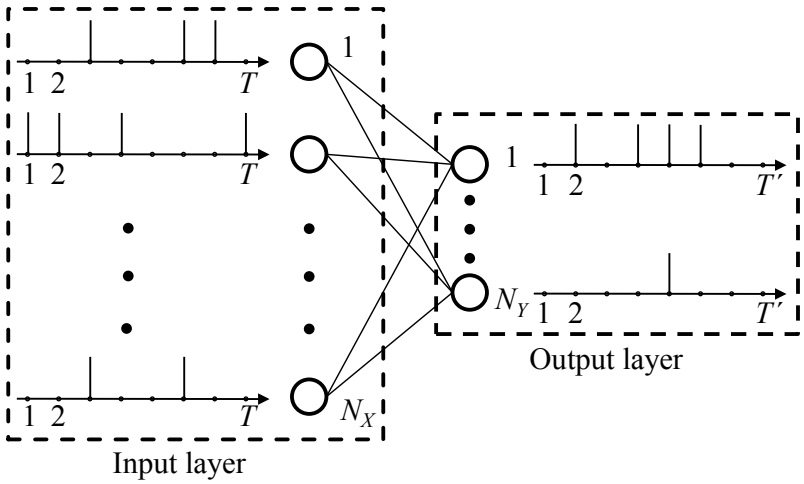


Figure 3.1 Two-layer SNN for supervised learning.

references therein). Deterministic models are known to be limited in their expressive power, especially as it pertains prior domain knowledge, uncertainty, and definition of generic queries and tasks. Training for probabilistic models of SNNs has recently been investigated in, e.g., [18, 34, 35, 3] using ML and variational inference principles.

In this dissertation, for the first time, the sensitivity of SNNs trained via ML is studied under white-box adversarial attacks, and a robust training mechanism is proposed that is demonstrated to enhance the performance of SNNs under adversarial examples. Specifically, we focus on a two-layer SNN (see Figure 3.1), and consider rate and time encoding, as well as rate and first-to-spike decoding [3]. Our results illuminate the sensitivity of SNNs to adversarial example under different encoding and decoding schemes, and the effectiveness of robust training methods.

The rest of the chapter is organized as follows. In Section 3.2, we describe the architecture of the two-layer SNN with Generalized Linear Model (GLM) neuron, as well as information encoding and decoding mechanisms. The design of adversarial perturbations is covered in Section 3.3, while a robust training is presented in Section 3.4. Section 3.5 presents numerical results, and closing remarks are given in Section 3.6.

3.2 SNN-based Classification

In this section, we introduce the classification task and the SNN architecture under study.

3.2.1 Network Architecture

We consider the problem of classification using the two-layer SNN illustrated in Figure 3.1. The SNN is fully connected and has N_X presynaptic neurons in the input, or sensory layer, and N_Y neurons in the output layer. Each output neuron is associated with a class. In order to feed the SNN, an input example, e.g., a gray scale image, is encoded into a set of N_X discrete-time spike trains, each with T samples. The input spike trains are fed to the N_Y postsynaptic GLM neurons, which output discrete-time spike trains. A decoder then selects the image class on the basis of the spike trains emitted by the output neurons.

3.2.2 Information Encoding

We consider two encoding mechanisms.

Rate encoding With the conventional rate encoding method (see, e.g., [36]), each entry of the input signal is converted into a discrete-time spike train by generating an independent and identically distributed (i.i.d.) Bernoulli vector. The probability of generating a "1", i.e., a spike, is proportional to the value of the entry. In the experiments in Section 3.5, we use gray scale images of USPS dataset with pixel intensities normalized between 0 and 1 that yield a proportional spike probability between 0 and 1/2.

Time encoding With the time encoding method, each entry of the input signal is converted into a spike train having only one spike, whose timing depends on the entry value. In particular, assuming intensity-to-latency encoding [19, 37, 36], the

spike timing in the time interval $[1, T]$ depends linearly on the entry value, such that the maximum value yields a spike at the first time sample $t = 1$, and the minimum value is mapped to a spike in the last time sample $t = T$.

GLM Neuron Model The relationship between the input spike trains from the N_X presynaptic neurons and the output spike train of any postsynaptic neuron i follows a Bernoulli GLM with canonical link function (see, e.g., [1, 3]). To elaborate, we denote as $x_{j,t}$ and $y_{i,t}$ the binary signal emitted by the j-th presynaptic and the i-th postsynaptic neurons, respectively, at time t. Also, we let $\mathbf{x}_{j,a}^{b} = (x_{j,a}, ..., x_{j,b})$ be the vector of samples from spiking process of the presynaptic neuron j in the time interval $[a, b]$. Similarly, the vector $\mathbf{y}_{i,a}^{b} = (y_{i,a}, ..., y_{i,b})$ contains samples from the spiking process of the neuron i in the interval $[a, b]$. The membrane potential of postsynaptic neuron i at time t is given by

$$u_{i,t} = \sum_{j=1}^{N_X} \boldsymbol{\alpha}_{j,i}^{T} \mathbf{x}_{j,t-\tau_y}^{t-1} + \boldsymbol{\beta}_{i}^{T} \mathbf{y}_{i,t-\tau_y'}^{t-1} + \gamma_i, \tag{3.1}$$

where $\boldsymbol{\alpha}_{j,i} \in \mathbb{R}^{\tau_y}$ is a vector that defines the *synaptic kernel* (SK) applied on the $\{j, i\}$ synapse between presynaptic neuron j and postsynaptic neuron i; $\boldsymbol{\beta}_i \in \mathbb{R}^{\tau_y'}$ is the *feedback kernel* (FK); and γ_i is a bias parameter. Note that τ_y and τ_y' denote the lengths of the SK and FK, respectively. The vector of variable parameters $\boldsymbol{\theta}_i$ includes the bias γ_i and the parameters that define the SK and FK filters, which are discussed below. According to the GLM, the log-probability of the output spike train $\mathbf{y}_i = [y_{i,1}, ..., y_{i,T}]^T$ conditioned on the input spike trains $\mathbf{x} = \{\mathbf{x}_j\}_{j=1}^{N_X}$ can be written as

$$\log p_{\boldsymbol{\theta}_i}(\mathbf{y}_i \,|\mathbf{x}) = \sum_{t=1}^{T} [y_{i,t} \log g\,(u_{i,t}) + \bar{y}_{i,t} \log \bar{g}\,(u_{i,t})], \tag{3.2}$$

where $g\,(\cdot)$ is an activation function, such as the sigmoid function $g\,(x) = \sigma\,(x) = 1/\,(1 + \exp\,(-x))$, and we defined $\bar{y}_{i,t} = 1 - y_{i,t}$ and $\bar{g}\,(u_{i,t}) = 1 - g\,(u_{i,t})$. As

per equation (3.2), each sample $y_{i,t}$ is Bernoulli distributed with spiking probability $g\left(u_{i,t}\right)$.

As in [3], we adopt the parameterized model of [1] for the SK and FK filters. Accordingly, we write the SK $\boldsymbol{\alpha}_{j,i}$ and the FK $\boldsymbol{\beta}_i$ as

$$\boldsymbol{\alpha}_{j,i} = \sum_{k=1}^{K_{\boldsymbol{\alpha}}} w_{j,i,k}\mathbf{a}_k = \mathbf{A}\mathbf{w}_{j,i}, \tag{3.3}$$

and

$$\boldsymbol{\beta}_i = \sum_{k=1}^{K_{\boldsymbol{\beta}}} v_{i,k}\mathbf{b}_k = \mathbf{B}\mathbf{v}_i, \tag{3.4}$$

respectively, where we have defined the fixed basis matrices $\mathbf{A} = [\mathbf{a}_1, ..., \mathbf{a}_{K_{\boldsymbol{\alpha}}}]$ and $\mathbf{B} = \left[\mathbf{b}_1, ..., \mathbf{b}_{K_{\boldsymbol{\beta}}}\right]$ and the vectors $\mathbf{w}_{j,i} = [w_{j,i,1}, ..., w_{j,i,K_{\boldsymbol{\alpha}}}]^T$ and $\mathbf{v}_i = \left[v_{i,1}, ..., v_{i,K_{\boldsymbol{\beta}}}\right]^T$; $K_{\boldsymbol{\alpha}}$ and $K_{\boldsymbol{\beta}}$ denote the respective number of basis functions; $\mathbf{a}_k = [a_{k,1}, ..., a_{k,\tau_y}]^T$ and $\mathbf{b}_k = [b_{k,1}, ..., b_{k,\tau_y'}]^T$ are the basis vectors; and $\{w_{j,i,k}\}$ and $\{v_{i,k}\}$ are the learnable weights for the kernels $\boldsymbol{\alpha}_{j,i}$ and $\boldsymbol{\beta}_i$, respectively. For the experiments discussed in Section 3.5, we adopt the raised cosine basis functions introduced in [1, Section Methods].

3.2.3 Information Decoding

We also consider two alternative decoding methods, namely rate decoding and first-to-spike decoding. *1) Rate decoding*: With rate decoding, decoding is carried out by selecting the output neuron with the largest number of spikes. *2) First-to-spike decoding*: With first-to-spike decoding, the class that corresponds to the neuron that spikes first is selected.

ML training: Conventional ML training is performed differently under rate and first-to-spike decoding methods, as briefly reviewed next.

1) Rate decoding: With rate decoding, the postsynaptic neuron corresponding to the correct label $c \in \{1, ..., N_Y\}$ is assigned a desired output spike train $\mathbf{y}_c$ containing a number of spikes, while an all-zero vector $\mathbf{y}_i$, $i \neq c$, is assigned to the other

postsynaptic neurons. Using the ML criterion, one hence maximizes the sum of the log-probabilities equation (2.2) of the desired output spikes $\mathbf{y}(c) = \{\mathbf{y}_1, ..., \mathbf{y}_{N_Y}\}$ for the given N_X input spike trains $\mathbf{x} = \{\mathbf{x}_1, ..., \mathbf{x}_{N_X}\}$. The log-likelihood function for a given training example $(\mathbf{x}, c)$ can be written as

$$L_{\boldsymbol{\theta}}(\mathbf{x}, c) = \sum_{i=1}^{N_Y} \log p_{\boldsymbol{\theta}_i}(\mathbf{y}_i | \mathbf{x}), \tag{3.5}$$

where the parameter vector $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{V}, \boldsymbol{\gamma}\}$ includes the parameters $\mathbf{W} = \{\mathbf{W}_i\}_{i=1}^{N_Y}$, $\mathbf{V} = \{\mathbf{v}_i\}_{i=1}^{N_Y}$ and $\boldsymbol{\gamma} = \{\gamma_i\}_{i=1}^{N_Y}$. The sum in equation (3.5) is further extended to all examples in the training set. The negative log-likelihood (NLL) $-L_{\boldsymbol{\theta}}$ is convex with respect to $\boldsymbol{\theta}$ and can be minimized via SGD [3].

2) First-to-spike decoding: With first-to-spike decoding, the class that corresponds to the neuron that spikes first is selected. The ML criterion hence maximizes the probability to have the first spike at the output neuron corresponding to the correct label. The logarithm of this probability for a given example $(\mathbf{x}, c)$ can be written as

$$L_{\boldsymbol{\theta}}(\mathbf{x}, c) = \log\left(\sum_{t=1}^{T} p_t(\boldsymbol{\theta})\right), \tag{3.6}$$

where

$$p_t(\boldsymbol{\theta}) = \prod_{i=1, i\neq c}^{N_Y} \prod_{t'=1}^{t} \bar{g}(u_{i,t'}) g(u_{c,t}) \prod_{t'=1}^{t-1} \bar{g}(u_{c,t'}), \tag{3.7}$$

is the probability of having the first spike at the correct neuron c at time t. In equation (3.7), the potential $u_{i,t}$ for all i is obtained from equation (2.1) by setting $y_{i,t} = 0$ for all i and t. The minimization of the log-likelihood function $L_{\boldsymbol{\theta}}$ in equation (3.6), which is not concave, can be tackled via SGD as proposed in [3].

3.3 Designing Adversarial Examples

In this dissertation, we consider white-box attacks based on full knowledge of the model, i.e., of the parameter vector $\boldsymbol{\theta}$, as well as of the encoding and decoding strategies. Accordingly, given an example $(\mathbf{x}, c)$, an adversarial spike train $\mathbf{x}^{\text{adv}}$ is obtained as a perturbed version of the original input $\mathbf{x}$, where the perturbation is selected so as to cause the classifier to be more likely to predict an incorrect label $c' \neq c$, while being sufficiently small.

We consider the following types of perturbations: (*i*) *Remove attack*: one or more spikes are removed from the input $\mathbf{x}$; (*ii*) *Add attack*: one or more spikes are added to the input $\mathbf{x}$; and (*iii*) *Flip attack*: one or more spikes are added or removed. The size of the disturbance is measured for all attacks by the number of spikes that are added and/or removed. Mathematically, this can be expressed as the Hamming distance

$$d_H\left(\mathbf{x}, \mathbf{x}^{\text{adv}}\right) = \sum_{j=1}^{N_X} \sum_{t=1}^{T} 1\left(x_{j,t} \neq x_{j,t}^{\text{adv}}\right), \tag{3.8}$$

where $1(\cdot)$ is the indicator function, i.e., $1(a) = 1$ if condition a is true and $1(a) = 0$ otherwise.

In order to select the adversarial perturbation of an input $\mathbf{x}$, we consider the maximization of the likelihood of a given incorrect target class $c' \neq c$. According to [38], an effective way to choose the target class c' is to find the class $c^{\text{LL}} \neq c$ that is the least likely under the given model $\boldsymbol{\theta}$. Mathematically, for a given training example $(\mathbf{x}, c)$, the least likely class is obtained by solving the problem

$$c^{\text{LL}} = \underset{c' \neq c}{\operatorname{argmin}} \quad L_{\boldsymbol{\theta}}(\mathbf{x}, c'), \tag{3.9}$$

where the log-likelihood $L_{\boldsymbol{\theta}}(\mathbf{x}, c')$ is given by equation (3.5) for rate decoding and equation (3.6) for first-to-spike decoding.

Algorithm 3.1 Greedy Design $(\boldsymbol{\theta}, T_A, \epsilon)$

Input: $\mathbf{x}$, $\boldsymbol{\theta}$, T_A, ϵ

1: Compute c^{LL} from equation (3.9)

2: Initialize: $\mathbf{x}^{\mathrm{adv}}(0) \leftarrow \mathbf{x}$

3: **for** $i = 1$ **to** $\lfloor \epsilon N_X T \rfloor$ **do**

4: $\quad \mathbf{x}^{\mathrm{adv}}(i) \leftarrow \mathbf{x}^{\mathrm{adv}}(i-1) + \mathbf{p}$, where $\mathbf{p}$ is obtained by solving problem equation (3.10) with $\mathbf{x}^{\mathrm{adv}}(i-1)$ in lieu of $\mathbf{x}$ and $p_{j,t} = 0$ for all $t > T_A$.

5: **end for**

Output: $\mathbf{x}^{\mathrm{adv}}$

Then, in order to compute the adversarial perturbation $\mathbf{p}$, we maximize the likelihood of class c^{LL} under model $\boldsymbol{\theta}$ by tackling the following optimization problem

$$\begin{aligned} \max_{\mathbf{p} \in \mathcal{C}} \quad & L_{\boldsymbol{\theta}}\left(\mathbf{x} + \mathbf{p}, c^{\mathrm{LL}}\right) \\ \text{s.t.} \quad & \|\mathbf{p}\|_0 \leq \epsilon N_X T, \end{aligned} \tag{3.10}$$

where $\|\mathbf{p}\|_0$ denotes the number of non-zero elements of $\mathbf{p}$. In equation (3.10), the perturbation $\epsilon > 0$ controls the adversary strength. In particular, the adversary is allowed to add or remove spikes from a fraction ϵ of the $N_X T$ input samples, i.e., T samples for each input neuron. The constraint set $\mathcal{C}$ in problem equation (3.10) is given by the set of binary perturbations, i.e., $\mathcal{C} = \{0, 1\}^{N_X T}$, for add attacks, since spikes can only be added; $\mathcal{C} = \{0, -1\}^{N_X T}$ for remove attacks; and $\mathcal{C} = \{0, \pm 1\}^{N_X T}$ for flip attacks.

The exact solution of problem equation (3.10) requires an exhaustive search over all possible perturbations of $\epsilon N_X T$ samples. In the worst case of flip attacks, the resulting search space is hence exponential in N_X and T. Therefore, here we resort to a greedy search method. As detailed in Algorithm 3.1, at each of the $\lfloor \epsilon N_X T \rfloor$ steps, the method looks for the best spike to add, remove or flip, depending on the attack type. We further reduce complexity by searching only among the first $T_A \leq T$ samples

Algorithm 3.2 Adversarial Training (T_A, ϵ_A)

Input: Training set, basis functions $\mathbf{A}$ and $\mathbf{B}$, learning rate η, T_A, and ϵ_A

Initialize: $\boldsymbol{\theta}$

1: **for** each iteration **do**

2: Choose example $(\mathbf{x}, c)$ from the training set

3: Compute $\mathbf{x}^{\text{adv}}$ and c^{LL} from Algorithm 3.1 with input $\boldsymbol{\theta}$, T_A and ϵ_A

4: Update $\boldsymbol{\theta}$: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \nabla_{\boldsymbol{\theta}} L_{\boldsymbol{\theta}} \left(\mathbf{x}^{\text{adv}}, c\right)$

5: **end for**

Output: $\boldsymbol{\theta}$

across all input neurons. As a results, the complexity of each step of Algorithm 3.1 is at most $N_X T_A$.

3.4 Robust Training

In order to increase the robustness of the trained SNN to adversarial examples, in this section, we propose a robust training procedure. Accordingly, in a manner similar to [31], during the SGD-based training phase, each training example $(\mathbf{x}, c)$ is substituted with the adversarial example $\mathbf{x}^{\text{adv}}$ obtained from Algorithm 3.1 for the current iterate $\boldsymbol{\theta}$. The training algorithm is detailed in Algorithm 3.2. Note that, the robust training algorithm is parameterized by T_A and ϵ_A, which determine the parameters of the assumed adversary during training.

3.5 Numerical Results

In this section, we numerically study the performance of the described probabilistic SNN under the adversarial attacks. We use the standard USPS dataset as the input data. As a result, we have $N_X = 256$, with one input neuron per pixel of the 16×16 images. Unless stated otherwise, we focus solely in the classes $\{1, 5, 7, 9\}$ and we set $T = K = 16$. We assume the worst-case $T_A = T$ for the adversary during the test

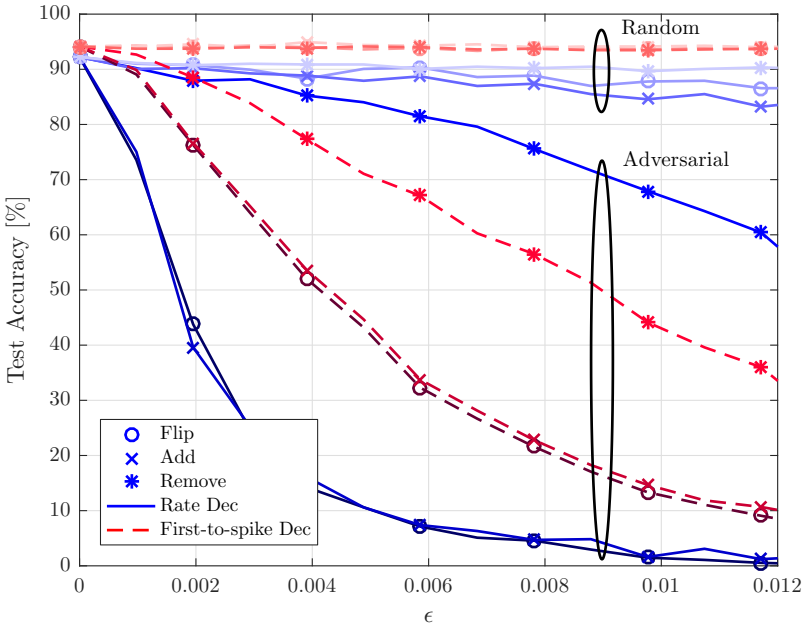


Figure 3.2 Test accuracy for ML training under adversarial and random changes versus ϵ with rate encoding for both rate and first-to-spike decoding rules ($T = K = 16$).

phase. For rate decoding, we use a desired spike train with one spike after every three zeros. SGD is applied for 200 training epochs and early stopping is used for all schemes. Holdout validation with 20% of training samples is applied to select between 10^{-3} and 10^{-4} for the constant learning rate η. The model parameters $\boldsymbol{\theta}$ are randomly initialized with uniform distribution between -1 and 1.

We first evaluate the sensitivity of different encoding and decoding schemes to adversarial examples obtained as explained in Section 3.3. For reference, we consider also perturbations obtained by randomly and uniformly adding, removing and flipping spikes. Figure 3.2 illustrates the test accuracy under adversarial and random perturbations when performing standard ML training. The accuracy is plotted versus the adversary's power ϵ assuming rate encoding and both rate and first-to-spike decoding rules. The results highlight the notable difference in performance degradation caused by random perturbations and adversarial attacks. In particular, adversarial changes can cause a significant drop in classification accuracy even with small values of ϵ, particularly when the most powerful flip attacks are used.

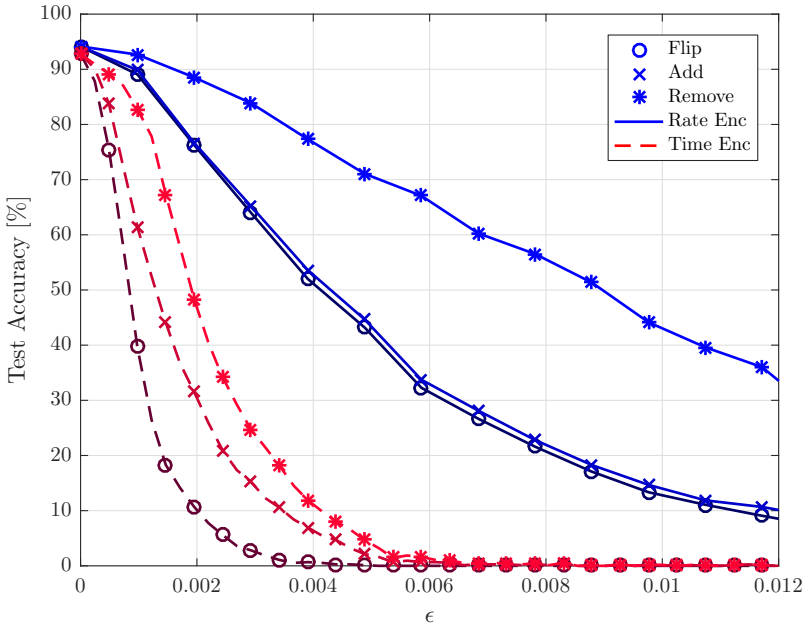


Figure 3.3 Test accuracy for ML training under adversarial attacks versus ϵ with both rate and time encoding rules for first-to-spike decoding ($T = K = 16$).

First-to-spike decoding is seen to be more resistant to add and flip attacks, while it is more vulnerable than rate decoding to remove spike attacks. The resilience of first-to-spike decoding can be interpreted as a consequence of the fact that the log-likelihood equation (3.6), unlike equation (3.5) for rate decoding, associates multiple outputs to the correct class, namely all of those with the correct neuron spiking first. Nevertheless, removing properly selected spikes can be more deleterious to first-to-spike decoding as it may prevent spiking by the correct neuron.

The comparison between rate and time encoding in terms of sensitivity to adversarial examples is considered in Figure 3.3 under the assumption of first-to-spike decoding. Time encoding is seen to be significantly less resilient than rate encoding. This is due to the fact that time encoding, in the form considered here of intensity-to-latency encoding, which associated a single spike per input neuron [36], can be easily made ineffective by removing selected spikes.

We then evaluate the impact of robust adversarial training as compared to standard ML. To this end, in Figure 3.4, we plot the test accuracy for the case of flip

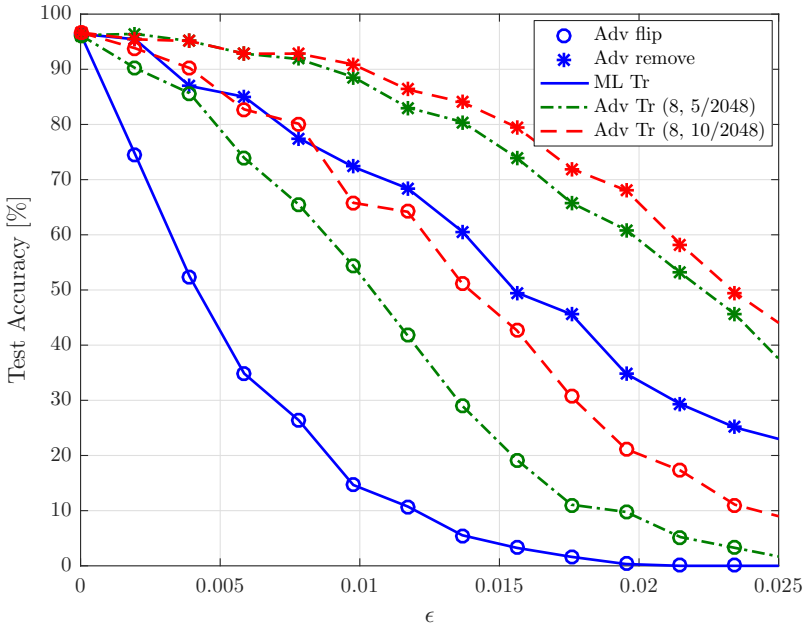


Figure 3.4 Test accuracy under adversarial attacks versus ϵ with rate encoding and rate decoding with ML and adversarial training ($T = K = 8$).

and remove attacks for both ML and adversarial training when $T = K = 8$. Here, we also focus solely on the two classes $\{5, 7\}$. We recall that the adversarial training scheme is parametrized by the time support T_A of the attacks considered during training, here $T_A = 8$, and by its power ϵ_A, here $\epsilon_A = 5/2048$ and $\epsilon_A = 10/2048$. It is observed that robust training can significantly improve the robustness of the SNN classifier, even when ϵ_A is not equal to the value ϵ used by the attacker during the test phase. Furthermore, increasing ϵ_A enhances the robustness of the trained SNN at the cost of a higher computational complexity. For instance, for an attacker in the test phase with $\epsilon = 10/2048$, i.e., with 10 bit flips, conventional ML achieves an accuracy of 45%, while adversarial training with $\epsilon_A = 10/2048$ (i.e., 10 bit flips) achieves an accuracy of 87%. The results show that the classifier remains resilient against other type of attacks, despite being trained assuming the flip attack.

Finally, under the same conditions as in Figure 3.5, we study the effect of limiting the power of the adversary assumed during training by considering $T_A = 1$ and $T_A = 8$ with the same $\epsilon_A = 5/2048$. We assume time encoding and rate

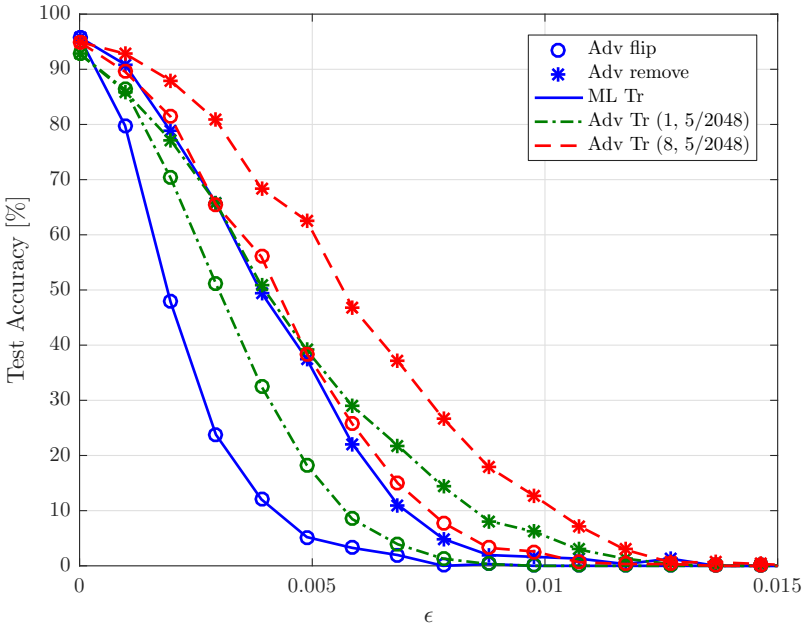


Figure 3.5 Test accuracy under adversarial attacks versus ϵ with time encoding and rate decoding with ML and adversarial training ($T = K = 8$).

decoding. It is observed that robust training can still improve the robustness of the SNN classifier, even when $T_A \ll T$ during training. For instance, for an attacker in the test phase with $\epsilon = 5/2048$, i.e., 5 bit flips, conventional ML achieves an accuracy of 34.2%, while adversarial training with $\epsilon_A = 5/2048$ and $T_A = 1$ and 8 achieves accuracy levels of 60.3% and 77.5%, respectively.

3.6 Conclusions

In this chapter, we have studied for the first time the sensitivity of a probabilistic two-layer SNN under adversarial perturbations. We considered rate and time encoding, as well as rate and first-to-spike decoding. We have proposed mechanisms to build adversarial examples, as well as a robust training method that increases the resilience of the SNN. Additional work is needed in order to generalize the results to multi-layer networks.

CHAPTER 4

UNSUPERVISED TRAINING OF PROBABILISTIC SPIKING NEURAL NETWORKS WITH HYBRID STOCHASTIC-MAP VARIATIONAL LEARNING

4.1 INTRODUCTION

The classical task of unsupervised learning is to find the best representation of the unlabeled data. This type of machine learning aims to capture as much information about the data as possible, while preserving the representation of the data simpler in terms of lower dimensionality, sparsity or independency of features [39]. Generative models provide promising framework towards this goal. The intuition behind of this approach follows a famous quote from Richard Feynman: "What I cannot create, I do not understand". In a generative model, the observations, or the sensory input, is assumed to be generated, or caused, by the state of the hidden variables [40]. In this chapter, we design probabilistic-based generative models for SNNs. SNNs have recently gained a lot of attentions on a variety of machine learning tasks by both academia and industry sectors (see, e.g., [41, 2] and references therein).

The most existing training algorithms for SNNs are based on deterministic computation units, called neurons. The Leaky Integrate-and-Fire (LIF) neuron is indeed the simplest and the most popular one for building such SNNs [11]. However, the conventional training algorithms for SNNs mainly suffer from two big issues: 1) the SNNs are typically trained based on back-propagation method, which unlike ANNs, it becomes challenging due to the non-differentiability nature of the spike dynamics. Although many works have proposed to tackle this problem by introducing approximated derivative approaches [42], this non-differentiability of the spike activity yet greatly challenges the effective learning of SNNs. 2) the deterministic-based dynamics of neurons that limit their flexibility and expressive power [15]. Motivated by this, we consider the problem of training a probabilistic model for SNNs utilizing

rich model properties of Generalized Linear Model (GLM) neuron, which will be discussed in continue. It should be noted that, the research work on probabilistic models for SNNs has been mostly done in the computational neuroscience literature with two main models: 1) neural sampling, in which information is encoded in the steady-state behavior of a subpopulation of neurons [43, 44]; and 2) finite-presentation time models in which the spiking behavior of the network over a given amount of time encodes information (see, e.g., [35, 45, 46, 47], and references therein).

The papers on probabilistic SNNs operating over a finite interval of time adopt either heuristic approximations of the Expectation-Maximization (EM) algorithm [18] or a variational EM based on simple variational posterior [35]. To the best of our knowledge, there is no work on probabilistic models, except [3, 4, 48] on generalized neuromorphic and Statistical Parametric Mapping (SPM).

Many probabilistic models are difficult to train because of their intractable inference part. In the context of deep learning, the challenge of inference refers to the difficult problem of predicting the value of some variables, e.g. latent variables, given other variables, e.g. visible variables, or predicting the probability distribution over some variables given the value of other variables. In fact, the inference problem arise from interactions between latent variables in a structured graphical model. While in undirected models there are direct interactions, in directed models, these interactions are due to explaining away interactions between mutual ancestors of the same visible unit [39]. Several techniques have been proposed to confront the intractable inference problems. The Expectation Maximization (EM) algorithm is the popular training algorithm developed in [49]. EM is not an approach to approximate inference, but rather an approach to learning with an approximate posterior [39]. The Maximum A Posterior (MAP) inference computes the single most likely value of the missing variables, rather than to infer the entire distribution over their possible values [39]. In other words, MAP inference enable us to learn using a point estimate

of posterior rather than inferring the entire distribution. Variational learning [50], instead of computing the log probability of the observed data, uses a variational lower bound, called the Evidence Lower BOund (ELBO), to optimize. There are also many other techniques to confront the intractable inference problems such as amortized variational inference [51], Wake-Sleep (WS) learning [52], adversarial learning [53], and many more.

Sigmoid Belief Networks (SBNs), as directed graphical models, were considered in [54]. In SBN, the data can be generated using ancestral sampling with a fully generative process. However, it has been noted that training a deep directed generative model can be difficult due to the explaining away property. In [55], the mean field approximations have used for SBNs. In [56], the difficulty raised from explaining away has tackled by introducing the idea of complementary priors. The Deep Belief Network (DBN) [56] is a SBN whose top hidden layers is replaced by the RBM that is undirected. It has shown that the RBM can provide a good initialization to the DBN. In [57], Neural Variational Inference and Learning (NVIL) method for directed latent variable models has proposed. This scheme uses a feedforwad network for sampling from the variational posterior for the given observation. The inference network is trained jointly with the model by maximizing the variational lower bound on the log-likelihood. In order to reduce the variance of estimating all the required gradients, NVIL uses a model-independent variance reduction technique. NVIL extends the wake-sleep algorithm [52] to training fast variational approximations for the SBN. In [58], a Gibbs sampling algorithm and mean field Variational Bayes (VB) approximation are developed for learning and inference of model parameters for deep SBNs with sparsity-encouraging priors placed on the model parameters via data augmentation. In [59], a Bayesian inference method is proposed based on doubly stochastic gradient Markov Chain Monte Carlo (MCMC) for Deep Generative Models (DGMs) in a continuous parameter space. At each MCMC sampling step, the

algorithm randomly draws a mini-batch of data samples to estimate the gradient of log-posterior and further estimates the intractable expectation over hidden variables via a Neural Adaptive Importance Sampler (NAIS), where the proposal distribution is parameterized by a deep neural network and learnt jointly.

In [60, 61], a Maximum A Posteriori (MAP) inference method is used for deep SBNs. In this work, unlike variational methods that often use an auxiliary network to approximate the posterior probability inference, MAP scheme is used to maximally preserve the dependencies among latent variables. More specifically, in learning step, the data marginal log-likelihood is maximized directly with a max operation to overcome the exponential number of latent configurations, while in inference step, the pseudo-likelihood method is used to preserve dependencies of latent variables.

In [62], the gradient of a quadratic loss function for supervised learning is approximated by solving two MAP problems with respect to the hidden variables. The first MAP problem corresponds to an unsupervised learning set-up as it does not use information about the labels, yielding the anti-Hebbian component of the gradient. In contrast, the second MAP problem solves a supervised learning problem, and it yields the Hebbian component of the gradient.

In this chapter, we consider unsupervised training task for probabilistic SNNs. We utilize GLM neurons for building the SNNs. The problem of training a Variational Autoencoder (VAE) is studied including two-layer and multi-Layers SNNs for both probabilistic generative and encoding parts. In order to train the VAEs, we consider the standard Maximum Likelihood (ML) approach. For the first time in the field, we develop MAP-based variational learning, as well as Hybrid Stochastic-MAP Variational Learning (HSM-VL) schemes for SNNs. The latter is developed based on Rao-Blackwellization estimator. We also consider the doubly stochastic gradient learning method for the comparison purpose. Numerical results show performance improvements using HSM-VL compared to the other two training schemes.

The rest of this chapter is organized as follows. Variational autoencoder based on probabilistic SNNs is discussed in Section 4.2. Section 4.3 briefly discusses backgrounds on Hybrid Stochastic-MAP Variational Learning (HSM-VL). HSM-VL for a two-layer and multi-Layer SNNs are developed in Sections 4.4 and 4.5, respectively. Performance metrics and numerical results are provided, respectively, in Sections 4.6 and 4.7, followed by conclusion in Section 4.8.

4.2 Variational Autoencoder Based on Probabilistic SNNs

In this chapter, we consider the problem of training a Variational Autoencoder (VAE) [50] in which the encoder and decoder probabilistic mappings are both modeled as SNNs. SNNs are networks of spiking neurons, in which each neuron receives as input and produces as output spiking, that is, binary sparse, signals. As shown in Figure 4.1, the decoder SNN includes: (i) a generative probabilistic SNN illustrated in Figure 4.2, whose behavior is defined by a parameterized joint distribution $P_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{h})$ over visible spiking signals $\mathbf{x}$ and hidden spiking signals $\mathbf{h}$; and (ii) an encoder SNN shown in Figure 4.3, whose output spiking signals $\mathbf{h}$ are distributed according to a parameterized distribution $q_{\boldsymbol{\phi}}(\mathbf{h}|\,\mathbf{x})$ given the visible spiking signals $\mathbf{x}$. The term "visible" refers to the fact that examples for spiking signals $\mathbf{x}$ are included in the training set, as further discussed below, while "hidden" variables are not observed. As seen in Figure 4.1, the VAE may also contain a Natural-to-Spike (N2S) encoder to convert a natural signal, such as an image, into spiking signals, and a Spike-to-Natural (S2N) decoder to recover a natural signal from a spiking signals. As further discussed below, N2S and S2N converters are typically based on rate, time, or population encoding principles [63].

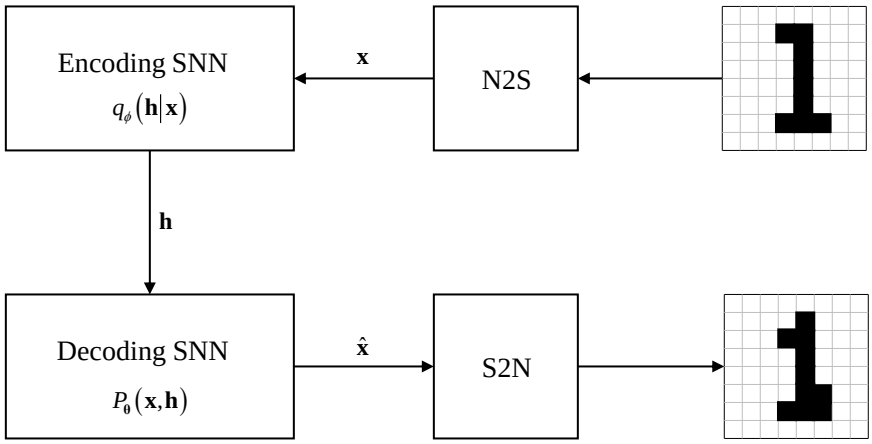


Figure 4.1 Variational Autoencoder (VAE) based on two SNNs: a decoding SNN defined by a generative model $P_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{h})$ and an encoding SNN defined by an encoding model $q_{\boldsymbol{\phi}}(\mathbf{h}|\mathbf{x})$. When the data is a natural (non-spiking) signal, the block diagram includes a Spike domain-to-Natural signal (S2N) decoder, and a Natural-to-Spike domain (N2S) encoder.

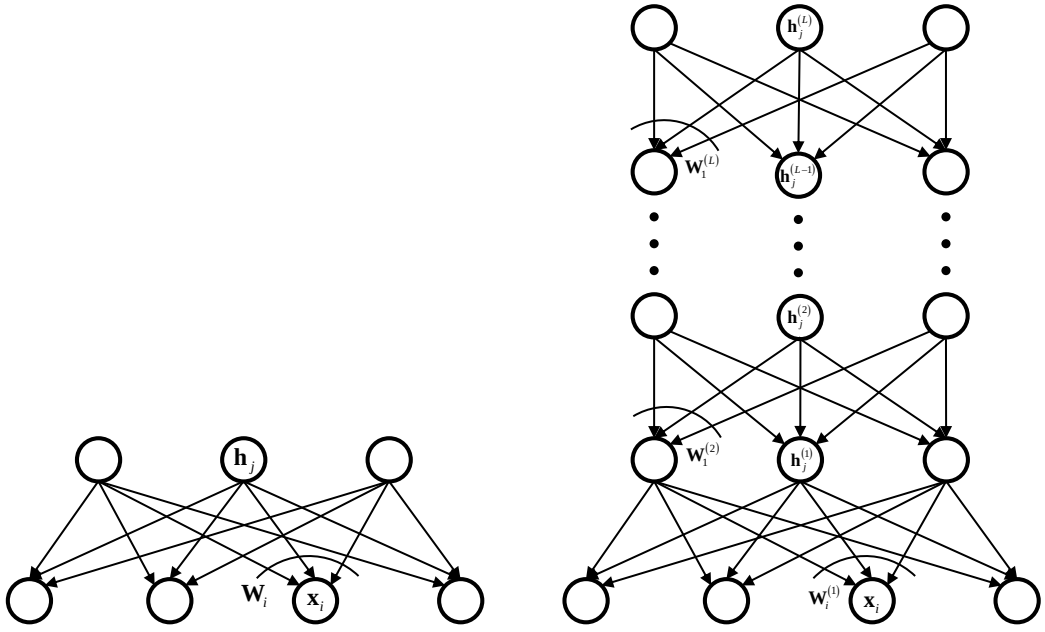
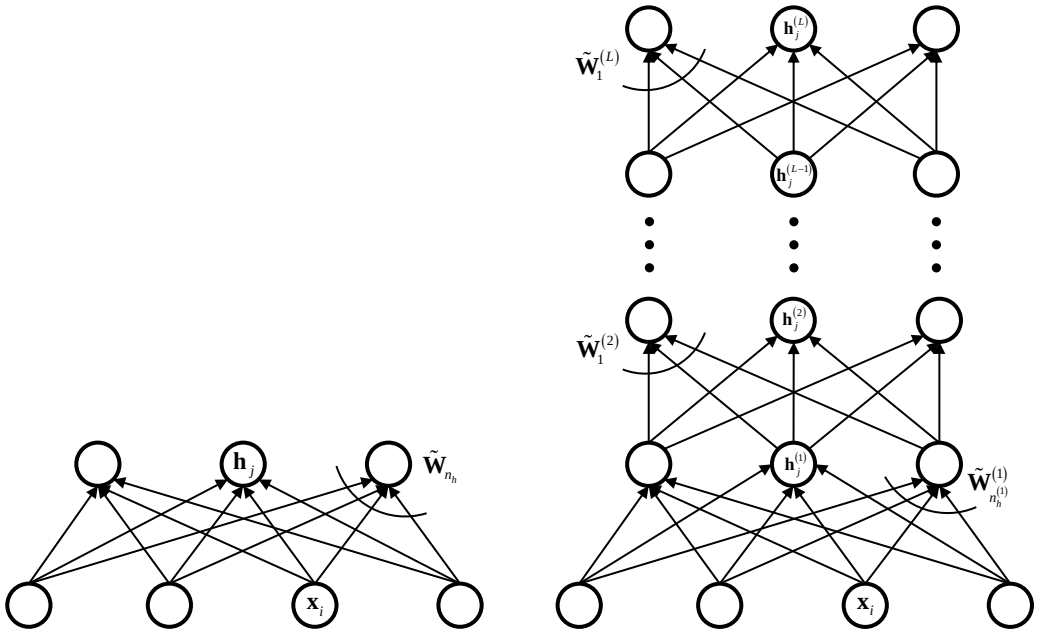


Figure 4.2 Decoding probabilistic SNNs used in the VAE of Figure 4.1: (a) two-layer SNN ($L = 1$), and (b) multi-layer SNN ($L \geq 2$).



(a) **(b)**

Figure 4.3 Encoding probabilistic SNNs used in the VAE of Figure 4.1: (a) two-layer SNN ($L = 1$), and (b) multi-layer SNN ($L \geq 2$).

Training a VAE amounts to the problem of identifying suitable values for parameter vectors $\boldsymbol{\theta}$, for the decoding SNN, and $\boldsymbol{\phi}$, for the encoding SNN, such that the trained encoding SNN operating according to the generative model $P_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{h})$ outputs spiking signals $\mathbf{x}$ that are approximately distributed according to the same distribution underlying the generation of the available training data. In other words, the cascade of the decoding SNN $q_{\boldsymbol{\phi}}(\mathbf{h}|\,\mathbf{x})$ as applied to input spiking signals $\mathbf{x}$ and of the encoding SNN behaving according to the conditional distribution $P_{\boldsymbol{\theta}}(\mathbf{x}|\,\mathbf{h}) = P_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{h}) / P_{\boldsymbol{\theta}}(\mathbf{h})$ reconstructs a signal similar, in some specified sense, to the input spiking signal $\mathbf{x}$.

In the rest of this section, we first describe in more details training data and the models for the decoding SNN defined by generative model $P_{\boldsymbol{\theta}}(\mathbf{x}|\,\mathbf{h})$ and the encoding SNN defined by probability $q_{\boldsymbol{\phi}}(\mathbf{h}|\,\mathbf{x})$.

4.2.1 Training Data

Each example $\mathbf{x}^m \in \{0,1\}^{n_v \times T}$ with $m = 1, ..., M$ in the training set is a collection of n_v sequences of T binary samples with value "1" representing a spike. Parameter n_v is hence the number of observed, or visible, spike trains. All M examples in the training set $\mathcal{X} = \{\mathbf{x}^m\}_{m=1}^M$ are conventionally assumed to be independent identically distributed (i.i.d.) according to a given true data distribution. As illustrated in Figure 4.1, each sample $\mathbf{x}^m$ in the training set $\mathcal{X}$ may be obtained by using an N2S block that converts a natural signal into discrete-time spike signals. As an example, for the case of images, a standard solution is to encode each of the n_v pixels of the input image into a spike train. This can be done by using different methods such as time encoding, whereby the value of the pixel is encoded in the timings of the spikes, or rate encoding, which converts a pixel value into a proportional number of spikes [63, 4]. Alternatively, the data points $\{\mathbf{x}^m\}$ may be directly obtained from neuromorphic or time-based sensors. Examples include data obtained from neuromorphic vision, auditory, and olfactory sensors [64], and time-based sensors [65].

As illustrated in Figure 4.1, we define the generative model by means of an SNN whose output layer provides the desired $n_v \times T$ binary samples $\mathbf{x}$. If the desired output is a natural signal, the spiking signals $\mathbf{x}$ need to be decoded so as to be converted in the desired natural domain by using an S2N block. As for the N2S converter, S2N conversion can be done in different ways. For instance, each spike train may be converted into the pixel of an image by selecting an intensity proportional to the spiking rate or the spike timings. We refer to [63] for an overview of N2S and S2N solutions.

4.2.2 Decoding SNN

For the decoding SNN, we consider a multi-layer SNN model with $L + 1$ layers. As seen in Figure 4.2, the SNN has L layers of hidden, or latent spiking neurons, and

an output layer of observed neurons. The l-th hidden layer has $n_h^{(l)}$ hidden neurons, each producing a spike train of T binary samples. The matrix of T binary samples produced by the $n_h^{(l)}$ neurons in the l-th layer is defined as $\mathbf{h}^{(l)} = \{\mathbf{h}_1^{(l)}, ..., \mathbf{h}_{n_h^{(l)}}^{(l)}\}$. We set $\mathbf{h}^{(0)} = \mathbf{x}$ as the input layer, so that the input $\mathbf{x}$ can be modeled as being generated via the cascade of stochastic layers $\mathbf{h}^{(L)} \to \mathbf{h}^{(L-1)} \to ... \to \mathbf{h}^{(1)} \to \mathbf{h}^{(0)} = \mathbf{x}$. The special case of a two-layer SNN, i.e., $L = 1$, is illustrated in Figure 4.2(a). For reference, in the experiments of Section 4.7, we will also consider a stochastic binary ANN with the same architecture (see, e.g., [61]).

In order to provide mathematical details, we consider first the two-layer SNN as illustrated in Figure 4.2(a). This SNN model has visible spike trains $\mathbf{x} = \{\mathbf{x}_1, ..., \mathbf{x}_{n_v}\}$ and a single layer of latent spike trains $\mathbf{h} = \{\mathbf{h}_1, ..., \mathbf{h}_{n_h}\}$. The joint distribution of $\mathbf{x}$ and $\mathbf{h}$ is modeled as

$$P_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{h}) = P_{\boldsymbol{\theta}}(\mathbf{h})\, P_{\boldsymbol{\theta}}(\mathbf{x}|\, \mathbf{h}) = \prod_{j=1}^{n_h} P_{\theta_j}(\mathbf{h}_j) \prod_{i=1}^{n_v} P_{\boldsymbol{\theta}_i}(\mathbf{x}_i|\, \mathbf{h}), \tag{4.1}$$

where $\boldsymbol{\theta} = \{\{\boldsymbol{\theta}_i\}_{i=1}^{n_v}, \{\theta_j\}_{j=1}^{n_h}\}$ is the vector of parameters that define the prior distribution $P_{\boldsymbol{\theta}}(\mathbf{h})$ of the latent spike trains and the conditional distribution $P_{\boldsymbol{\theta}}(\mathbf{x}|\, \mathbf{h})$. As in most related works, the latent variables $\{\mathbf{h}_j\}$ are assumed to be independent from each other [50, 66, 53]. Furthermore, each latent spike train $\mathbf{h}_j = [h_{j,1}, ..., h_{j,T}]^T$ is assumed to have i.i.d. Bernoulli samples, so that its distribution is

$$P_{\theta_j}(\mathbf{h}_j) = \prod_{t=1}^{T} \sigma\left((2h_{j,t} - 1)\, \theta_j\right), \tag{4.2}$$

where θ_j is the prior log-likelihood ratio for every sample $h_{j,t} \in \{0, 1\}$, and $\sigma(x) \triangleq 1/(1 + \exp(-x))$ is the sigmoid function. As for the distribution of each of the n_v visible spike trains $\mathbf{x}_i$, when conditioned on the latent variables $\mathbf{h}$, we adopt the GLM spiking neuron model, also known as Spike Response Model (SRM), which is widely used in computation neuroscience (see, e.g., [67, 1]). Accordingly, spike trains $\{\mathbf{x}_i\}_{i=1}^{n_v}$ are conditionally independent given the latent variables $\mathbf{h}$, and we have the

conditional distribution

$$P_{\boldsymbol{\theta}_i}\left(\mathbf{x}_i|\,\mathbf{h}\right)=\prod_{t=1}^{T}\sigma\left(\left(2x_{i,t}-1\right)u_{i,t}\right), \tag{4.3}$$

where $u_{i,t}$ is the membrane potential of the i-th visible neuron at time t. The membrane potential $u_{i,t}$ evolves over time as a dynamic system that depends on the past spiking behavior of the hidden neurons and of the visible neuron i, as explained next.

Define as $\mathbf{z}_a^b=[z_b,...,z_a]^T$ the $(b-a+1)\times 1$ vector describing the value of time sequence z_t in the interval $t\in[a,b]$. Assuming a feedforward synaptic memory of τ_α samples and a feedback memory of τ_β samples, the membrane potential is given as [1, 3]

$$u_{i,t}=\sum_{j=1}^{n_h}\boldsymbol{\alpha}_{j,i}^T\mathbf{h}_{j,t-\tau_\alpha}^{t-1}+\boldsymbol{\beta}_i^T\mathbf{x}_{i,t-\tau_\beta}^{t-1}+\gamma_i, \tag{4.4}$$

where $\boldsymbol{\alpha}_{j,i}\in\mathbb{R}^{\tau_\alpha\times 1}$ is the synaptic filter, or kernel, for the synapse between hidden neuron j and visible neuron i, and $\boldsymbol{\beta}_i\in\mathbb{R}^{\tau_\beta\times 1}$ is the feedback filter for spike history dynamics of neuron i. Following [1, 3], we model the feedforward and feedback filters as the linear combination of $K_{\boldsymbol{\alpha}}\leq\tau_\alpha$ and $K_{\boldsymbol{\beta}}\leq\tau_\beta$ basis functions, respectively:

$$\boldsymbol{\alpha}_{j,i}=\mathbf{A}\mathbf{w}_{j,i}=\sum_{k=1}^{K_{\boldsymbol{\alpha}}}w_{j,i,k}\mathbf{a}_k, \tag{4.5}$$

and

$$\boldsymbol{\beta}_i=\mathbf{B}\mathbf{v}_i=\sum_{k=1}^{K_{\boldsymbol{\beta}}}v_{i,k}\mathbf{b}_k, \tag{4.6}$$

where we have defined the matrices $\mathbf{A}=[\mathbf{a}_1,...,\mathbf{a}_{K_{\boldsymbol{\alpha}}}]$ and $\mathbf{B}=[\mathbf{b}_1,...,\mathbf{b}_{K_{\boldsymbol{\beta}}}]$ and the vectors $\mathbf{w}_{j,i}=[w_{j,i,1},...,w_{j,i,K_{\boldsymbol{\alpha}}}]^T$ and $\mathbf{v}_i=[v_{i,1},...,v_{i,K_{\boldsymbol{\beta}}}]^T$; $\mathbf{a}_k=[a_{k,1},...,a_{k,\tau_\alpha}]^T$ and $\mathbf{b}_k=[b_{k,1},...,b_{k,\tau_\beta}]^T$ are the basis vectors; and $\{w_{j,i,k}\}$ and $\{v_{i,k}\}$ are the learnable weights for the kernels $\boldsymbol{\alpha}_{j,i}$ and $\boldsymbol{\beta}_i$, respectively; and the parameter vector $\boldsymbol{\theta}_i=\{\mathbf{W}_i,\mathbf{v}_i,\gamma_i\}$ includes $\mathbf{W}_i=\{\mathbf{w}_{j,i}\}_{j=1}^{n_h}$. For the experiments discussed in Section 4.7,

we adopt the raised cosine basis functions introduced in [1, Section Methods] and illustrated in Figure 4.4.

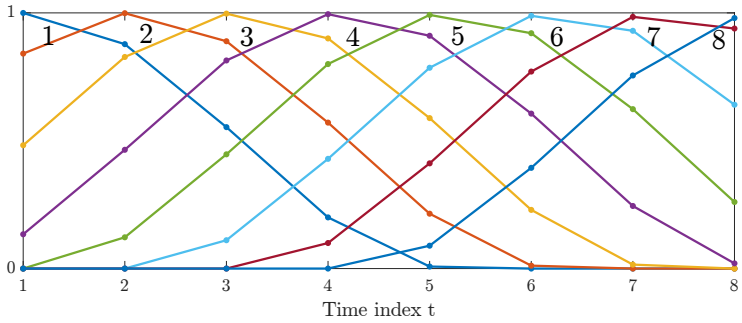


Figure 4.4 Basis functions used in Section 4.7 ($a = 7500$ and $c = 1$ in [1, Section Methods]).

In the case of multi-layer SNN with $L \geq 2$, the prior probability for a variable of the top layer $P_{\theta_j^{(L)}}(\mathbf{h}_j^{(L)})$ is defined as in equation (4.2) with log-likelihood $\theta_j^{(L)}$, while the conditional probability for the remaining layers is given by the GLM

$$P_{\boldsymbol{\theta}_i^{(l-1)}}(\mathbf{h}_i^{(l-1)}|\mathbf{h}^{(l)}) = \prod_{t=1}^{T} \sigma((2h_{i,t}^{(l-1)} - 1)u_{i,t}^{(l-1)}), \tag{4.7}$$

where

$$u_{i,t}^{(l-1)} = \sum_{j=1}^{n_h^{(l)}} \left(\boldsymbol{\alpha}_{j,i}^{(l-1)}\right)^T \mathbf{h}_{j,t-\tau_\alpha}^{(l),t-1} + \left(\boldsymbol{\beta}_i^{(l-1)}\right)^T \mathbf{h}_{i,t-\tau_\beta}^{(l-1),t-1} + \gamma_i^{(l-1)}, \tag{4.8}$$

is the membrane potential for neuron i in layer $l-1$, which is defined in a manner similar to equation (4.4). In particular, the feedforward and feedback filters are parameterized by weight vectors $\mathbf{w}_{j,i}^{(l-1)}$ and $\mathbf{v}_i^{(l-1)}$, respectively, as in equation (4.5)-equation (4.6). Note that, neurons in layer $l-1$ depend solely on the spiking behavior of the upper layer l.

4.2.3 Encoding SNN

For the encoding SNN, we consider a multi-layer SNN model with $L+1$ layers, as illustrated in Figure 4.3. Considering $\mathbf{h}^{(0)} = \mathbf{x}$ as the input layer, we have the

following cascade of stochastic layers $\mathbf{h}^{(0)} = \mathbf{x} \rightarrow \mathbf{h}^{(1)} \rightarrow ... \rightarrow \mathbf{h}^{(L-1)} \rightarrow \mathbf{h}^{(L)}$, that are generated based on variational distributions, as explained next. The special case of a two-layer SNN, i.e., $L = 1$, is illustrated in Figure 4.3(a).

The variational distribution in the best choice is equal to the actual posterior distribution

$$P_{\boldsymbol{\theta}}(\mathbf{h}|\,\mathbf{x}) = \prod_{t=1}^{T} P_{\boldsymbol{\theta}}(\mathbf{h}_t|\,\mathbf{x}), \tag{4.9}$$

where $\mathbf{h}_t = (h_{1,t}, ..., h_{n_h,t})$. As illustrated by this expression, when conditioning of the input $\mathbf{x}$, the hidden variables $\mathbf{h}_t$ at any time t are mutually dependent and they are also dependent on the entire history of $\mathbf{x}$. In fact, past value $\mathbf{x}_{t'}$ with $t' \leq t$ are generally correlated with the future values $\mathbf{x}_{t'}$ with $t' > t$, which in turn depend on the hidden variables $\mathbf{h}_t$ through the membrane potentials equation (4.4). Considering a distribution akin to $P_{\boldsymbol{\theta}}(\mathbf{h}|\,\mathbf{x})$ for the variational distribution generally yields distributions that are difficult to sample from due to the mutual correlation among the latent variables. They also call for the introduction of many variational parameters in order to capture the mutual dependence of all samples $\mathbf{h}_t$ and their dependence on the entire history of $\mathbf{x}$.

For the reasons explained above, the typical approach is to define a variational distribution that has the following simplifying properties: 1) the latent variables $\mathbf{h}_t$ at each time sample t are conditionally independent across the latent neurons; and 2) the dependency of the latent variables $\mathbf{h}_t$ on the input $\mathbf{x}$ is limited. For point 2), one possibility is to make $\mathbf{h}_t$ depend only on the past values of $\mathbf{x}$ (see, e.g., [48]). This approach yields distributed learning rules [48], but it may not capture the most significant correlations between $\mathbf{h}_t$ and the values $\mathbf{x}_{t'}$ with $t' > t$ that are causally influenced by $\mathbf{h}_t$ through the membrane potential equation (4.4). Therefore, here we consider an alternative model, also adopted in [68, 46] in which, under the variational distribution, the values $\mathbf{h}_t$ depend only on τ_α future values of $\mathbf{x}_t$ and τ_β future values

of $\mathbf{h}_t$. More specifically, we assume that the variational distribution over the hidden variables given $\mathbf{x}$ factorizes over the latent variables as

$$q_{\boldsymbol{\phi}}\left(\mathbf{h}|\,\mathbf{x}\right) = \prod_{j=1}^{n_h}\prod_{t=1}^{T}\sigma\left(\left(2h_{j,t}-1\right)\tilde{u}_{j,t}\right), \tag{4.10}$$

where we have defined the *variational membrane potential*

$$\tilde{u}_{j,t} = \sum_{i=1}^{n_v}\tilde{\mathbf{w}}_{j,i}^{T}\mathbf{A}^{T}\mathbf{x}_{i,t+1}^{t+\tau_\alpha} + \tilde{\mathbf{v}}_{j}^{T}\mathbf{B}^{T}\mathbf{h}_{j,t+1}^{t+\tau_\beta} + \tilde{\gamma}_j, \tag{4.11}$$

with variational parameters $\boldsymbol{\phi} = \left\{\boldsymbol{\phi}_j\right\}_{j=1}^{n_h}$ and $\boldsymbol{\phi}_j = \{\tilde{\mathbf{W}}_j, \tilde{\mathbf{v}}_j, \tilde{\gamma}_j\}$; Basis matrices $\mathbf{A}$ and $\mathbf{B}$ are defined as in equation (4.4)–equation (4.6). As compared to the membrane potential equation (4.4), the variational membrane potential is characterized by distinct parameters $\{\tilde{\mathbf{w}}_{j,i}\}$, $\{\tilde{\mathbf{v}}_j\}$, and $\{\tilde{\gamma}_j\}$. Furthermore, the variational membrane potential evolves *backwards*, rather than *forwards* in time.

4.3 Background on Hybrid Stochastic-MAP Variational Learning

In this section, we review the recently proposed Rao-Blackwellized (RB) gradient-based training method as applied to VAEs [69]. The RB stochastic gradient method generalizes techniques based on doubly stochastic variational learning, such as [57], and MAP-based variational learning, such as [61]. We emphasize that neither RB methods nor MAP based techniques have been previously derived for SNNs. All of these techniques aim at approximately maximizing the likelihood of the training data $\mathcal{X} = \{\mathbf{x}^m\}_{m=1}^{M}$. Maximum Likelihood (ML) is a standard method to train generative probabilistic models [70]. In order to introduce the approach, in this section, we provide a general presentation that will be specialized to SNN models in the next sections.

ML Learning. Consider a general generative model, whereby each example $\mathbf{x}$ in the training set is generated from some hidden variables $\mathbf{h}$, so that the joint

probability of $\mathbf{x}$ and $\mathbf{h}$ is given as

$$P_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{h}) = P_{\boldsymbol{\theta}}(\mathbf{h})\, P_{\boldsymbol{\theta}}(\mathbf{x}|\, \mathbf{h}). \tag{4.12}$$

In equation (4.12), functions $P_{\boldsymbol{\theta}}(\mathbf{h})$ and $P_{\boldsymbol{\theta}}(\mathbf{x}|\, \mathbf{h})$ denote the prior probability of the latent variables and the conditional probability of the visible variables $\mathbf{x}$ given latent variables $\mathbf{h}$, respectively. For a given example $\mathbf{x}$, the log marginal probability of the observed data $\mathbf{x}$ is given as

$$\log P_{\boldsymbol{\theta}}(\mathbf{x}) = \log \left(\sum_{\mathbf{h}} P_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{h}) \right). \tag{4.13}$$

The marginalization over $\mathbf{h}$ in equation (4.13) involves summing over all possible configurations of the hidden variables. This entails a complexity that is exponential with the number of hidden variables. Therefore, the exact calculation of equation (4.13) is prohibitively expensive for problems of practical interest. ML learning of the model parameters $\boldsymbol{\theta}$ would require the solution of the problem

$$\underset{\boldsymbol{\theta}}{\text{maximize}} \quad \sum_{m=1}^{M} \log P_{\boldsymbol{\theta}}(\mathbf{x}^m) = \sum_{m=1}^{M} \log \sum_{\mathbf{h}} P_{\boldsymbol{\theta}}(\mathbf{x}^m, \mathbf{h}), \tag{4.14}$$

which maximizes the marginal log-likelihood over training set $\{\mathbf{x}^m\}_{m=1}^{M}$. The intractable marginalization over the latent variables $\mathbf{h}$ in equation (4.14) motivates the use of variational methods as discussed next.

Variational Autoencoder (VAE). In variational learning, the marginalization in equation (4.14) is replaced by an optimization over an auxiliary distribution, known as variational distribution [50, 66, 57, 71, 45]. The variational distribution plays the role of the encoding model $q_{\boldsymbol{\phi}}(\mathbf{h}|\, \mathbf{x})$ in a VAE architecture. The key step in the derivation of variational learning methods definition of a lower bound on the log-likelihood $\log P_{\boldsymbol{\theta}}(\mathbf{x})$, referred as the Evidence Lower BOund (ELBO), that depends on the variational, or encoding, distribution $q_{\boldsymbol{\phi}}(\mathbf{h}|\, \mathbf{x})$. For any distribution

$q_{\boldsymbol{\phi}}(\mathbf{h}|\mathbf{x})$ parameterized by a vector $\boldsymbol{\phi}$, the ELBO is given as [72]

$$\log P_{\boldsymbol{\theta}}(\mathbf{x}) \geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{h}|\mathbf{x})}\left[\log \frac{P_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{h})}{q_{\boldsymbol{\phi}}(\mathbf{h}|\mathbf{x})}\right] \triangleq \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}), \tag{4.15}$$

where the expectation is taken over the hidden variables $\mathbf{h} \sim q_{\boldsymbol{\phi}}(\mathbf{h}|\mathbf{x})$ distributed according to the variational distribution. The ELBO is tight when the difference between approximate and true posterior is zero, i.e., $q_{\boldsymbol{\phi}}(\mathbf{h}|\mathbf{x}) = p_{\boldsymbol{\theta}}(\mathbf{h}|\mathbf{x})$, and the tightness of the bound depends on the Kullback-Leibler (KL) divergence of $q_{\boldsymbol{\phi}}(\mathbf{h}|\mathbf{x})$ and $p_{\boldsymbol{\theta}}(\mathbf{h}|\mathbf{x})$ [72]. Therefore, the variational distribution "encodes" the observation $\mathbf{x}$ into hidden variables $\mathbf{h}$ that are ideally obtained as samples from the posterior $P_{\boldsymbol{\theta}}(\mathbf{h}|\mathbf{x})$.

Doubly Stochastic Gradient Learning. VAE aims at solving the problem [50]

$$\underset{\boldsymbol{\theta},\boldsymbol{\phi}}{\text{maximize}} \quad \sum_{m=1}^{M} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^m). \tag{4.16}$$

This maximizes a lower bound on the log-likelihood in equation (4.14). A key novel element of VAEs is the reuse of the same encoding distribution $q_{\boldsymbol{\phi}}(\mathbf{h}|\mathbf{x})$ for all training points in $\mathcal{X}$, a choice also known as amortized variational inference [51, 73, 74]. This allows the simultaneous learning of the generative model $P_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{h})$ and of the encoding model $q_{\boldsymbol{\phi}}(\mathbf{h}|\mathbf{x})$ (recall discussion in Section 4.2). Optimization is typically done by means of stochastic gradient methods, as discussed next.

In stochastic gradient descent, one example $\mathbf{x}$, or more generally a minibatch of examples are selected at random from the training set. The parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are then updated in the directions of the gradients $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})$ and $\nabla_{\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})$, respectively. The gradient of the ELBO for an example $\mathbf{x}$ with respect to the model parameters $\boldsymbol{\theta}$ can be calculated as

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{h}|\mathbf{x})}\left[\nabla_{\boldsymbol{\theta}} \log P_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{h})\right]. \tag{4.17}$$

The expectation in equation (4.17) can be approximated by the Monte Carlo average as

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) \approx \frac{1}{N} \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}} \log P_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{h}^n), \tag{4.18}$$

where samples $\{\mathbf{h}^n\}_{n=1}^N$ are drawn i.i.d. from the distribution $q_{\boldsymbol{\phi}}(\mathbf{h}|\,\mathbf{x})$. The estimate equation (4.18) of the gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})$ is unbiased and hence it can be used to perform maximization of the ELBO in equation (4.15) through stochastic gradient ascent [75, 59]. This yields the doubly stochastic gradient update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \Delta \boldsymbol{\theta}$, where the double stochastically arises from the random choice of $\mathbf{x}$ and of $\mathbf{h}$.

The gradient of the ELBO for an example $\mathbf{x}$ with respect to the inference parameters $\boldsymbol{\phi}$ can be instead calculated as

$$\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{h}|\mathbf{x})} \left[l_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{h}) \nabla_{\boldsymbol{\theta}} \log q_{\boldsymbol{\phi}}(\mathbf{h}|\,\mathbf{x}) \right], \tag{4.19}$$

where

$$l_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{h}) = \log P_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{h}) - \log q_{\boldsymbol{\phi}}(\mathbf{h}|\,\mathbf{x}) \tag{4.20}$$

is defined as the learning signal (see, e.g., [57]). Eq. equation (4.19) is derived by using the identity $\nabla_{\boldsymbol{\phi}} q_{\boldsymbol{\phi}}(\mathbf{h}|\,\mathbf{x}) = q_{\boldsymbol{\phi}}(\mathbf{h}|\,\mathbf{x}) \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\mathbf{h}|\,\mathbf{x})$. This approach is known as the Likelihood-Ratio (LR) estimator [76, 71, 77, 78], the REINFORCE gradient [79], or score gradient estimator [80]. An unbiased estimate of gradient equation (4.19) using Monte Carlo sampling is given by

$$\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) \approx \frac{1}{N} \sum_{n=1}^{N} l_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{h}^n) \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\mathbf{h}^n|\,\mathbf{x}), \tag{4.21}$$

where $\{\mathbf{h}^n\}_{n=1}^N$ are drawn i.i.d. from the distribution $q_{\boldsymbol{\phi}}(\mathbf{h}|\,\mathbf{x})$. Unlike the estimator in equation (4.18), the variance of the estimator equation (4.21) can be very high due to the high variability of the learning signal [57]. Therefore, the estimate equation (4.21) is typically not directly used in as update step as in equation (4.18).

In fact, doubly stochastic gradient methods, such as NVIL [57], reduce the variance of the estimate equation (4.21) by adding a control variate in the form of a baseline term C. The control variate does not depend on $\mathbf{h}$ and is subtracted from the learning signal in equation (4.21). The resulting estimate

$$\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) \approx \frac{1}{N} \sum_{n=1}^{N} \left(l_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{h}^n) - C \right) \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\mathbf{h}^n | \mathbf{x}), \tag{4.22}$$

is still unbiased but it may have reduced variance depending on the choice of C [81]. A typical choice for C is a moving average of previous values for the learning signal [57]. This yields the doubly stochastic update $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \eta \Delta \boldsymbol{\phi}$.

MAP-based Learning. A MAP-based method still uses the update equation (4.18) for parameters $\boldsymbol{\theta}$, but it substitutes the empirical average in equation (4.21) with a MAP estimate of the hidden variables as

$$\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) \approx l_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{h}_{\text{MAP}}) \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\mathbf{h}_{\text{MAP}} | \mathbf{x}), \tag{4.23}$$

where

$$\mathbf{h}_{\text{MAP}} = \underset{\mathbf{h}}{\operatorname{argmax}}\, q_{\boldsymbol{\phi}}(\mathbf{h} | \mathbf{x}), \tag{4.24}$$

is the MAP estimate of $\mathbf{h}$ under the variational distribution $q_{\boldsymbol{\phi}}(\mathbf{h} | \mathbf{x})$. Note that, we have assumed in equation (4.24) that $q_{\boldsymbol{\phi}}(\mathbf{h} | \mathbf{x})$ is unimodal so that a single maximizer exists, which is the one for typical choice of $q_{\boldsymbol{\phi}}(\mathbf{h} | \mathbf{x})$. In [61], the MAP approach equation (4.23) is introduced for the special case in which the variational distribution $q_{\boldsymbol{\phi}}(\mathbf{h} | \mathbf{x})$ equals the exact posterior $p_{\boldsymbol{\theta}}(\mathbf{h} | \mathbf{x}) = p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{h}) / p_{\boldsymbol{\theta}}(\mathbf{x})$. Note that, unlike the stochastic approximation methods, MAP does not require the variational distribution to be easy to sample from, but the optimization in equation (4.24) should be feasible. The MAP estimate equation (4.23) of the gradient $\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})$ is biased but it may have a lower variance, as we will further discuss in Section 4.7.

RB-based Learning. In [69], a technique is proposed that can be derived as the result of a Rao-Blackwellization step on the stochastic gradient equation (4.21). The resulting gradient estimate is unbiased and its variance is guaranteed to be no larger than for the estimator equation (4.21) [69]. A proof of this result is provided for completeness in Appendix B. The RB gradient estimate is given as

$$\begin{aligned}\nabla_{\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) \approx & q_{\boldsymbol{\phi}}(\mathbf{h}_{\mathrm{MAP}}|\,\mathbf{x})\,(l_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{h}_{\mathrm{MAP}})-C)\,\nabla_{\boldsymbol{\phi}}\log q_{\boldsymbol{\phi}}(\mathbf{h}_{\mathrm{MAP}}|\,\mathbf{x}) \\ & + (1-q_{\boldsymbol{\phi}}(\mathbf{h}_{\mathrm{MAP}}|\,\mathbf{x}))\,\frac{1}{N}\sum_{n=1}^{N}(l_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{h}^n)-C)\,\nabla_{\boldsymbol{\phi}}\log q_{\boldsymbol{\phi}}(\mathbf{h}^n|\,\mathbf{x}).\end{aligned} \tag{4.25}$$

The estimate equation (4.25) can be interpreted as a weighted average of the MAP estimator equation (4.23) and of the doubly stochastic estimator equation (4.22). The weights are given by the probability $q_{\boldsymbol{\phi}}(\mathbf{h}_{\mathrm{MAP}}|\,\mathbf{x})$ and by its complement $1-q_{\boldsymbol{\phi}}(\mathbf{h}_{\mathrm{MAP}}|\,\mathbf{x})$. Note that in equation (4.25), we have subtracted a baseline C also for further reduction in variance of the estimator. Algorithm 4.1 describes the resulting RB-based, also referred to as HSM-VL, scheme.

Algorithm 4.1 Hybrid Stochastic-MAP Variational Learning (HSM-VL)

Input: Training set $\mathcal{X}$; constant λ and learning rate η

Initialize: Model parameters $\boldsymbol{\theta}$ and variational parameters $\boldsymbol{\phi}$

repeat

  Randomly choose a minibatch of data with size K from the training set

  for each example $\mathbf{x}^i$ in the minibatch **do**

    Compute MAP estimate $\mathbf{h}^i_{\text{MAP}} = \underset{\mathbf{h}}{\text{argmax}}\, q_{\boldsymbol{\phi}}\left(\mathbf{h}|\,\mathbf{x}^i\right)$

    Draw a sample $\mathbf{h}^i \sim q_{\boldsymbol{\phi}}\left(\mathbf{h}|\,\mathbf{x}^i\right)$

    Compute the MAP learning signal: $l^i_{\text{MAP}} \leftarrow \log P_{\boldsymbol{\theta}}\left(\mathbf{x}^i, \mathbf{h}^i_{\text{MAP}}\right) - \log q_{\boldsymbol{\phi}}\left(\mathbf{h}^i_{\text{MAP}}|\,\mathbf{x}^i\right)$

    Compute the stochastic learning signal: $l^i \leftarrow \log P_{\boldsymbol{\theta}}\left(\mathbf{x}^i, \mathbf{h}^i\right) - \log q_{\boldsymbol{\phi}}\left(\mathbf{h}^i|\,\mathbf{x}^i\right)$

  end for

  $\bar{C}_b \leftarrow \text{mean}\left(l^1, ..., l^K\right)$

  $\bar{C} \leftarrow \lambda \bar{C} + (1-\lambda)\,\bar{C}_b$

  $\Delta\boldsymbol{\theta} \leftarrow 0$, $\Delta\boldsymbol{\phi} \leftarrow 0$

  for each example i in the minibatch **do**

    $\Delta\boldsymbol{\theta} \leftarrow \Delta\boldsymbol{\theta} + q_{\boldsymbol{\phi}}\left(\mathbf{h}^i_{\text{MAP}}|\,\mathbf{x}^i\right)\nabla_{\boldsymbol{\theta}} \log P_{\boldsymbol{\theta}}\left(\mathbf{x}^i, \mathbf{h}^i_{\text{MAP}}\right) + \left(1 - q_{\boldsymbol{\phi}}\left(\mathbf{h}^i_{\text{MAP}}|\,\mathbf{x}^i\right)\right)\nabla_{\boldsymbol{\theta}} \log P_{\boldsymbol{\theta}}\left(\mathbf{x}^i, \mathbf{h}^i\right)$

    $\Delta\boldsymbol{\phi} \leftarrow \Delta\boldsymbol{\phi} + q_{\boldsymbol{\phi}}\left(\mathbf{h}^i_{\text{MAP}}|\,\mathbf{x}^i\right)\left(l^i_{\text{MAP}} - \bar{C}\right)\nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}\left(\mathbf{h}^i_{\text{MAP}}|\,\mathbf{x}^i\right) + \left(1 - q_{\boldsymbol{\phi}}\left(\mathbf{h}^i_{\text{MAP}}|\,\mathbf{x}^i\right)\right)\left(l^i - \bar{C}\right)\nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}\left(\mathbf{h}^i|\,\mathbf{x}^i\right)$

  end for

  Update model parameters: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta\Delta\boldsymbol{\theta}$

  Update variational parameters: $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \eta\Delta\boldsymbol{\phi}$

until training converged

Output: $\boldsymbol{\theta}$

4.4 Hybrid Stochastic-MAP Variational Learning For a Two-Layer SNN

In this section, we propose an implementation of the HSM-VL learning rule summarized in Algorithm 4.1 for a VAE based on two-layer probabilistic SNN, as illustrated in Figure 4.2(a) the decoding SNN and in Figure 4.3(a) for the encoding SNN. The extension to multi-layer SNNs will be presented in Section 4.5. The key challenge is the definition of a computationally feasible way of evaluating the MAP estimate in equation (4.24). In fact, a brute-force calculation would have a complexity that is exponential in the product $n_h T$.

We start by recalling that ML learning of a general multi-layer SNN with the model parameters $\boldsymbol{\theta} = \{\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, ..., \boldsymbol{\theta}^{(L)}\}$ requires the solution of the problem

$$\underset{\boldsymbol{\theta}}{\text{maximize}} \quad \sum_{m=1}^{M} \log \sum_{\left\{\mathbf{h}^{(l)}\right\}} P_{\boldsymbol{\theta}} \left(\mathbf{x}^m, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, ..., \mathbf{h}^{(L)}\right), \tag{4.26}$$

where one needs to marginalize over the hidden layers $l = 1, ..., L$. As mentioned, in this section we focus on the case $L = 1$. Considering the prior distribution $P_{\boldsymbol{\theta}}(\mathbf{h})$ of the latent spike trains in equation (4.2), and the conditional distribution $P_{\boldsymbol{\theta}}(\mathbf{x}|\,\mathbf{h})$ of the visible spike trains $\mathbf{x}$ given latent spike trains $\mathbf{h}$ in equation (4.3), the gradients of the model parameters $\boldsymbol{\theta}$ in equation (4.17) can be easily computed as

$$\nabla_{\mathbf{w}_{j,i}} \log P_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{h}) = \sum_{t=1}^{T} (x_{i,t} - \sigma(u_{i,t}))\, \mathbf{A}^T \mathbf{h}_{j,t-\tau_\alpha}^{t-1}, \tag{4.27}$$

$$\nabla_{\mathbf{v}_i} \log P_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{h}) = \sum_{t=1}^{T} (x_{i,t} - \sigma(u_{i,t}))\, \mathbf{B}^T \mathbf{x}_{i,t-\tau_\beta}^{t-1}, \tag{4.28}$$

$$\nabla_{\gamma_i} \log P_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{h}) = \sum_{t=1}^{T} (x_{i,t} - \sigma(u_{i,t})), \tag{4.29}$$

and

$$\nabla_{\theta_j} \log P_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{h}) = \sum_{t=1}^{T} (h_{j,t} - \sigma(\theta_j)). \tag{4.30}$$

These expressions can be directly used to update the decoding SNN parameters $\boldsymbol{\theta}$.

With the variational distribution equation (4.10), the gradients of the variational parameters $\boldsymbol{\phi}_j$ can be similarly computed as

$$\nabla_{\tilde{\mathbf{w}}_{j,i}} \log q_{\boldsymbol{\phi}_j}(\mathbf{h}_j|\,\mathbf{x}) = \sum_{t=1}^{T} \left(h_{j,t} - \sigma\left(\tilde{u}_{j,t}\right)\right) \mathbf{A}^T \mathbf{x}_{i,t+1}^{t+\tau_\alpha}, \tag{4.31}$$

$$\nabla_{\tilde{\mathbf{v}}_j} \log q_{\boldsymbol{\phi}_j}(\mathbf{h}_j|\,\mathbf{x}) = \sum_{t=1}^{T} \left(h_{j,t} - \sigma\left(\tilde{u}_{j,t}\right)\right) \mathbf{B}^T \mathbf{h}_{j,t+1}^{t+\tau_\beta}, \tag{4.32}$$

$$\nabla_{\tilde{\gamma}_j} \log q_{\boldsymbol{\phi}_j}(\mathbf{h}_j|\,\mathbf{x}) = \sum_{t=1}^{T} \left(h_{j,t} - \sigma\left(\tilde{u}_{j,t}\right)\right). \tag{4.33}$$

The update of the encoding SNN for $\boldsymbol{\phi}$, however, requires also to compute the MAP estimate of $\mathbf{h}$ for a given example $\mathbf{x}$. To tackle the discussed complexity of this calculation, we propose to maximize the pseudo-likelihood [82], rather than directly maximizing the likelihood. Pseudo-likelihood maximization, also referred as coordinate-ascent variational inference [70], iteratively optimizes each parameter of the likelihood, while holding the others fixed. Accordingly, this method climbs the likelihood to a local optimum.

The pseudo-likelihood function for parameter $\boldsymbol{\phi}_j$ can be written as

$$\mathbf{h}_{\text{MAP}} = \underset{\mathbf{h}}{\text{argmax}} \prod_{j=1}^{n_h} q_{\boldsymbol{\phi}_j}(\mathbf{h}_j|\,\mathbf{h}_{-j}, \mathbf{x}), \tag{4.34}$$

where $\mathbf{h}_{-j}$ is the set of all latent variables except $\mathbf{h}_j$. Accordingly, the maximization of function equation (4.34) can be carried out in parallel for each hidden neuron j. Furthermore, for each such neuron, optimization can be carried out in parallel for all times t by solving

$$\begin{aligned} h_{j,t}^{\text{MAP}} &= \underset{h_{j,t}}{\text{argmax}}\, q_{\boldsymbol{\phi}_j}(h_{j,t}|\,\mathbf{h}_{j,-t}, \mathbf{x}) \\ &= \underset{h_{j,t}}{\text{argmax}} \prod_{t'=1}^{t} \sigma\left(\left(2h_{j,t'} - 1\right)\tilde{u}_{j,t'}\right), \end{aligned} \tag{4.35}$$

where $\mathbf{h}_{j,-t}$ is the set of all elements of $\mathbf{h}_j$ except $h_{j,t}$. Considering the definition of variational membrane potential in equation (4.11), $h_{j,t}$ appears only in the time interval $t' \in [t, t+\tau_\beta]$, optimization equation (4.35) can be rewritten as

$$h_{j,t}^{\text{MAP}} = \underset{h_{j,t}}{\text{argmax}} \prod_{t'=\max(1,t-\tau_\beta)}^{t} \sigma\left((2h_{j,t'} - 1)\,\tilde{u}_{j,t'}\right). \tag{4.36}$$

To summarize, the pseudo-likelihood optimization in equation (4.36) is carried in parallel for each time sample $t \in [1, ..., T]$ and latent variable $j \in \{1, ..., n_h\}$ for given previous values of all other variables. This procedure is iterated until convergence as detailed in Algorithm 4.2. Since $h_{j,t}$ is a binary variable, each iteration of this approach requires $2n_hT$ evaluations of the function in equation (4.36). This is in contrast to the exhaustive optimization of equation (4.24).

A further reduction by noting that there is no need to compute $\tilde{u}_{j,t'}$ in equation (4.36) separately for $h_{j,t}$ and for its complementary value $\bar{h}_{j,t} = 1 - h_{j,t}$. In fact, once we compute $\tilde{u}_{j,t'}$ for a value of $h_{j,t}$, the corresponding $\tilde{u}_{j,t'}$ for the $\bar{h}_{j,t}$, can be obtained readily as

$$\tilde{u}_{j,t'}|_{\bar{h}_{j,t}} = \tilde{u}_{j,t'}|_{h_{j,t}} + (1 - 2h_{j,t})\,\alpha_{j,i,t'-t+\tau_\beta+1}, \tag{4.37}$$

when $t' < t$ and $\tilde{u}_{j,t'}|_{\bar{h}_{j,t}} = \tilde{u}_{j,t'}|_{h_{j,t}}$ when $t' = t$. This reduces the number of calculations of the ... by half as compared to a naive computation. Moreover, the part of the membrane potential $\tilde{u}_{j,t'}$ that depends only on $\mathbf{x}$ and $\tilde{\gamma}_j$ needs to be computed only once for all values of $\mathbf{h}$. Only, the second part that is given as

$$\tilde{\boldsymbol{\beta}}_j^T \mathbf{h}_{j,t'+1}^{t'+\tau_\beta} = \sum_{k=1}^{\tau_\beta} \tilde{\beta}_{j,k} h_{j,t'+\tau_\beta+1-k}, \tag{4.38}$$

needs to be evaluated as is function of $h_{j,t}$. Overall, the complexity of each maximization equation (4.36) is of the order $\mathcal{O}(\tau_\beta)$, and hence, the total computational complexity for the maximization in equation (4.24) is of the order $\mathcal{O}(\tau_\beta n_h T)$.

Algorithm 4.2 Pseudo-MAP Successive Inference

Input: $\mathbf{x}$ and variational parameters $\boldsymbol{\phi}$

Initialize: $\mathbf{h}$

1: **repeat**

2: **for** each latent variable $j \in \{1, ..., n_h\}$ **do**

3: **for** each time sample $t \in [1, 2, ..., T]$ **do**

4: Optimize the pseudo MAP of $h_{j,t}$ using equation (4.36).

5: **end for**

6: **end for**

7: **until** convergence or for a fixed number of iterations

Output: $\mathbf{h}_{\text{MAP}}$

4.5 Hybrid Stochastic-MAP Variational Learning For Multi-Layer SNN

In the case of multi-layer SNN with $L \geq 2$, the variational distribution for the lth layer is given by

$$q_{\boldsymbol{\phi}_j^{(l)}}\left(\mathbf{h}_j^{(l)} \middle| \mathbf{h}^{(l-1)}\right) = \prod_{t=1}^{T} \sigma\left(\left(2h_{j,t}^{(l)} - 1\right)\tilde{u}_{j,t}^{(l)}\right), \tag{4.39}$$

where

$$\tilde{u}_{j,t}^{(l)} = \sum_{i=1}^{n_h^{(l-1)}} \left(\tilde{\boldsymbol{\alpha}}_{j,i}^{(l)}\right)^T \mathbf{h}_{i,t+1}^{(l-1),t+\tau_{\alpha^{(l)}}} + \left(\tilde{\boldsymbol{\beta}}_j^{(l)}\right)^T \mathbf{h}_{j,t+1}^{(l),t+\tau_{\beta^{(l)}}} + \tilde{\gamma}_j^{(l)}, \tag{4.40}$$

where $\tilde{u}_{j,t}^{(l)}$ is the membrane potential for neuron j in layer l, which is defined in a manner similar to equation (4.10).

In order to train the deep SNN in equation (4.26), we perform successively the following two steps until convergence.

Layer-wise Pre-training In this pre-training procedure, we start to learn the deep model from the first layer ($l = 1$) to the last layer ($l = L$). At the first step, we apply Algorithm 4.1 to the two-layer network considering of the observed variables $\mathbf{x}$

and of the latent layer $\mathbf{h}^{(1)}$. At each step $l > 1$, Algorithm 4.1 is used with the MAP estimate or the stochastic sample $\hat{\mathbf{h}}^{(l-1)}$ obtained at the end of Algorithm 4.1 for step $l-1$ as input, while the hidden layer variables are given by layer $\mathbf{h}^{(l)}$.

Global Fine-tuning In this fine tuning procedure, which is performed for every three consecutive hidden layers, we include the effects of upper and lower layers for the given middle layer. The learning parameters are initialized by the parameters learned by layer-wise learning $\left\{\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\phi}}\right\}$, and the hidden variables are initialized to the values $\hat{\mathbf{h}}^{(1)}, \hat{\mathbf{h}}^{(2)}, ..., \hat{\mathbf{h}}^{(L)}$ obtained from the pre-training procedure for each training example $\mathbf{x}$. Accordingly, MAP inference of the latent spike trains from $l = 1$ to $l = L-1$ can be obtained by solving the problem

$$\mathbf{h}_{\text{MAP}}^{(l)} = \underset{\mathbf{h}^{(l)}}{\operatorname{argmax}}\, q_{\boldsymbol{\phi}^{(l)}}\left(\mathbf{h}^{(l)} \middle| \mathbf{h}_{\text{MAP}}^{(l-1)}\right) q_{\boldsymbol{\phi}^{(l+1)}}\left(\hat{\mathbf{h}}^{(l+1)} \middle| \mathbf{h}^{(l)}\right), \tag{4.41}$$

while for the last hidden layer $l = L$ we need to tackle the following problem

$$\mathbf{h}_{\text{MAP}}^{(L)} = \underset{\mathbf{h}^{(L)}}{\operatorname{argmax}}\, q_{\boldsymbol{\phi}^{(L)}}\left(\mathbf{h}^{(L)} \middle| \mathbf{h}_{\text{MAP}}^{(L-1)}\right). \tag{4.42}$$

4.6 Performance Metrics

In order to evaluate the performance of different training methods for VAEs, we consider the following metrics that have been widely considered in prior works.

4.6.1 Evidence Lower BOund (ELBO)

The Negative Log-Likelihood (NLL) is a direct measure of how well the generative model $P_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_{\mathbf{h}} P_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{h})$ learned actual distribution of the data. Assuming the availability of M_T text examples $\{\mathbf{x}^m\}_{m=1}^{M_T}$, distinct from the training examples in $\mathcal{X}$, the NLL is defined as

$$\text{NLL} = -\sum_{m=1}^{M_T} \log P_{\boldsymbol{\theta}}(\mathbf{x}^m). \tag{4.43}$$

Evaluating the NLL in equation (4.43) requires to marginalize over the hidden variables $\mathbf{h}$. In order to estimate the corresponding average, we use here the ELBO. As it discussed before, the ELBO for a given $\mathbf{x}$ and $\mathbf{h}$ can be calculated as

$$\mathcal{L}\left(\mathbf{x}|\,\mathbf{h}\right) = \log P_{\boldsymbol{\theta}}\left(\mathbf{h}\right) + \log P_{\boldsymbol{\theta}}\left(\mathbf{x}|\,\mathbf{h}\right) - \log q_{\boldsymbol{\phi}}\left(\mathbf{h}|\,\mathbf{x}\right). \tag{4.44}$$

In the experiments of Section 4.7, the ELBO for each sample $\mathbf{x}$ of the test set is calculated after making average over N stochastic samples of $\mathbf{h}$ as given by

$$\mathcal{L}\left(\mathbf{x}\right) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}\left(\mathbf{x}|\,\mathbf{h}\left(n\right)\right), \tag{4.45}$$

where $\left\{\mathbf{h}\left(n\right)\right\}_{n=1}^{N} \sim q_{\boldsymbol{\phi}}\left(\mathbf{h}|\,\mathbf{x}\right)$. Note that, the ELBO estimator is asymptotically unbiased, i.e., it converges to the ground truth log-likelihood $\log P_{\boldsymbol{\theta}}\left(\mathbf{x}\right)$ when $N \to \infty$.

4.6.2 Reconstruction Error

The other way of measuring the performance of the model is to compute the mean reconstruction error of the model on the test set. In this metric, we first encode the input image $\mathbf{x}$ to its hidden representation $\mathbf{h}$. Then, the decoder maps back from $\mathbf{h}$ to the input space $\hat{\mathbf{x}}$. The reconstruction error is hence defined as the Mean Squared Error (MSE) between $\mathbf{x}$ and $\hat{\mathbf{x}}$ given as

$$\text{MSE} = \left\|\mathbf{x} - \hat{\mathbf{x}}\right\|^2 = \frac{1}{n_v T}\sum_{i=1}^{n_v}\sum_{t=1}^{T}\left(x_{i,t} - \hat{x}_{i,t}\right)^2. \tag{4.46}$$

4.6.3 Data Generation

Data generation is a qualitative performance metric that we use to visualize the generative performance of the learned models. Accordingly, we randomly generate samples $\mathbf{x}$ on the following ancestral sampling:

$$\mathbf{h}^{(L)} \sim P_{\boldsymbol{\theta}^{(L)}}\left(\mathbf{h}^{(L)}\right) \searrow \mathbf{h}^{(L-1)} \sim P_{\boldsymbol{\theta}^{(L-1)}}\left(\mathbf{h}^{(L-1)}\middle|\,\mathbf{h}^{(L)}\right) \searrow \ldots \searrow \mathbf{x} \sim P_{\boldsymbol{\theta}^{(1)}}\left(\mathbf{x}|\,\mathbf{h}^{(1)}\right) \tag{4.47}$$

More specifically, in the generative model, the top layer is first sampled following the prior distribution $P_{\boldsymbol{\theta}^{(L)}}\left(\mathbf{h}^{(L)}\right)$. Then, the following layers are sampled and conditioned on their upper layers, and so on, until finally we sample $\mathbf{x}$. Figure 4.5 describes the data generation mechanism for a two-hidden layers network. As it clear, since we have here directed graphical models, producing samples with the ancestral sampling from the joint distribution represented by the model is generally very fast and convenient.

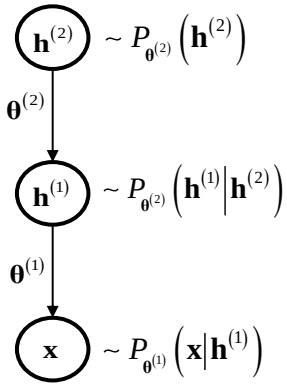


Figure 4.5 Data generation mechanism for a two-hidden layers network.

4.7 Experiment Results

In this section, we evaluate performance of the described training algorithms for SNNs with single-hidden layer and multi-hidden layers based on the metrics introduced in Section 4.6. We use the digits dataset from the UCI machine learning repository [83] as the input data. As a result, we have $n_v = 64$ visible neurons, one visible neuron per pixel of the 8×8 gray scale handwritten digits images. We randomly pick 60%, 20%, and 20% of the dataset to build training, validation, and test sets. Pixel values scaled between 0 and 1. SGD is applied for 500 full pass of dataset samples (each pass termed as an epoch) with batch size of 20 and early stopping is used for all schemes. Holdout validation is applied to select from log-spaced values 10^{-1}, 10^{-2} and 10^{-3} for the constant model and inference learning rates. The model and variational parameters $\{\boldsymbol{\theta}, \boldsymbol{\phi}\}$ are randomly initialized with uniform distribution

between -1 and 1. We assume $T \geq \tau_\alpha = \tau_\beta$ and $K \leq \tau_\alpha = \tau_\beta$. Rate encoding is used to map pixel values to spike sequences with a proportional spike probability between 0 and 1/2.

Figure 4.6 depicts the average reconstruction error, measured in terms of the percentage of samples recovered incorrectly, versus epoch for a single-hidden-layer SNN with $n_h = 10$ and $T = 2$. The average is done over the examples of validation set as well as 100 random realizations of hidden variables for each given example. In the figure, the lines and shaded boundaries represent, respectively, the mean and the standard deviation of reconstitution errors for the stochastic, MAP and HSM-VL training methods. It is observed that the HSM-VL scheme brings less errors in terms of both mean and variance compared to the other two schemes.

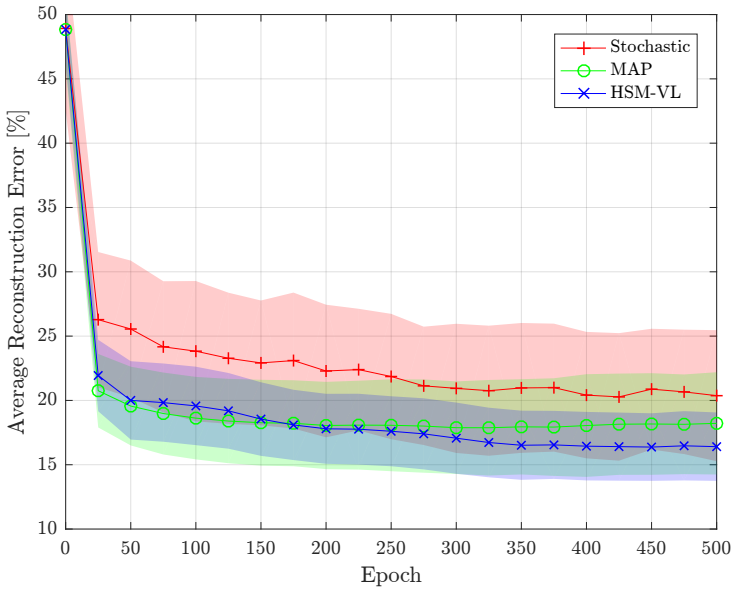


Figure 4.6 Average reconstruction error percentage versus epoch over validation set for stochastic, MAP and HSM-VL training schemes for a single-hidden-layer SNN with $n_h = 10$ and $T = 2$.

Figure 4.6 illustrates the average reconstruction error percentage versus T for a single-hidden-layer SNN trained over 50 epochs using the HSM-VL scheme with $n_h = 10$. The figure illustrates that, the performance of SNNs improves through increasing spike train length, T, as well as increasing K_α and K_β. The latter brings

larger model capacity that enables to have better learning over the shapes of the synaptic and feedback filters with including more basis functions.

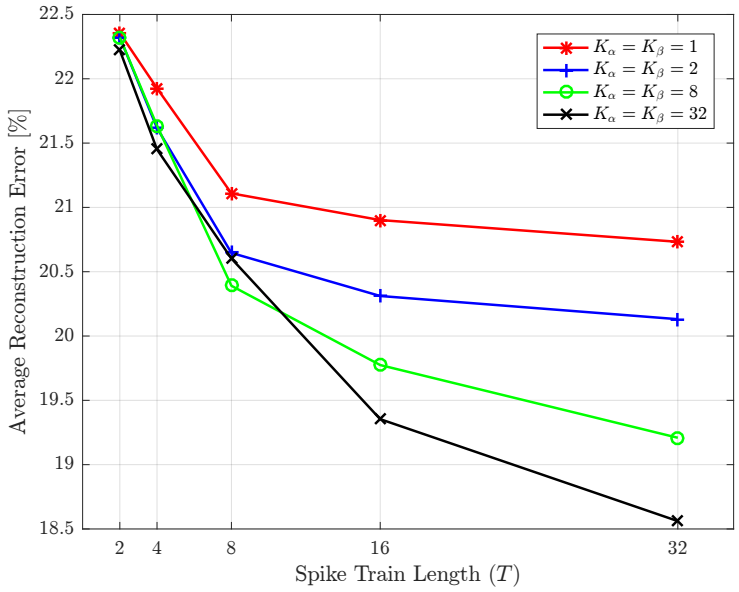


Figure 4.7 Average reconstruction error percentage versus spike train length, T, for a single-hidden-layer SNN trained via the HSM-VL scheme with $n_h = 10$, $\tau_\alpha = \tau_\beta = 32$, and different K_α and K_β values.

Figure 4.8 shows the average ELBO versus epoch for a single-hidden-layer SNN trained using HSM-VL scheme with $n_h = 10$. In this figure, we change the history size of previous inputs τ_α, and previous outputs τ_β for the encoder SNN. As it clear, increasing the window size, the receptive field, helps to improve the performance.

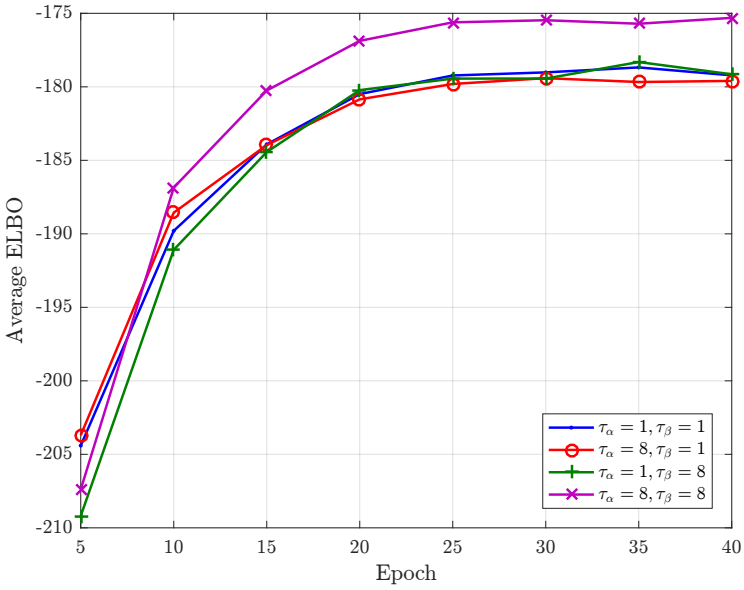


Figure 4.8 Average ELBO versus epoch over validation set for the HSM-VL training scheme for a single-hidden-layer SNN with $n_h = 10$, $T = 8$, $K_\alpha = K_\beta = 1$ and different τ_α and τ_β values.

4.8 Conclusions

In this chapter, the problem of training multi-layers VAEs with probabilistic generative and encoding SNNs were considered. Variational learning schemes including the doubly stochastic gradient learning, MAP-based variational learning, and HSM-VL were developed for SNNs. The HSM-VL showed superior performance compared to the other two schemes.

APPENDIX A

CALCULATION OF GRADIENTS FOR FIRST-TO-SPIKE DECODING

In this appendix you will find the details of calculations for the gradient of $L(\boldsymbol{\theta})$ with respect to $\mathbf{w}_{j,i}$ given in equation (2.12).

The gradient of $L(\boldsymbol{\theta})$ with respect to $\mathbf{w}_{j,i}$ for $i \neq c$ can be calculated as:

$$\begin{aligned}\nabla_{\mathbf{w}_{j,i}} L(\boldsymbol{\theta}) &= \nabla_{\mathbf{w}_{j,i}} \log\left(\sum_{t=1}^{T} p_t(\boldsymbol{\theta})\right) \\ &= \frac{\sum_{t=1}^{T} \nabla_{\mathbf{w}_{j,i}} p_t(\boldsymbol{\theta})}{\sum_{t=1}^{T} p_t(\boldsymbol{\theta})} \\ &= \sum_{t=1}^{T} k_{i,t} \nabla_{\mathbf{w}_{j,i}} \prod_{t'=1}^{t} \bar{g}(u_{i,t'}),\end{aligned} \tag{A.1}$$

where

$$k_{i,t} = \frac{\prod_{i'=1, i' \neq i,c}^{N_Y} \prod_{t'=1}^{t} \bar{g}(u_{i',t'}) g(u_{c,t}) \prod_{t'=1}^{t-1} \bar{g}(u_{c,t'})}{\sum_{t'=1}^{T} p_{t'}(\boldsymbol{\theta})}. \tag{A.2}$$

Using the generalized product rule for derivative of k factors [84] as

$$\begin{aligned}\frac{d}{dx}\prod_{i=1}^{k} f_i(x) &= \sum_{i=1}^{k}\left(\frac{d}{dx} f_i(x) \prod_{j \neq i} f_j(x)\right) \\ &= \left(\prod_{i=1}^{k} f_i(x)\right)\left(\sum_{i=1}^{k} \frac{{f'}_i(x)}{f_i(x)}\right),\end{aligned} \tag{A.3}$$

we have

$$\begin{aligned}\nabla_{\mathbf{w}_{j,i}} \prod_{t'=1}^{t} \bar{g}(u_{i,t'}) &= -\prod_{t'=1}^{t} \bar{g}(u_{i,t'}) \sum_{t'=1}^{t} \frac{g'(u_{i,t'})}{\bar{g}(u_{i,t'})} \nabla_{\mathbf{w}_{j,i}} u_{i,t'} \\ &= -\prod_{t'=1}^{t} \bar{g}(u_{i,t'}) \sum_{t'=1}^{t} \rho_{i,t'} g(u_{i,t'}) \nabla_{\mathbf{w}_{j,i}} u_{i,t'},\end{aligned} \tag{A.4}$$

where we have used the equality $\bar{g}'(u) = -g'(u)$ and $\rho_{i,t}$ is defined as in equation (2.14). After substituting equation (A.4) into equation (A.1), we have

$$\begin{aligned}\nabla_{\mathbf{w}_{j,i}} L(\boldsymbol{\theta}) &= -\sum_{t=1}^{T} q_t \sum_{t'=1}^{t} \rho_{i,t'} g(u_{i,t'}) \nabla_{\mathbf{w}_{j,i}} u_{i,t'} \\ &= -\sum_{t=1}^{T} h_t \rho_{i,t} g(u_{i,t}) \nabla_{\mathbf{w}_{j,i}} u_{i,t},\end{aligned} \tag{A.5}$$

where we have defined q_t and h_t as in equation (2.16) and equation (2.15), respectively. Given that we have the equality $\nabla_{\mathbf{w}_{j,i}} u_{i,t} = \mathbf{A}^T \mathbf{x}_{j,t-\tau_y}^{t-1}$, we have equation (2.12) for $\nabla_{\mathbf{w}_{j,i}} L(\boldsymbol{\theta})$ when $i \neq c$.

The gradient of $L(\boldsymbol{\theta})$ with respect to $\mathbf{w}_{j,i}$ for $i = c$ can be calculated as:

$$\nabla_{\mathbf{w}_{j,c}} L(\boldsymbol{\theta}) = \sum_{t=1}^{T} k_{c,t} \nabla_{\mathbf{w}_{j,c}} \left(g(u_{c,t}) \prod_{t'=1}^{t-1} \bar{g}(u_{c,t'}) \right), \tag{A.6}$$

where

$$k_{c,t} = \frac{\prod_{i=1, i\neq c}^{N_Y} \prod_{t'=1}^{t} \bar{g}(u_{i,t'})}{\sum_{t'=1}^{T} p_{t'}(\boldsymbol{\theta})}. \tag{A.7}$$

Using equation (A.3), we have

$$\begin{aligned}\nabla_{\mathbf{w}_{j,c}} L(\boldsymbol{\theta}) &= \sum_{t=1}^{T} q_t \left[\frac{g'(u_{c,t})}{g(u_{c,t})} \nabla_{\mathbf{w}_{j,c}} u_{c,t} - \sum_{t'=1}^{t-1} \frac{g'(u_{c,t'})}{\bar{g}(u_{c,t'})} \nabla_{\mathbf{w}_{j,c}} u_{c,t'} \right] \\ &= \sum_{t=1}^{T} q_t \left[\rho_{c,t} \nabla_{\mathbf{w}_{j,c}} u_{c,t} - \sum_{t'=1}^{t} \rho_{c,t'} g(u_{c,t'}) \nabla_{\mathbf{w}_{j,c}} u_{c,t'} \right].\end{aligned} \tag{A.8}$$

Thus, by substituting $\nabla_{\mathbf{w}_{j,c}} u_{c,t} = \mathbf{A}^T \mathbf{x}_{j,t-\tau_y}^{t-1}$ into equation (A.8) and after simplification, equation equation (2.12) is obtained for $i = c$, which completes the proof. Note that, the same procedure is done for $\nabla_{\gamma_i} L(\boldsymbol{\theta})$ by considering the equality $\nabla_{\gamma_i} u_{i,t} = 1$ for all i.

APPENDIX B

VARIANCE OF THE HSM-VL SCHEME COMPARED TO THE STOCHASTIC SCHEME

In this appendix, we will quantify the variance reduction by the HSM-VL scheme compared to the stochastic scheme.

In general, we can consider $\mathbf{h} \in \{\mathbf{h}_{\mathrm{MAP}}, \mathbf{h}_{\mathrm{S}}\}$ where $\mathbf{h}_{\mathrm{MAP}}$ is defined as in equation (4.24) and $\mathbf{h}_{\mathrm{S}}$ is a random sample from the distribution $q_{\boldsymbol{\phi}}(\mathbf{h}|\,\mathbf{x})$ with the assumption that $\mathbf{h}_{\mathrm{S}} \neq \mathbf{h}_{\mathrm{MAP}}$, i.e., $\mathbf{h}_{\mathrm{S}} \sim q_{\boldsymbol{\phi}}(\mathbf{h}|\,\mathbf{h} \neq \mathbf{h}_{\mathrm{MAP}}, \mathbf{x})$ where

$$q_{\boldsymbol{\phi}}(\mathbf{h}|\,\mathbf{h} \neq \mathbf{h}_{\mathrm{MAP}}, \mathbf{x}) = \frac{q_{\boldsymbol{\phi}}(\mathbf{h}|\,\mathbf{x})}{q_{\boldsymbol{\phi}}(\mathbf{h} \neq \mathbf{h}_{\mathrm{MAP}}|\,\mathbf{x})} = \frac{q_{\boldsymbol{\phi}}(\mathbf{h}|\,\mathbf{x})}{1 - q_{\boldsymbol{\phi}}(\mathbf{h}_{\mathrm{MAP}}|\,\mathbf{x})}. \tag{B.1}$$

Accordingly, we consider $\mathbf{h} = \mathbf{h}_{\mathrm{MAP}}^{b}\mathbf{h}_{\mathrm{S}}^{1-b}$ where b is a Bernoulli random variable with probability of $q_{\boldsymbol{\phi}}(\mathbf{h}_{\mathrm{MAP}}|\,\mathbf{x})$, i.e., $b \sim \mathrm{Bern}\left(q_{\boldsymbol{\phi}}(\mathbf{h}_{\mathrm{MAP}}|\,\mathbf{x})\right)$. With this fact, the RB estimation of equation (4.19) can be obtained as

$$\begin{aligned}\hat{g}_{\mathrm{RB}}(\mathbf{x}) &= E_{b\sim\mathrm{Bern}\left(q_{\boldsymbol{\phi}}(\mathbf{h}_{\mathrm{MAP}}|\mathbf{x})\right)}\left[g\left(\mathbf{x}, \mathbf{h}_{\mathrm{MAP}}^{b}\mathbf{h}_{\mathrm{S}}^{1-b}\right)\big|\,\mathbf{h}_{\mathrm{S}}\right] \\ &= q_{\boldsymbol{\phi}}(\mathbf{h}_{\mathrm{MAP}}|\,\mathbf{x})\, g(\mathbf{x}, \mathbf{h}_{\mathrm{MAP}}) + (1 - q_{\boldsymbol{\phi}}(\mathbf{h}_{\mathrm{MAP}}|\,\mathbf{x}))\, g(\mathbf{x}, \mathbf{h}_{\mathrm{S}}),\end{aligned} \tag{B.2}$$

where $g(\mathbf{x}, \mathbf{h}) = l_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{h})\, \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\mathbf{h}|\,\mathbf{x})$. It is straightforward to show that the RB estimator is unbiased, i.e.,

$$\mathcal{B}\left[\hat{g}_{\mathrm{RB}}(\mathbf{x})\right] = E_{\mathbf{h}\sim q_{\boldsymbol{\phi}}(\mathbf{h}|\mathbf{x})}\left[\hat{g}_{\mathrm{RB}}(\mathbf{x})\right] - \nabla_{\boldsymbol{\phi}}\mathcal{L}_{\theta,\boldsymbol{\phi}}(\mathbf{x}) = 0. \tag{B.3}$$

Considering the stochastic estimator for equation (4.19) given as

$$\hat{g}_{\mathrm{S}}(\mathbf{x}) = g(\mathbf{x}, \mathbf{h}), \tag{B.4}$$

where $\mathbf{h} \sim q_{\boldsymbol{\phi}}(\mathbf{h}|\,\mathbf{x})$, the conditional variance decomposition gives us

$$\mathrm{var}\left[\hat{g}_{\mathrm{S}}(\mathbf{x})\right] = \mathrm{var}\left[\hat{g}_{\mathrm{RB}}(\mathbf{x})\right] + E_{\mathbf{h}_{\mathrm{S}}\sim q_{\boldsymbol{\phi}}(\mathbf{h}|\mathbf{h}\neq\mathbf{h}_{\mathrm{MAP}},\mathbf{x})}\left[\mathrm{var}\left[g\left(\mathbf{x}, \mathbf{h}_{\mathrm{MAP}}^{b}\mathbf{h}_{\mathrm{S}}^{1-b}\right)\big|\,\mathbf{h}_{\mathrm{S}}\right]\right]. \tag{B.5}$$

Following the non-negativity of the second term in equation (B.5), it is clear that $\text{var}\left[\hat{g}_{\text{RB}}\left(\mathbf{x}\right)\right] \leq \text{var}\left[\hat{g}_{\text{S}}\left(\mathbf{x}\right)\right]$.

REFERENCES

[1] J. W. Pillow, J. Shlens, L. Paninski, A. Sher, A. M. Litke, E. J. Chichilnisky, and E. P. Simoncelli, "Spatio-temporal correlations and visual signalling in a complete neuronal population," *Nature*, vol. 454, no. 7207, pp. 995–999, 2008.

[2] B. Rajendran, A. Sebastian, M. Schmuker, N. Srinivasa, and E. Eleftheriou, "Low-power neuromorphic hardware for signal processing applications," *arXiv preprint arXiv:1901.03690*, 2019.

[3] A. Bagheri, O. Simeone, and B. Rajendran, "Training probabilistic spiking neural networks with first-to-spike decoding," *IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, 2018.

[4] ——, "Adversarial training for probabilistic spiking neural networks," *IEEE Int. Wksh. Signal Process. Adv. Wireless Commun. (SPAWC)*, 2018.

[5] H. Paugam-Moisy and S. Bohte, "Computing with spiking neuron networks," *Hdbk. Natrl. Comput.*, pp. 335–376, 2012.

[6] J. Vincent, "Intel investigates chips designed like your brain to turn the AI tide," *https://www.theverge.com/2017/9/26/16365390/intel-investigates-chips-designed-like-your-brain-to-turn-the-ai-tide*, accessed Sept. 26, 2017.

[7] A. Diamond, T. Nowotny, and M. Schmuker, "Comparing neuromorphic solutions in action: implementing a bio-inspired solution to a benchmark classification task on three parallel-computing platforms," *Front. Neurosci.*, vol. 9, p. 491, 2016.

[8] F. Ponulak and A. Kasiński, "Supervised learning in spiking neural networks with ReSuMe: sequence learning, classification, and spike shifting," *Neural Comput.*, vol. 22, no. 2, pp. 467–510, 2010.

[9] R. V. Florian, "Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity," *Neural Comput.*, vol. 19, no. 6, pp. 1468–1502, 2007.

[10] P. O'Connor and M. Welling, "Deep spiking networks," *arXiv preprint. arXiv:1602.08323*, 2016.

[11] E. Hunsberger and C. Eliasmith, "Spiking deep networks with LIF neurons," *arXiv preprint. arXiv:1510.08829*, 2015.

[12] N. Anwani and B. Rajendran, "NormAD-normalized approximate descent based supervised learning rule for spiking neurons," *IEEE Int. Joint Conf. Neural Netw. (IJCNN)*, 2015.

[13] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Front. Neurosci.*, vol. 10, p. 508, 2016.

[14] J. W. Pillow, L. Paninski, V. J. Uzzell, E. P. Simoncelli, and E. J. Chichilnisky, "Prediction and decoding of retinal ganglion cell responses with a probabilistic spiking model," *J. Neurosci.*, vol. 25, no. 47, pp. 11 003–11 013, 2005.

[15] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. Cambridge, MA: MIT Press, 2009.

[16] O. Simeone, "A brief introduction to machine learning for engineers," *arXiv preprint arXiv:1709.02840*, 2017.

[17] R. Jolivet, A. Rauch, H. Lüscher, and W. Gerstner, "Predicting spike timing of neocortical pyramidal neurons by simple threshold models," *J. Comput. Neurosci.*, vol. 21, no. 1, pp. 35–49, 2006.

[18] B. Gardner and A. Grüning, "Supervised learning in spiking neural networks for precise temporal encoding," *PloS ONE*, vol. 11, no. 8, p. e0161335, 2016.

[19] T. Masquelier and S. J. Thorpe, "Unsupervised learning of visual features through spike timing dependent plasticity," *PLoS Comput. Biol.*, vol. 3, no. 2, p. e31, 2007.

[20] M. Mozafari, S. Kheradpisheh, T. Masquelier, A. Nowzari-Dalini, and M. Ganjtabesh, "First-spike based visual categorization using reward-modulated STDP," *arXiv preprint arXiv:1705.09132*, 2017.

[21] J. Wang, A. Belatreche, L. P. Maguire, and T. M. McGinnity, "SpikeTemp: An enhanced rank-order-based learning approach for spiking neural networks with adaptive structure," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 1, pp. 30–43, 2017.

[22] Z. Lin, D. Ma, J. Meng, and L. Chen, "Relative ordering learning in spiking neural network for pattern recognition," *Neurocomputing*, vol. 275, pp. 94–106, 2018.

[23] J. Shlens, "Notes on generalized linear models of neurons," *arXiv preprint arXiv:1404.1999*, 2014.

[24] P. Baldi and A. F. Atiya, "How delays affect neural dynamics and learning," *IEEE Trans. Neural Netw.*, vol. 5, no. 4, pp. 612–621, 1994.

[25] A. Taherkhani, A. Belatreche, Y. Li, and L. P. Maguire, "DL-ReSuMe: a delay learning-based remote supervised method for spiking neurons," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 12, pp. 3137–3149, 2015.

[26] N. Frémaux and W. Gerstner, "Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules," *Front. Neural Circuits*, vol. 9, 2015.

[27] Y. LeCun, "The MNIST database of handwritten digits," *http://yann.lecun.com/exdb/mnist/*, accessed Sept. 26, 2017.

[28] R. Ranjan, S. Sankaranarayanan, A. Bansal, N. Bodla, J.-C. Chen, V. M. Patel, C. D. Castillo, and R. Chellappa, "Deep learning for understanding faces: Machines may be just as good, or better, than humans," *IEEE Signal Process. Mag.*, vol. 35, no. 1, pp. 66–83, 2018.

[29] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *Int. Conf. on Learn. Repr. (ICLR)*, 2015.

[30] A. Fawzi, S.-M. Moosavi-Dezfooli, and P. Frossard, "The robustness of deep networks: A geometrical perspective," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 50–62, 2017.

[31] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[32] J. E. Smith, "Research agenda: Spacetime computation and the neocortex," *IEEE Micro*, vol. 37, no. 1, pp. 8–14, 2017.

[33] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: VGG and residual architectures," *arXiv preprint arXiv:1802.02627*, 2018.

[34] S. Guo, Z. Yu, F. Deng, X. Hu, and F. Chen, "Hierarchical bayesian inference and learning in spiking neural networks," *IEEE Trans. Cybern.*, vol. 49, no. 1, pp. 133–145, 2019.

[35] D. J. Rezende, D. Wierstra, and W. Gerstner, "Variational learning for recurrent spiking networks," *Adv. Neural Inf. Process. Syst.*, pp. 136–144, 2011.

[36] E. Stromatias, M. Soto, T. Serrano-Gotarredona, and B. Linares-Barranco, "An event-driven classifier for spiking neural networks fed with synthetic or dynamic vision sensor data," *Front. Neurosci.*, vol. 11, pp. 1–17, 2017.

[37] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "STDP-based spiking deep neural networks for object recognition," *arXiv preprint arXiv:1611.01421*, 2016.

[38] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv preprint arXiv:1607.02533*, 2016.

[39] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. Cambridge, MA: MIT Press, 2016.

[40] G. E. Hinton and Z. Ghahramani, "Generative models for discovering sparse distributed representations," *Philos. Trans. R. Soc. Lond., B, Biol. Sci.*, vol. 352, no. 1358, pp. 1177–1190, 1997.

[41] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities & challenges," *Front. Neurosci.*, vol. 12, p. 774, 2018.

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

[43] L. Buesing, J. Bill, B. Nessler, and W. Maass, "Neural dynamics as sampling: a model for stochastic computation in recurrent networks of spiking neurons," *PLoS Comput. Biol.*, vol. 7, no. 11, p. e1002211, 2011.

[44] D. Pecevski, L. Buesing, and W. Maass, "Probabilistic inference in general graphical models through sampling in stochastic networks of spiking neurons," *PLoS Comput. Biol.*, vol. 7, no. 12, p. e1002294, 2011.

[45] A. Mnih and D. J. Rezende, "Variational inference for monte carlo objectives," *arXiv preprint arXiv:1602.06725*, 2016.

[46] J. Brea, W. Senn, and J.-P. Pfister, "Sequence learning with hidden units in spiking neural networks," *Adv. Neural Inf. Process. Syst.*, pp. 1422–1430, 2011.

[47] H. Jang, O. Simeone, B. Gardner, and A. Grüning, "Spiking neural networks: A stochastic signal processing perspective," *arXiv preprint arXiv:1812.03929*, 2018.

[48] H. Jang and O. Simeone, "Training dynamic exponential family models with causal and lateral dependencies for generalized neuromorphic computing," *arXiv preprint arXiv:1810.08940*, 2018.

[49] R. M. Neal and G. E. Hinton, "A view of the EM algorithm that justifies incremental, sparse, and other variants," *Learn. graphical models*, pp. 355–368, 1998.

[50] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[51] S. Gershman and N. Goodman, "Amortized inference in probabilistic reasoning," *Cogsci.*, vol. 36, no. 36, 2014.

[52] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal, "The wake-sleep algorithm for unsupervised neural networks," *Science*, vol. 268, no. 5214, pp. 1158–1161, 1995.

[53] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Adv. Neural Inf. Process. Syst.*, pp. 2672–2680, 2014.

[54] R. M. Neal, "Connectionist learning of belief networks," *AI*, vol. 56, no. 1, pp. 71–113, 1992.

[55] L. K. Saul, T. Jaakkola, and M. I. Jordan, "Mean field theory for sigmoid belief networks," *J. Artif. Intell. Res. (JAIR)*, vol. 4, pp. 61–76, 1996.

[56] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.

[57] A. Mnih and K. Gregor, "Neural variational inference and learning in belief networks," *arXiv preprint arXiv:1402.0030*, 2014.

[58] Z. Gan, R. Henao, D. Carlson, and L. Carin, "Learning deep sigmoid belief networks with data augmentation," *AI and Stats. (AISTATS)*, pp. 268–276, 2015.

[59] C. Du, J. Zhu, and B. Zhang, "Learning deep generative models with doubly stochastic gradient MCMC," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 7, pp. 3084–3096, 2018.

[60] S. Nie, Y. Zhao, and Q. Ji, "Latent regression bayesian network for data representation," *Int. Conf. Pattern Recog. (ICPR)*, pp. 3494–3499, 2016.

[61] S. Nie, M. Zheng, and Q. Ji, "The deep regression bayesian network and its applications: Probabilistic deep learning for computer vision," *IEEE Signal Process. Mag.*, vol. 35, no. 1, pp. 101–111, 2018.

[62] B. Scellier and Y. Bengio, "Equilibrium propagation: Bridging the gap between energy-based models and backpropagation," *Front. Comput. Neurosci.*, vol. 11, p. 24, 2017.

[63] W. Gerstner and W. M. Kistler, *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge, UK: Cambridge University Press, 2002.

[64] A. Vanarse, A. Osseiran, and A. Rassau, "A review of current neuromorphic approaches for vision, auditory, and olfactory sensors," *Front. Neurosci.*, vol. 10, p. 115, 2016.

[65] C. Posch, D. Matolin, and R. Wohlgenannt, "An asynchronous time-based image sensor," *Int. Sym. Circuits Syst. (ISCAS)*, pp. 2130–2133, 2008.

[66] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," *arXiv preprint arXiv:1401.4082*, 2014.

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

[68] J. W. Pillow and P. E. Latham, "Neural characterization in partially observed populations of spiking neurons," *Adv. Neural Inf. Process. Syst.*, pp. 1161–1168, 2008.

[69] R. Liu, J. Regier, N. Tripuraneni, M. I. Jordan, and J. McAuliffe, "Rao-blackwellized stochastic gradients for discrete distributions," *arXiv preprint arXiv:1810.04777*, 2018.

[70] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY: Springer-Verlag, 2006.

[71] S. Gu, S. Levine, I. Sutskever, and A. Mnih, "Muprop: Unbiased backpropagation for stochastic neural networks," *arXiv preprint arXiv:1511.05176*, 2015.

[72] O. Simeone, "A brief introduction to machine learning for engineers," *Fdn. Trends Signal Process.*, vol. 12, no. 3-4, pp. 200–431, 2018.

[73] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," *arXiv preprint arXiv:1505.05770*, 2015.

[74] J. Marino, Y. Yue, and S. Mandt, "Iterative amortized inference," *arXiv preprint arXiv:1807.09356*, 2018.

[75] B. Dai, B. Xie, N. He, Y. Liang, A. Raj, M.-F. F. Balcan, and L. Song, "Scalable kernel methods via doubly stochastic gradients," *Adv. Neural Inf. Process. Syst.*, pp. 3041–3049, 2014.

[76] P. W. Glynn, "Likelihood ratio gradient estimation for stochastic systems," *Commun. ACM*, vol. 33, no. 10, pp. 75–84, 1990.

[77] D. Ritchie, P. Horsfall, and N. D. Goodman, "Deep amortized inference for probabilistic programs," *arXiv preprint arXiv:1610.05735*, 2016.

[78] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," *arXiv preprint arXiv:1611.00712*, 2016.

[79] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[80] M. C. Fu, "Gradient estimation," *Hdbk. Ops. Res. Mngmt. Sci.*, vol. 13, pp. 575–616, 2006.

[81] J. Paisley, D. Blei, and M. Jordan, "Variational bayesian inference with stochastic search," *arXiv preprint arXiv:1206.6430*, 2012.

[82] J. Besag, "Statistical analysis of non-lattice data," *J. R. Stat. Soc. Series D Stat. Methodol.*, vol. 24, no. 3, pp. 179–195, 1975.

[83] A. Frank and A. Asuncion, "UCI machine learning repository," *http://archive.ics.uci.edu/ml*, accessed Apr. 12, 2019.

[84] Wikipedia, "Product rule," *https://en.wikipedia.org/wiki/Product_rule*, accessed Oct. 27, 2017.