AGH

AGH University of Krakow

FIELD OF SCIENCE ENGINEERING AND TECHNOLOGY

SCIENTIFIC DISCIPLINE AUTOMATION, ELECTRONICS, ELECTRICAL ENGINEERING AND SPACE TECHNOLOGIES

DOCTORAL THESIS

Event-Based Machine Learning with Bayesian Methods and Spiking Neural Networks

Author: Mateusz Pabian, MSc

First supervisor: Prof. Mirosław Pawlak, DSc
Assisting supervisor: Dominik Rzepka, PhD

Completed in: AGH University of Krakow, Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering

Krakow, 2024

AGH

AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

DZIEDZINA NAUK INŻYNIERYJNO-TECHNICZNYCH

DYSCYPLINA AUTOMATYKA, ELEKTRONIKA, ELEKTROTECHNIKA
I TECHNOLOGIE KOSMICZNE

ROZPRAWA DOKTORSKA

Uczenie maszynowe dla danych zdarzeniowych przy pomocy metod bayesowskich oraz impulsowych sieci neuronowych

Autor:	mgr inż. Mateusz Pabian
Promotor rozprawy:	prof. dr hab. Mirosław Pawlak
Promotor pomocniczy:	dr inż. Dominik Rzepka

Praca wykonana: Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie, Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Kraków, 2024

I would like to express my deepest gratitude to my supervisor, Prof. Mirosław Pawlak, for his patient guidance, thoughtful feedback and constructive criticism throughout my PhD studies. Additionally, this endeavor would not have been possible without my assisting supervisor, Dr. Dominik Rzepka, who inspired me to start this journey and was always eager to discuss and refine research ideas. Lastly, I would like to thank my family, friends, and everyone else who supported me along the way. You helped me grow into the person I am today, for which I am forever grateful.

The research that contributed to this thesis was supported by the Polish National Science Centre under Grant DEC-2017/27/B/ST7/03082.

Abstract

With the rising volume of data processed by modern measurement and IT systems there is a need for the development of scalable machine learning solutions analyzing event data. While existing methods can be adapted to process such types of data, doing so introduces redundancy. And so, algorithms that are dedicated to sparse, event-based data representations are required.

The aim of this dissertation is to discuss the applicability of two different approaches to machine learning from event data. The classical theory of point processes is used extensively in computational neuroscience to describe spiking patterns; nevertheless, its application to supervised temporal sequence classification is surprisingly under-developed. Conversely, the artificial spiking neural networks (SNN) are more widely used. However, these models are usually trained by simulating the state of the entire network over time. For efficiency, the training procedure should instead consider the network state only at event occurrence. Furthermore, it is unclear how the signal-to-spike conversion process impacts the trained model performance when extracting event sequences from analog and digital data.

The aforementioned knowledge gaps are addressed using statistical analysis and numerical simulations. Classification rules for point processes are proposed based on the Bayes theory of classification. This algorithm is subsequently analyzed in terms of the rate of convergence to the optimal Bayes risk as the number of training examples increases. The impact of boundary effects on the kernel classifier performance is also assessed. In the scope of the SNN framework an existing single-spike time-to-first-spike layer computation is parallelized to achieve a significant speed-up compared to the original formulation. This model is further extended to process multiple input events and generate multiple output events. Lastly, modifying the training objective leads to the Siamese SNN model – the first-ever end-to-end training of a Siamese network in the spiking domain.

The practical implications of this research are evaluated in three supervised machine learning tasks: social media bot detection, cosmic ray imaging artifact rejection, and in-roadway sensor vehicle type identification. In all scenarios the impact of event sequence preprocessing (including signal-to-spike conversion) and hyperparameter choice is assessed. Selecting such distinct case-studies highlights the versatility of the methods developed in this thesis.

Streszczenie

W związku z narastającą ilością danych przetwarzanych przez współczesne systemy informatyczne i pomiarowe, nastaje konieczność opracowania skalowalnych metod uczenia maszynowego analizujących dane zdarzeniowe. O ile możliwe jest przetworzenie tego rodzaju danych przez klasyczne metody, ich użycie wymaga konwersji danych do postaci zawierającej redundancję. Kluczowe jest zatem rozwijanie algorytmów dedykowanych do danych zdarzeniowych.

Celem tej rozprawy jest przeanalizowanie dwóch różnych podejść do tematu uczenia maszynowego dla danych zdarzeniowych. Teoria procesów punktowych jest często wykorzystywana w neurobiologii obliczeniowej do analizy impulsów, jednakże jej użycie w szerszym kontekście nadzorowanej klasyfikacji szeregów czasowych jest zaskakująco rzadko spotykane. Z drugiej strony, impulsowe sieci neuronowe cieszą się dużym zainteresowaniem badaczy, mimo iż ich proces uczenia zwykle wymaga symulowania stanu całej sieci w każdej chwili czasowej, co nie jest efektywnym wykorzystaniem zasobów obliczeniowych. Ponadto, nie jest jasne w jaki sposób konwersja sygnału do postaci zdarzeniowej wpływa na działanie wyuczonego modelu.

Wyżej wspomniane problemy zaadresowano przy pomocy metod analizy statystycznej oraz symulacji numerycznych. Zaproponowano reguły klasyfikacji procesów punktowych przy pomocy metod bayesowskich, a następnie przeanalizowano zbieżność algorytmu do ryzyka bayesowskiego w funkcji liczby przykładów uczących. Sprawdzono również wpływ efektów brzegowych na działanie klasyfikatora opartego o jądrowy estymator gęstości. W kontekście impulsowych sieci zrównoleglono algorytm obliczania wyjścia neuronu reagującego na czas wystąpienia impulsu wejściowego, co znacząco skróciło czas uczenia sieci. Ponadto rozszerzono ów model o możliwość obserwowania i generowania wielu zdarzeń. Zdefiniowano również funkcję kosztu, która umożliwia uczenie sieci syjamskiej bezpośrednio w dziedzinie zdarzeń.

Praktyczną stosowalność tych metod zweryfikowano w kontekście trzech problemów badawczych: identyfikacji botów w mediach społecznościowych, wykrywaniu artefaktów podczas detekcji cząstek promieniowania kosmicznego oraz kategoryzacji pojazdów drogowych. W każdym z tych scenariuszy sprawdzono wpływ reprezentacji sygnału w postaci zdarzeniowej oraz hiperparametrów procesu uczenia na działanie gotowego modelu. Wybór tak zróżnicowanych zastosowań dowodzi dużej uniwersalności opracowanych metod.

List of Acronyms

ANN Artificial Neural Network

API application programming interface

BC-KDE boundary-corrected kernel density estimation

BPTT backpropagation through time

CDF cumulative distribution function

CNN convolutional neural network

CPU central processing unit

CR cosmic ray

CREDO Cosmic-Ray Extremely Distributed Observatory

DVS Dynamic Vision Sensor

DWT discrete wavelet transform

ECG electrocardiography

EMD Earth Mover's Distance

EMG electromyography

IF Integrate-and-Fire

IL inductive loop

ISI interspike interval

IT information technology

ITS Intelligent Transportation System

KDE kernel density estimation

KL Kullback-Leibler [divergence]

k-NN k-nearest neighbors

LIF Leaky Integrate-and-Fire

LSTM long short-term memory

MCC Matthews correlation coefficient

MFCC mel-frequency cepstrum coefficients

MIMO multiple-input, multiple-output

NN neural network

PCA	principal component analysis
RBM	restricted Boltzmann machine
ReLU	rectified linear unit
RKHS	reproducing kernel Hilbert space
RNN	recurrent neural network
ROI	region of interest
R-VMP	resistive [component of the] vehicle magnetic profile
SGD	stochastic gradient descent
SNN	spiking neural network
SOM	self-organizing map
SRM	Spike Response Model
STPD	spike-timing-dependent plasticity
SVM	support vector machine
TEM	time encoding machine
TICA	temporal independent component analysis
VAE	variational autoencoder
VMP	vehicle magnetic profile
VP	Victor-Purpura [distance]
X-VMP	reactive [component of the] vehicle magnetic profile

Contents

Chapter 1

Introduction

1.1 Background

Event-driven systems are often encountered in science and engineering. In such arrangements data points represent temporal sequences of discrete events, with each event composed of three descriptors: a timestamp, an event category, and a value [1, 2]. The latter two pieces of information may be provided implicitly: some systems register only a single type of event, others do not define event values (with a binary indicator of event occurrence or its implied absence). Event data is natural in scenarios such as modeling customer behavior in banking [3, 4] or social networks [5], analyzing the activity of malicious agents in cybersecurity applications [6], log-based anomaly and fault detection [7], and knowledge discovery and health state prediction from electronic health records [8, 9, 10].

Another type of event data arises by discretizing an analog signal into a set of predefined events based on some criteria. Monitoring the activity of neural populations in neuroscientific studies is a prime example of this type of event data [11]. Neurons communicate via an action potential – a sudden change of membrane voltage of a given cell that travels down an axon to propagate the signal to other cells – called a *spike* due to its characteristic shape. Electrodes recording a change in voltage of a neural tissue register a superposition of the activity of different neurons. Identifying individual neurons from a superimposed voltage recording based on their properties (shape, latency) is done using a process called spike sorting [12]. Once complete, the individual signals are represented in an abstract way as sequences of spikes called *spike trains*, highlighting the importance of spikes as the information carrier in neural populations. These sequences encode the state of the network as well as details about the triggering signal (stimulus). This information is known as the neural code [13].

The empirical findings related to the neural code used in human perception can be applied to construct event sensors [14]. In contrast to typical sensors that register a uniformly-sampled signal, they apply data-driven dynamic sampling. This form of encoding has three main advantages over uniformly-sampled data: wide dynamic range, focus on the informative data, and low latency due to asynchronous communication. These principles have been applied to construct neuromorphic vision [15], hearing [16], smell [17] and touch [18] sensors.

Notably, registering event streams using a neuromorphic device is not a prerequisite to obtaining an event dataset. In case such data is unavailable (e.g., due to its proprietary nature), it is possible to transform existing sets of data to an event structure using software simulations [19, 20], or custom hardware-based conversion systems [21]. Software simulations can be used to design event-sampling hardware, or to implement new event-based signal processing algorithms even when the hardware is not yet available. Therefore, it presents the opportunity to conduct a proof of concept study prior to the expensive process of hardware implementation. Conversely, hardware-based conversion systems additionally allow to more realistically approximate the noise associated with signal acquisition, which can result in variations of event latency, missing or even spurious events. In addition to the benefits associated with event data relative to uniformly-sampled data, the signal-to-event dataset conversion paradigm encourages the comparison between typical and event-based algorithms acting on the same underlying signal.

1.2 Motivation

This thesis considers the problem of applying machine learning techniques to event-type data. A core objective of machine learning is to generalize task-solving capabilities, that were learned during the training procedure, to unseen data. To train any machine learning model – be it in a supervised or an unsupervised paradigm – it is necessary to specify a set of features present in the data. However, defining "features" of event data is elusive. Such data is, by definition, sparse and of undefined dimensionality. Take for example a time series of a hospital patient health record:

- samples of laboratory diagnostics are not taken at the exact same time of the day (leading to data that is non-uniformly distributed in time),
- some tests are rarely repeated (meaning that data points lack complete information, which causes data sparsity).

Admittedly, event data can be modified in a way that allows it to be processed by classical machine learning methods. Doing so circumvents its event nature (e.g., by introducing additional features that encode time-of-event or no change or data resampling), which in turn discards the succinctness inherently present in event data. And so, to avoid data redundancy, we need a computational model that is dedicated to sparse, event-based data representations.

In light of these observations, the focus of this work is on two different approaches to training machine learning models on spike train data. A classical solution based on the theory of point processes attempts to describe the inherent randomness of the observed spiking patterns. Despite extensive use in the computational neuroscience field of research to model the underlying biological processes, this methodology is surprisingly under-developed in terms of its use in solving supervised machine learning problems. Given this research gap, it is imperative to develop fundamental training rules on simple spike train datasets, as well as assess the bounds on model performance with respect to the observation period length and training dataset size.

The other approach considered in this thesis are the artificial spiking neural networks (SNN), hailed as the new, biologically-inspired generation of artificial neural networks. This methodology has emerged quite recently by adapting deep learning concepts to incorporate sophisticated computational neuron models. In contrast to the point-process-theory-based approach, there already exist multiple streams of research dedicated to applying these models to the supervised and unsupervised learning problems. The vast majority of the SNN models are trained on traditional von Neumann architecture, and so it is necessary to simulate the evolving state of the entire network over time. Ideally, the training process itself should mimic the sparsity of event data – by considering the network state only at time-instants related to event occurrence – but it is not the case in such training simulations. There is a pressing need to design both signal propagation and training rules that conform to these requirements.

By analyzing these two approaches in parallel it is possible to assess their applicability to practical problems, noting the degree of customization and robustness that they exhibit. And so, in addition to the fundamental properties of these models, it is crucial to develop a methodology that evaluates the performance of event-centric machine learning models in general. Furthermore, it is equally important to consider the impact of event-triggered data acquisition on model performance. This stems from the fact that some spike train data can be obtained by event-based sampling of the underlying analog signal, or by conversion of its digital version. Understanding how data encoding impacts the downstream model performance should be a factor that contributes to the broader measurement system design scope.

1.3 Dissertation outline

The main results of this study are described in four parts:

- Chapter 2 considers the classical approach of event sequence classification based on the theory of point processes. We derive an optimal classification rule in terms of intensity functions and propose a general class of plug-in nonparametric rules from multiple replications of spiking processes. A kernel classifier is introduced as an example of the latter approach. These findings are supported by a set of finite sample simulation studies. We show that the proposed kernel rule converges to the optimal Bayes classifier as the number of training examples increases. Additionally, we assess the impact of boundary effects on the classifier performance when the length of the event observation interval is sufficiently short. Lastly, the proposed approach is applied to real data in a Twitter bot classification task. In this specific task the observation window is quite long, with events occurring at timescales differing by many orders of magnitude. The results regarding the convergence of the kernel classifier to the optimal classification rule for point processes described in this Chapter was presented in [22].

- Chapter 3 focuses on applying spiking neural networks to machine learning problems for event data. It discusses an existing single-spike time-to-first-spike SNN model and identifies limitations of this model that stem from its assumptions. Based on this discussion, we propose several modification that alleviate these shortcomings by reducing its processing time, stabilizing training dynamics, allowing a finer control over the spiking activity, and proposing a framework for processing signals varying over time. This last point is described in terms of iterating over a space of events, which stands in contrast to other SNN models that iterate over discretized time. Finally, the performance of the proposed SNN is evaluated on the aforementioned Twitter bot classification task. Reusing the same dataset allows us to compare the point process and SNN approaches, and discuss their applicability to other tasks.
- Chapter 4 shows how adapting spike train similarity measures used in neuroscientific research allows us to construct a Siamese SNN. This leads to the first-ever Siamese network trained end-to-end in the spiking domain. The proposed methodology is evaluated on two image datasets. First, we use the MNIST digit dataset, known to be a relatively simple benchmark that promotes conducting reproducibility studies. The impact of the number of output neurons on the classifier performance, network spiking activity and prediction latency is assessed. Secondly, the Siamese SNN is applied to the problem of differentiating signal from artefacts in the context of cosmic ray detection on images taken by modern smartphones. This peculiar dataset is a representative example of the Siamese SNN applicability as the images themselves contain a very small informative region of interest compared to the total image area, making the image data sparse. Our findings regarding the Siamese SNN model properties on the MNIST and CREDO data were published in [23, 24].
- Chapter 5 presents a case-study that applies the SNN model to vehicle type identification problem. Various event-triggered sampling schemes with the ability to convert a

multivariate time series into event sequences are discussed. We select a subset of such methods and show how to determine the parameters of the sampling scheme such that information important for the downstream classification task is preserved. Moreover, we train the SNN models on the produced event sequences, and relate the model classification performance of trained SNN models with the number of events produced by the selected event-triggered sampling schemes. This allows to assess the different conversion methods in terms of their efficiency.

Note that each subsequent chapter builds upon the results presented in the earlier sections. Finally, Chapter 6 presents a summary of the conducted research, as well as the proposed directions for future work.

Chapter 2

Bayes Rules for Spike Train Data Classification

Spike train data find a growing list of applications in computational neuroscience, imaging, streaming data and finance. Statistical analysis of spike trains is based on various probabilistic and neural network models. The statistical approach relies on parametric or nonparametric specifications of the underlying model. The goal of this Chapter is to develop a rigorous classification methodology based on the Bayes theory of classification [25] for the two-class classification problem for a class of spike train data characterized by nonparametricaly specified intensity functions. This strategy can be applied to research problems where event occurrence is the primary information carrier [5, 26].

In Section 2.2 we derive the optimal Bayes classification rule in terms of class intensity functions for such processes, as well as assess the limit behavior of the Bayes rule with respect to the increasing length of the observation interval. The theoretical findings related to the Bayes risk convergence are supported by simulated data study described in Section 2.2.1. In Section 2.3 a general plug-in nonparametric classification rule from multiple replications of spiking processes is proposed. This is followed in Section 2.4 by the asymptotically optimal result of the kernel classification rule, i.e., the convergence of the kernel rule to the Bayes rule. This outcome can be considered as the counterpart of the result in [27] concerning the classical plug-in nonparametric classification rules defined in the finite-dimensional Euclidean space. The obtained results are supported by a finite sample simulation studies, with the impact of the number of training examples and the sequences observation time assessed in Section 2.4.1, whereas Section 2.4.2 focuses on the impact of boundary effects on the classifier performance. Lastly, the proposed approach is applied to real data in a Twitter bot classification task (Section 2.5).

2.1 Introduction

2.1.1 Spike train probabilistic modeling

The mathematical framework for describing event data is the point process theory [28]. A temporal spiking process $\{N(t), t \geq 0\}$ consists of a sequence of random times $\{t_i\}$ of isolated events in time such that $N(0) = 0$. The process $N(t)$ can be defined by the counting function

$$N(t) = \sum_i \mathbf{1}\left(t_i \leq t\right) , \tag{2.1}$$

which is the number of events in $[0, t]$. Assuming that the process is observed on a time window $[0, T]$ – which is certainly the case in practical machine learning applications in which the analysis is conducted on a finite sample of data points – the process $N(t)$ can be represented by a variable-length vector $\mathbf{X} = [t_1, \ldots, t_N; N]$, where $0 < t_1 < \cdots < t_N < T$ are the event times, and $N = N(T)$. Writing $[t_1, \ldots, t_N; N]$ we emphasize the fact that the data vector consists of two parts: the occurrence times $\{t_1, \ldots, t_N\}$ and N being the number of events in $[0, T]$. The former is the continuous part of the vector $\mathbf{X}$, whereas the latter is its discrete part. Therefore, the spike train data are characterized by variable-length continuous-discrete observation vectors. The spike train can be equivalently described in terms of interarrival times with event occurrence times $\{t_1, t_2, \ldots, t_N\}$ substituted with their differences $\{t_1, t_2 - t_1, \ldots, t_N - t_{N-1}\}$ also called interspike intervals (ISI) or waiting times between successive events.

Temporal point processes can also be described in terms of an intensity function $\lambda(t)$ which describes the local (in time) arriving rate of events. In general, the intensity function is

$$\lambda(t|H_t) = \lim_{\Delta t \to 0} \frac{1}{\Delta t} \mathbf{P}\left(N(t + \Delta t) - N(t) = 1|H_t\right) , \tag{2.2}$$

with H_t being the history of all events which occurred before time t. The degree of dependence on the process history varies between different point process types. A simple example of a model that is conditioned on the entire history is the Hawkes process that models self-exciting phenomena, i.e., the occurrence of an event increases the probability that another event will occur in the near future [29]. When $\lambda(t|H_t) = \lambda(t|t_{N_t})$, i.e., the instantaneous intensity rate depends on the time of the last spike before t, and when the ISI are i.i.d., the process is called a renewal process. Finally, a temporal process that is memoryless (independent of its history), has i.i.d. ISI, and satisfies the property

$$\mathbb{E}\left[N(T)\right] = \int_0^T \lambda(u) du \tag{2.3}$$

is a nonhomogeneous Poisson process. Note that the point process intensity need not be a deterministic function, with Cox process being an important example that describes point processes in which the intensity measure is itself a Poisson process [30].

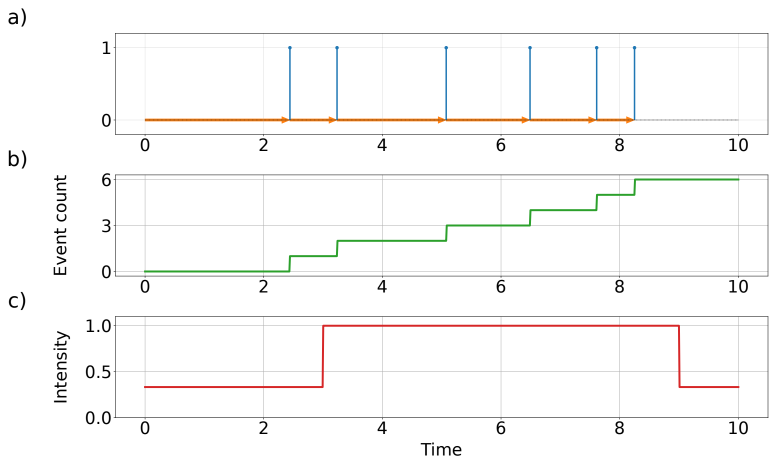


Figure 2.1: Descriptors of a point process for a given spike train. a) Event occurrence times $\{t_1, t_2, \ldots, t_N\}$ (vertical, in blue) and interspike intervals $\{t_1, t_2 - t_1, \ldots, t_N - t_{N-1}\}$ (horizontal, in orange). b) Counting process. c) Point process intensity function.

Figure 2.1 summarizes the different ways to describe a single point process realization (i.e., the spike train). In fact, different spike train descriptors are used in the methodologies presented in subsequent Sections and Chapters of this Thesis. The rest of this Chapter shows how point process realizations can be classified by applying intensity function estimation from event times. Chapter 3 focuses on the underlying event occurrence time representation of the point process processed by artificial spiking neurons exhibiting the memoryless property. It also provides further context for the application of the theory of point processes in neuroscience. Lastly, Chapter 4 shows how the counting function can be used to analyze the similarity of spike trains produced by the proposed spiking neural network.

2.1.2 Supervised classification: the point processes approach

Before moving on to the description of classification methods designed and evaluated in this Chapter, it is important to note that there are actually two different types of point process "classification" problems that can be studied. Given that this field of research for temporal point processes – i.e., the focus of our work – is surprisingly poorly developed, we shall include additional examples derived from spatial point processes analyses to elucidate different methods, as well as potential applications of such research.

The first type of classification problem concerns itself with assigning a class label to every event separately. In such type of event classification it is assumed that the observed events were generated by two or more point processes superimposed on one another. In spatial point process analysis this approach can be used to determine the position of fault lines from the earthquake occurrence data [31], estimate a minefield [32, 33], or reject artefacts when dating glacial environments [34]. These methods are additionally accompanied by local clutter removal using a k-Nearest Neighbor (k-NN) classifier. Event classification can also be used to analyze data exhibiting a temporal structure. For example, to infer the sentiment of comments in Twitter discussion threads [35], or to find outliers in an event stream [36].

On the other hand, point process "classification" can also mean predicting the class identity for the entire event sequence or the complete set of events (in the temporal and spatial contexts, respectively). Regardless of whether the data exhibits temporal or spatial structure, three main method types can be identified. The most common are distance-based approaches [37, 38, 39, 40, 41] as they do not attempt to use the probabilistic model of the underlying point process prior to classification (although the classifier performance can be improved by choosing a distance measure that is appropriate for a given data distribution). Conversely, the nonparametric [40, 42] and maximum likelihood [43, 44] estimation methods model the probabilistic properties of the point processes based on the observed event sequences (or event sets). In all these methods it is important to increase the total number of events observed in a given temporal or spatial interval. Typically it is done by using the *replicates* of the spiking process – different realizations of the same point process.

This Thesis is entirely focused on the second scenario of classifying entire spike train sequences to individual point process categories.

2.2 Bayes classification rule

The goal of this Section is to develop a rigorous classification methodology for temporal Poisson processes based on the Bayes theory of classification [25]. We consider the two-class classification problem (which can be extended to the multi-class problem) where class labels are denoted as ω_1, ω_2 with the prior probabilities π_1, π_2, respectively. Let us assume that the point process is observed on the time window $[0, T]$ and is characterized by a non-random intensity function $\lambda(t)$ that is defined for all $t \geq 0$, such that (2.3) describes the the average number of events in $[0, T]$. For example, consider the intensity function

$$\lambda(t;\phi) = 1.6 + \cos\left(\frac{\pi}{4\sqrt{3}}t + \phi\right) + 0.5\cos\left(\frac{\pi}{3\sqrt{2}}t + \frac{\pi}{4} + \phi\right). \tag{2.4}$$

Various choices of ϕ define $\lambda_1(t)$, $\lambda_2(t)$. One can repeatedly sample a new spike train from point processes according to $\lambda_1(t)$ or $\lambda_2(t)$. As mentioned in the previous Section, the task of classification considered in this work is to analyze each individual spike train and determine whether it was generated by a point process specified by either $\lambda_1(t)$ or $\lambda_2(t)$. This machine learning problem is illustrated in Figure 2.2.

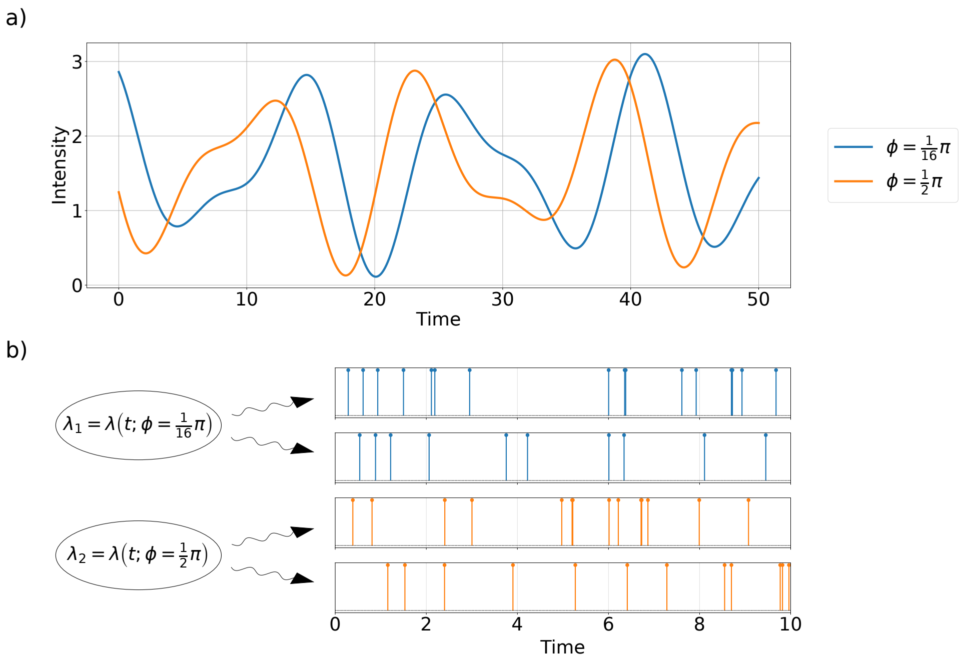


Figure 2.2: a) A pair of intensity functions (2.4) parameterized by ϕ. b) Sampling new spike trains according to the intensity function.

In order to form the optimal Bayes rule we recall the well-known result [45] on the joint occurrence density of the spike train $\mathbf{X} = [t_1, \ldots, t_N; N]$

$$f(\mathbf{x}) = \prod_{i=1}^{N} \lambda(t_i) \exp\left(-\int_0^T \lambda(u)du\right) \tag{2.5}$$

for $N = N(T) \geq 1$, whereas if $N = 0$ then $f(x) = \exp\left(-\int_0^T \lambda(u)du\right)$. It is worth noting that (2.5) is a continuous-discrete distribution and by virtue of (2.5) the marginal density of the occurrence times $\{t_1, \ldots, t_N\}$ for $N \in \{0, 1, \ldots\}$ is given by

$$\sum_{n=0}^{\infty} f\left(t_1, \ldots, t_N; N = n\right) = \exp\left(-\int_0^T \lambda(u)du\right) + \exp\left(-\int_0^T \lambda(u)du\right) \sum_{n=1}^{\infty} \prod_{j=1}^{n} \lambda(t_j), \tag{2.6}$$

which is defined over the simplex regions $\mathbb{C}_n = \{(t_1, \ldots, t_n) : 0 \leq t_1 \leq \ldots \leq t_n \leq T\}$, $n = 1, 2, \ldots$. The formula in (2.6) defines the proper density over $\{\mathbb{C}_n\}$, i.e., we have

$$\exp\left(-\int_0^T \lambda(u)du\right) + \exp\left(-\int_0^T \lambda(u)du\right) \sum_{n=1}^{\infty} \int_{\mathbb{C}_n} \prod_{j=1}^{n} \lambda(t_j) dt_1 \cdots dt_n = 1\,. \tag{2.7}$$

In the context of the classification problem the occurrence densities in (2.5) will be denoted $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ depending on whether $\mathbf{X}$ comes from class ω_1 (denoted as $\mathbf{X} \in \omega_1$) or if $\mathbf{X} \in \omega_2$, respectively. The corresponding class intensities $\lambda_1(t), \lambda_2(t)$ are nonnegative functions defined on $[0, \infty)$. Then using (2.5), one can form the optimal Bayes rule ψ_T^*: $\mathbf{X} \in \omega_1$ if

$$\prod_{i=1}^{N} \frac{\lambda_1(t_i)}{\lambda_2(t_i)} \exp\left(\int_0^T [\lambda_2(u) - \lambda_1(u)]\, du\right) \geq \frac{\pi_2}{\pi_1}\,. \tag{2.8}$$

assuming that $N \geq 1$ and $\exp\left(\int_0^T [\lambda_2(u) - \lambda_1(u)]\, du\right) \geq \frac{\pi_2}{\pi_1}$ if $N = 0$. Clearly, if the reverse inequality in (2.8) holds, then we classify $\mathbf{X}$ to ω_2. The log transform of (2.8) gives the alternative convenient form of the rule ψ_T^*, i.e., $\mathbf{X} \in \omega_1$ if

$$\sum_{i=1}^{N} \log\left(\frac{\lambda_1(t_i)}{\lambda_2(t_i)}\right) \geq \gamma\,, \tag{2.9}$$

where $\gamma = \int_0^T [\lambda_1(u) - \lambda_2(u)]\, du + \log\left(\frac{\pi_2}{\pi_1}\right)$.

For our further considerations it is useful to represent the class intensity functions in terms of the so-called intensity factor τ and shape function $p(t)$ [46]. Thus, let $\lambda_1(t) = \tau_1 p_1(t)$, $\lambda_2(t) = \tau_2 p_2(t)$, where

$$\tau_i = \int_0^T \lambda_i(u) du, \;\; p_i(t) = \lambda_i(u)/\tau_i, \;\; i = 1, 2\,. \tag{2.10}$$

Clearly $p_1(t), p_2(t)$ are probability density functions on $[0, T]$ that reflect the functional shape of the intensity functions. On the other hand the intensity factor measures the average number of events in the observation interval $[0, T]$. Equation (2.10) allows us to represent the classification problem in terms of the class intensity factors and shape densities. In fact, using (2.10), we can rewrite the rule in (2.9) as follows, ψ_T^*: $\mathbf{X} \in \omega_1$ if

$$\sum_{i=1}^{N} \log\left(\frac{p_1(t_i)}{p_2(t_i)}\right) \geq \eta\,, \tag{2.11}$$

where $\eta = \tau_1 - \tau_2 + N \log\left(\frac{\tau_2}{\tau_1}\right) + \log\left(\frac{\pi_2}{\pi_1}\right)$. The Bayes rule ψ_T^* in (2.11) will be written as $W_T(\mathbf{X}) \geq \eta_T$ emphasizing the fact the vector $\mathbf{X}$ is observed within the time window $[0, T]$.

The risk associated with the rule $\psi_T^*(\mathbf{x})$ in (2.11) is defined as $\mathbf{R}_T^* = \mathbf{P}(\psi_T^*(\mathbf{X}) \neq Y)$ and is referred as the Bayes risk. Here $Y \in \{\omega_1, \omega_2\}$ is the true class label of $\mathbf{X}$. For our future studies we express the Bayes risk in terms of the decision function $W_T(\mathbf{X})$, i.e., we write

$$\mathbf{R}_T^* = \mathbf{P}\left(\mathbf{W}_T(\mathbf{X}) \geq \eta_T | \mathbf{X} \in \omega_2\right) \pi_2 + \mathbf{P}\left(\mathbf{W}_T(\mathbf{X}) < \eta_T | \mathbf{X} \in \omega_1\right) \pi_1\,. \tag{2.12}$$

It is an important question to evaluate the Bayes risk. This includes various bounds on $\mathbf{R}_T^*$ and the behavior of $\mathbf{R}_T^*$ as a function of T. In order to do so, we need to introduce some assumptions.

1) First, let us note that the log-ratio $\log\left(\frac{\lambda_1(t)}{\lambda_2(t)}\right)$ or equivalently $\log\left(\frac{p_1(t)}{p_2(t)}\right)$ is generally an unbounded function. To prevent a singularity it suffices to assume that the class intensities $\lambda_1(t)$, $\lambda_2(t)$ are bounded away from zero. This is commonly true for intensity functions. All these restrictions can be formalized by assuming that there exist positive numbers δ and C such that

$$\mathrm{A1}: \ 0 < \delta \leq \lambda_i(t) \leq C, \ \ i = 1, 2, \ \ \text{for all} \ \ t \geq 0\,. \tag{2.13}$$

We refer to [47, 48] for some weaker conditions for the existence of the aforementioned log-ratio.

2) Furthermore, we need to put some condition on the growth of the class intensity functions. Hence, let us assume that there exists positive number d such that

$$\mathrm{A2}: \ \frac{1}{T}\int_0^T \lambda_i(u)du \longrightarrow d, \ \ i = 1, 2 \ \ \text{as} \ \ T \longrightarrow \infty\,. \tag{2.14}$$

The meaning of this condition is that the number of events from the each class increases linearly with T. It is worth mentioning that the condition in A2 does not hold if the intensity functions are integrable on $(0, \infty)$. This, e.g., takes place if $\lambda_1(t), \lambda_2(t)$ have a compact support.

Then under the assumptions A1 and A2, it can be shown that the Bayes risk (2.12) tends to zero as $T \longrightarrow \infty$. Hence, we have the following result.

Theorem 1. Let assumptions A1 and A2 hold. Then

$$\mathbf{R}_T^* \longrightarrow 0 \ \ \text{as} \ \ T \longrightarrow \infty \ \ \text{with the rate} \ \ \mathcal{O}\left(1/T\right)\,.$$

The proof of Theorem 1 is given in [49].

The convergence of the Bayes risk $\mathbf{R}_T^*$ to zero is mostly determined by the condition in A2. This is due to the fact that the class intensity functions $\lambda_1(t), \lambda_2(t)$ must grow with increasing T. If A2 does not hold, e.g, if $\lambda_1(t), \lambda_2(t)$ are compactly supported then the convergence of $\mathbf{R}_T^*$ to zero is impossible. It is also possible to have a nonzero limit Bayes risk if the class intensity functions overlap for some large value of t, i.e., if $\lambda_1(t) = \lambda_2(t)$ for $t \geq T_0$ for some large T_0.

It is worth mentioning that the asymptotic optimality does not hold if one observes a single long realization of the underlying spiking process. In fact, the intensity estimation problem for spiking processes does not fall into the classical large-sample, smaller-distance-between-sample-points framework as the point process is casual in time [50]. Hence, for a fixed observation interval one must increase the number of events. This can be achieved by either scaling the intensity function or by using the replicates of the spiking process (i.e., different realizations of the same spiking process). The former approach is based on the multiplicative intensity model due to Aalen [51], whereas the latter one (used in this work) is the standard machine learning strategy, where the replicates form the training set.

2.2.1 Bayes rule convergence for simulated data

The purpose of the simulated data study is to illustrate the findings expressed in Theorem 1 for some set of point process intensity function pairs. Doing so requires not only computing the Bayes risk $\mathbf{R}_T^*$ but also to show that $\mathbf{R}_T^* \to 0$ as $T \to \infty$. To establish this, we resort to estimating $\mathbf{R}_T^*$ for different values of T using a Monte Carlo simulation. First, let us focus on answering the question: given an intensity function pair $\lambda_1(t)$, $\lambda_2(t)$, how many spike trains are needed to estimate the Bayes risk (2.12)?

Simulation 1. Denote $\mathbf{D}_P = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_P, Y_P)\}$ as a sample of P independent observations of the spiking processes. Here $\mathbf{X}_j$ is the variable-length vector, i.e., $\mathbf{X}_j = \left[t_1^{[j]}, \ldots, t_{N^{[j]}}^{[j]}; N^{[j]}\right]$ and $Y_j \in \{\omega_1, \omega_2\}$, where $N^{[j]} = N^{[j]}(T)$. Without the loss of generality, we assume $\pi_1 = \pi_2 = 0.5$, which implies that the factor $\frac{P}{2}$ represents the number of examples sampled per class. Then, the simulation procedure can be summarized as follows:

1) Specify the number of simulation repetitions ξ; intensity function pair $\lambda_1(t)$, $\lambda_2(t)$; sequence observation interval $[0, T]$; and the total number of sampled spike trains P.
2) Construct a simulation grid $\{P_1, P_2, \ldots, P_K\} \in (0, P]$ specifying $k \in \{1, \ldots, K\}$ points at which to evaluate the classification rule.
3) For every repetition of the experiment:
    a) Simulate $\frac{P}{2}$ spike trains according to the intensity function $\lambda_1(t)$, and $\frac{P}{2}$ examples according to $\lambda_2(t)$.
    b) At each point P_k of the grid:
        i. Select a subset of spike trains that is valid for the current simulation evaluation point, i.e., $\mathbf{D}_{P_k} = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_{P_k}, Y_{P_k})\} \subset \mathbf{D}_P$.
        ii. Compute the classifier prediction according to the rule (2.11) for every pair $(\mathbf{X}_j, Y_j) \in \mathbf{D}_{P_k}$.
        iii. Compute empirical risk (2.12) based on classifier decisions for the P_k spike trains.
4) Compute the mean and standard deviation of the empirical risk over ξ repetitions.

Then the mean empirical risk over ξ repetitions $\mathbb{E}\left[\mathbf{R}_{T,P}\right]$ is the Monte Carlo estimate of the Bayes risk $\mathbf{R}_T^*$ for fixed T. Note that for every repetition of the experiment, this simulation protocol reuses some sequences whenever the effective sample size P_k increases. This simulates the scenario in which the classifier has to change its historic prediction as the number of

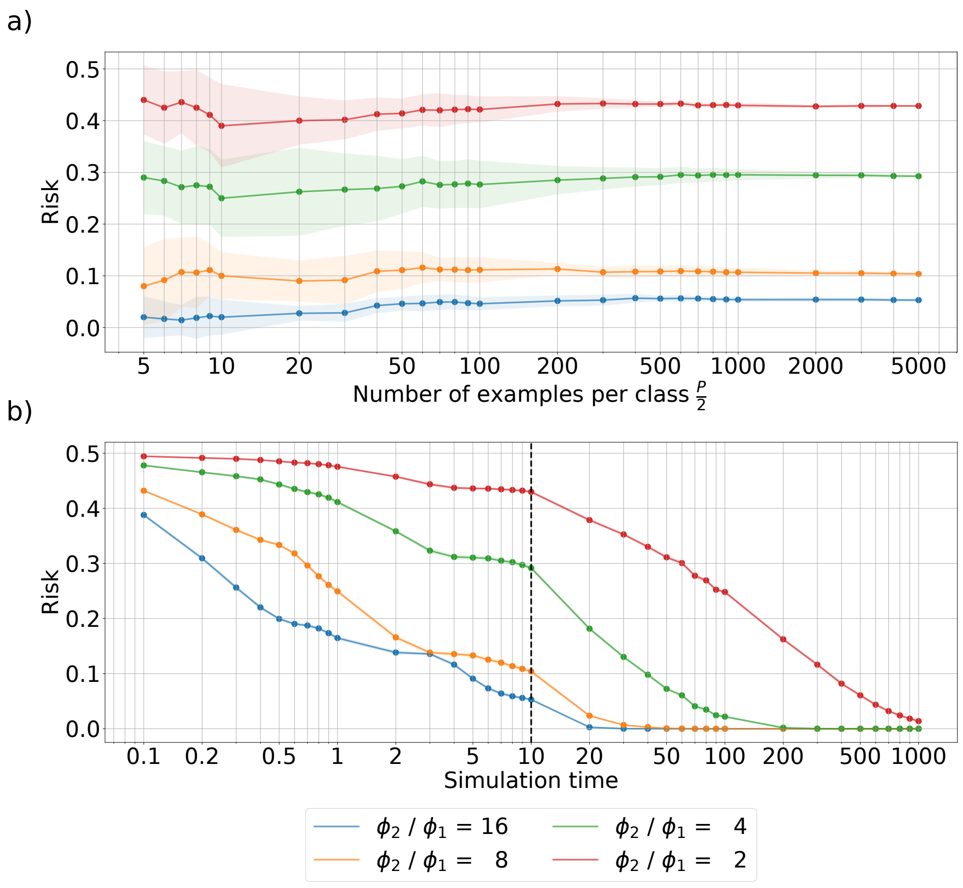


Figure 2.3: The results of Monte Carlo simulations for the two-class classification problem for several intensity function pairs (2.15) parameterized by ϕ. a) Risk versus P for a fixed $T = 10$. b) Bayes risk $\mathbf{R}_T^*$ versus T. Dashed vertical line at $T = 10$ denotes the simulation space slice presented in the top row.

observation increases. An alternative, incorrect, approach would be to sample P_k new examples at every point of the simulation grid.

Figure 2.3a presents the results of Simulation 1 for $T = 10$ conducted for the intensity function

$$\lambda(t;\phi) = 1.6 + \cos\left(\frac{\pi}{4\sqrt{3}}t + \phi\right) + 0.5\cos\left(\frac{\pi}{3\sqrt{2}}t + \frac{\pi}{4} + \phi\right). \tag{2.15}$$

Various choices of ϕ define $\lambda_1(t;\phi_1)$, $\lambda_2(t;\phi_2)$. It is clear that as $\phi_2 \longrightarrow \phi_1$ the classification problem becomes more difficult to solve, given that the spike trains sampled from $\lambda_1(t;\phi)$, $\lambda_2(t;\phi)$ become more similar to one another. The sequence observation time T was selected such that in neither scenario the empirical risk is zero in order to show that $\mathbb{E}\left[\mathbf{R}_{T,P}\right] \longrightarrow \mathbf{R}_T^*$ as $P \longrightarrow \infty$. The results also illustrate that $\mathbf{R}_T^*$ is higher for difficult classification problem.

The aforementioned results established that for $T = 10$ the Monte Carlo simulation classification rule risk is reasonably close to the Bayes risk for $P = 10^4$ for all intensity function pairs parameterized by ϕ. We set $P = 10^4$ in all subsequent experiments. Next, we wish to show that $\mathbf{R}_T^* \to 0$ as $T \to \infty$.

Simulation 2. The simulation procedure can be summarized as follows:

1) Specify the number of simulation repetitions ξ; intensity function pair $\lambda_1(t)$, $\lambda_2(t)$; sequence observation interval $[0, T]$; and the total number of sampled spike trains P (large enough that the risk estimate converges to the Bayes risk according to Simulation 1).
2) Construct a simulation grid $\{T_1, T_2, \ldots, T_K\} \in [0, T]$ specifying $k \in \{1, \ldots, K\}$ points at which to evaluate the classification rule.
3) For every repetition of the experiment:
    a) Simulate $\frac{P}{2}$ spike trains according to the intensity function $\lambda_1(t)$, and $\frac{P}{2}$ examples according to $\lambda_2(t)$.
    b) At each point T_k of the grid:
        i. Select a subset of events that is valid for the current simulation evaluation point, i.e., find $\mathbf{X}_{j,T_k} = \left[t_1^{[j,T_k]}, \ldots, t_{N^{[j,T_k]}}^{[j,T_k]}; N^{[j,T_k]}\right]$ such that $t_{N^{[j,T_k]}}^{[j,T_k]} \leq T_k$, for all $\mathbf{X}_j \in \mathbf{D}_P$.
        ii. Compute the classifier prediction according to the rule (2.11) for every pair $(\mathbf{X}_{j,T_k}, Y_j) \in \mathbf{D}_P$.
        iii. Compute empirical risk (2.12) based on the classifier decisions for the P spike trains.
4) Compute the mean and standard deviation of the empirical risk over ξ repetitions.

Then the mean empirical risk over ξ repetitions $\mathbb{E}\left[\mathbf{R}_{T,P}\right]$ is the Monte Carlo estimate of the Bayes risk. Note that by reusing an existing sequence whenever the effective observation interval $[0, T_k]$ increases we simulate the scenario in which the classifier has to update its historic prediction due to new evidence.

Figure 2.3b presents the results of Simulation 2 conducted for the intensity function (2.15) for different values of the ratio ϕ_2/ϕ_1. The curves computed over ξ repetitions of the experiment lack variability which confirms that $P = 10^4$ is enough to estimate the Bayes risk for this intensity function. The results also show that the Bayes risk tends to zero as T gets larger. The slowest decay of $\mathbf{R}_T^*$ is seen for very close intensities $\phi_2/\phi_1 = 2$ (in red), whereas the fast rate of convergence is observed for distant intensities $\phi_2/\phi_1 = 16$ (in blue).

To strengthen our analysis, we opt to apply Simulations 1-2 to additional classes of intensity functions. The functions used are summarized in Table 2.1. Note that all of them meet assumptions A1 and A2 and also they do not overlap for $t \geq T_0$ for some large T_0. The simulation results for binary classification problems constructed from these intensity functions is presented in Figure 2.4. Each row corresponds to a different intensity function pair. Left

Table 2.1: Additional set of intensity functions used in the simulated data study for the Bayes risk convergence.

a)	$\lambda(t;r) = r$
b)	$\lambda(t;r) = (r+1) + r\cos\left(\frac{2\pi}{30}t + 2.6\right) + \cos\left(\frac{2\pi}{28}t + 4.5\right)$
c)	$\lambda(t;\phi) = \left[3.1 + 3\cos\left(\frac{\pi}{3\sqrt{2}}t + \phi\right)\right]^{1/2}$
d)	$\lambda(t;\phi) = 1.3\exp\left[\cos\left(\frac{\pi}{3\sqrt{2}}t + \phi\right)\right]$
e)	$\lambda(t;\phi) = 0.1 + 0.5 \mod [t + \phi, 2\pi]$

column presents a condensed version of the result related to the convergence of Monte Carlo risk estimate to the Bayes risk versus P first shown in Figure 2.3a. Boxplots summarize how quickly each simulation run achieved the Bayes risk. Based on these results it was determined that $P = 10^4$ is enough to estimate the Bayes risk in the Simulation 2. The Bayes risk convergence plots on the right provide further evidence that $\mathbf{R}^*_T \longrightarrow 0$ as $T \longrightarrow \infty$, with the rate of convergence depending on the difficulty of the classification problem.

Let us briefly show a counter-example for which the Bayes risk is nonzero $\forall\, T > 0$. Consider the intensity function

$$\lambda(t;c,d) = c\exp\left[-d\,(t-0.5)^2\right] \,, \tag{2.16}$$

which clearly does not satisfy the assumptions A1 and A2. While the intensity function has an infinite support, in practice it is extremely unlikely for events to occur outside of some narrow time interval. Figure 2.5 the results of Simulation 2 for several combinations of parameters c, d. In all scenarios risk decreases for $t \in [0,1]$ which corresponds to the domain of (2.16) in which the vast majority of events occur. For $t > 1$ almost no new events can be observed, therefore the Bayes risk reaches a plateau.

2.3 Plug-in classification rules

In practice one does not know the true class intensities functions and must rely on some training data in order to form a data-driven classification rule. This work applies the plug-in strategy to design a classifier, i.e., the classifier that is the empirical counterpart of the optimal Bayes rule in (2.11). We have already pointed out that the single-sample based intensity function estimate cannot be consistent unless there is a certain multiplicative mechanism that makes the intensity function increase, see [51] for the multiplicative intensity model approach. Therefore, in this

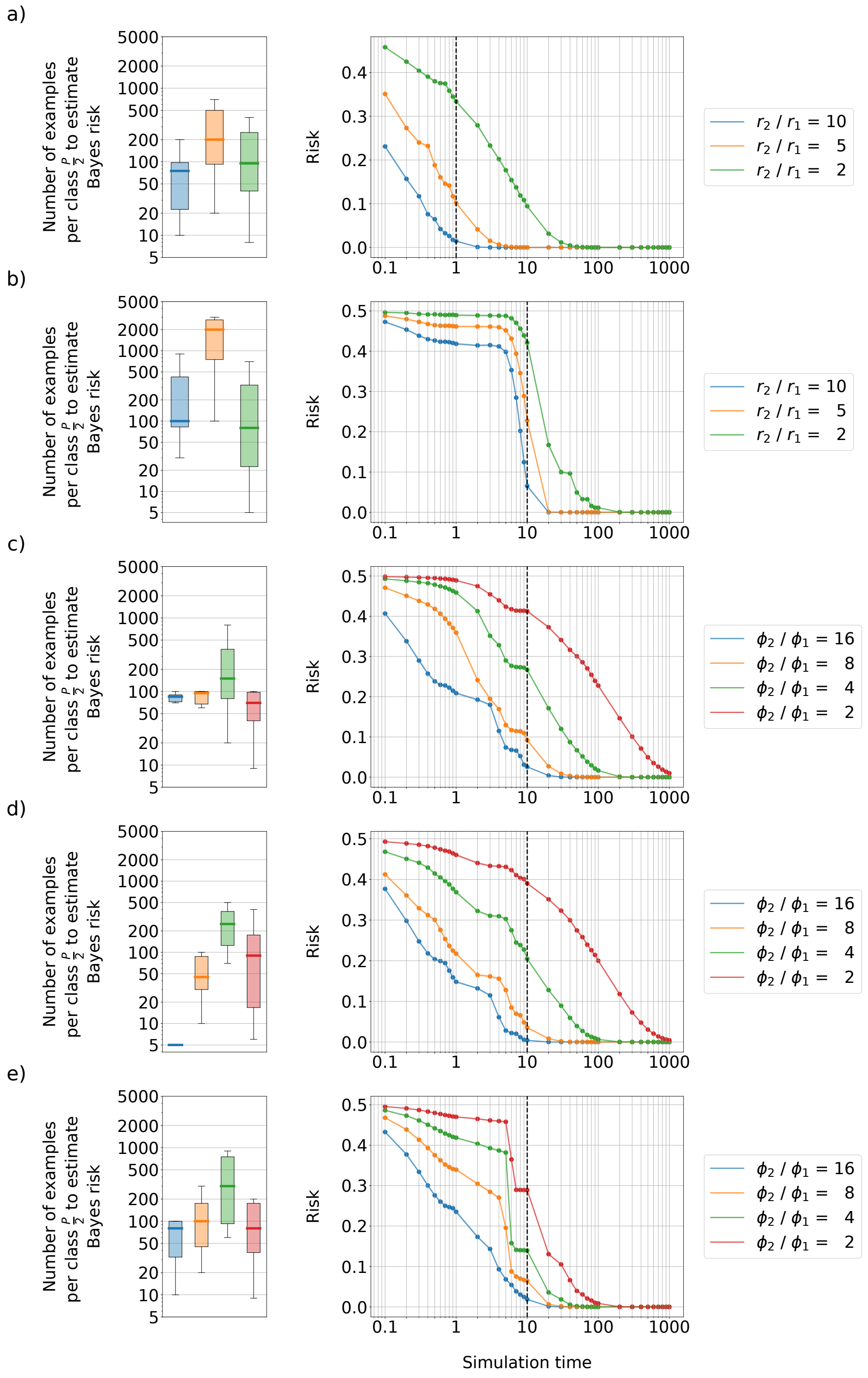


Figure 2.4: The results of Monte Carlo simulations for the two-class classification problem for different intensity functions. Each row corresponds to a different function presented in Table 2.1. Left column: boxplots summarizing how quickly the simulation over P converges to Bayes risk for fixed T. Right column: Bayes risk $\mathbf{R}_T^*$ versus T. Dashed vertical line at $T = 1$ (first row) and $T = 10$ (other rows) denote the simulation space slice presented in the left column.

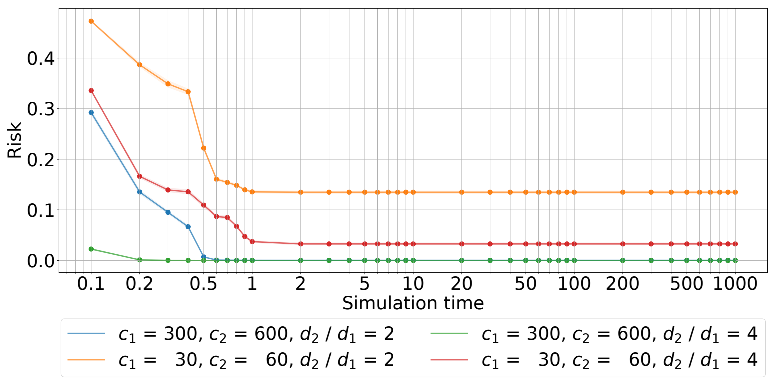


Figure 2.5: Bayes risk $\mathbf{R}_T^*$ versus T for the two-class classification problem specified by intensity functions (2.16) parameterized by c, d.

thesis we consider the intensity estimation model based on the increasing number of replicates of the class spiking processes.

Hence, let $\mathbf{D}_L = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_L, Y_L)\}$ be the learning sequence as a sample of L independent observations of the spiking processes. Recall from Section 2.2.1 that $\mathbf{X}_j$ is the variable-length vector, i.e., $\mathbf{X}_j = \left[t_1^{[j]}, \ldots, t_{N^{[j]}}^{[j]}; N^{[j]}\right]$ and $Y_j \in \{\omega_1, \omega_2\}$, where $N^{[j]} = N^{[j]}(T)$ and $j = 1, \ldots, L$. Hence, all data are measured in the fixed time window $[0, T]$. Let L_1, L_2 be the number of training data of classes ω_1 and ω_2, respectively.

We wish to form the plug-in classification rule based on the optimal decision given in (2.11). This requires estimating the class intensity functions $\lambda_1(t), \lambda_2(t)$, or equivalently the shape densities $p_1(t), p_2(t)$ and the corresponding intensity factors τ_1, τ_2. It is known that the prior probabilities can be estimated by $\widehat{\pi}_1 = L_1/L$ and $\widehat{\pi}_2 = L_2/L$. In order to estimate $(\tau_i, p_i(t))$ one can begin with the use of a single sample $\mathbf{X}_j$. Note that $\mathbb{E}\left[N^{[j]}|\omega_i\right] = \tau_i$ and one can form the unbiased estimate of τ_i as $\widehat{\tau}_i^{[j]} = N^{[j]}$. However, Var $\left[N^{[j]}|\omega_i\right] = \tau_i$ and this is an inconsistent estimate of τ_i. The latter fact results from the local Poisson behavior of the spiking process [49]. Nevertheless, the aggregation of $\widehat{\tau}_i^{[j]}$ leads to consistent estimate of τ_i for the increased size of the training set. Hence, let

$$\widehat{\tau}_i = \frac{1}{L_i} \sum_{j=1}^{L} N^{[j]} \mathbf{1}(Y_j = \omega_i) \tag{2.17}$$

be an estimate of τ_i, $i = 1, 2$. It is easy to show that $\widehat{\tau}_i$ is a consistent estimate of τ_i. This results from the fact that $\mathbb{E}\ [\widehat{\tau}_i] = \tau_i$ and Var $[\widehat{\tau}_i] = \mathcal{O}\left(1/L\right)$. In an analogous way we can deal with the problem of estimating $p_i(t)$. Let $\widehat{p}_i^{[j]}(t)$ be a certain nonparametric estimate of $p_i(t)$ based on a single sample $\mathbf{X}_j$ from the class ω_i. We only consider the estimates that are positive, i.e., $\widehat{p}_i^{[j]}(t) \geq 0$. Then, the aggregated estimate of $p_i(t)$ takes the following form

$$\widehat{p}_i(t) = \frac{1}{L_i}\sum_{j=1}^{L}\widehat{p}_i^{[j]}(t)\mathbf{1}(Y_j = \omega_i), \;\; i = 1, 2\,. \tag{2.18}$$

Plugging (2.17) and (2.18) into (2.11) gives us the following empirical classification rule $\widehat{\psi}_{L,T}$: classify $\mathbf{X} = [t_1, \ldots, t_N; N] \in \omega_1$ if

$$\widehat{W}_{L,T}(\mathbf{X}) \geq \widehat{\eta}_{L,T}\,, \tag{2.19}$$

where $\widehat{W}_{L,T}(\mathbf{X}) = \sum_{i=1}^{N}\log\left(\frac{\widehat{p}_1(t_i)}{\widehat{p}_2(t_i)}\right)$, $\widehat{\eta}_{L,T} = \widehat{\tau}_1 - \widehat{\tau}_2 + N\log\left(\frac{\widehat{\tau}_2}{\widehat{\tau}_1}\right) + \log\left(\frac{L_2}{L_1}\right)$. In Section 2.4 we propose a concrete kernel-type estimate of the shape densities or equivalently the class intensity functions.

In this section we refer to the general result on the convergence of the rule $\widehat{\psi}_{L,T}$ to the Bayes decision ψ_T^*. This result is in the spirit of the Bayes risk consistency theorem established in [27] in the context of the standard fixed dimension data sets. Let $\mathbf{P}(\widehat{\psi}_{L,T}(\mathbf{X}) \neq Y|\mathbf{D}_L)$ be the conditional risk associated with the rule $\widehat{\psi}_{L,T}$. By (P) we denote the weak convergence (in probability).

Theorem 2. Assume that for $i = 1, 2$

$$\text{A3}: \; \sup_{t\in[0,T]}|\widehat{p}_i(t) - p_i(t)| \to 0 \;\; (P) \tag{2.20}$$

as $L \to \infty$. Then,

$$\mathbf{R}_{L,T} \to \mathbf{R}_T^* \;\; (P)$$

as $L \to \infty$.
The proof of Theorem 2 is given in [49].

The result of Theorem 2 assures the convergence of the plug-in classification rule to the optimal Bayes decision if the class shape densities (or equivalently the class intensity functions) have uniformly consistent estimates. This fact should be verified for the particular type of shape densities estimates. We will do so in the next section in the context of the kernel estimates.

2.4 Kernel classifier

It is known [52, 50] that the intensity function of a point process can be efficiently estimated by a class of kernel methods [53, 54]. In particular, the standard convolution kernel estimate of $\lambda_i(t)$ from a single realization $\mathbf{X}_j = \left[t_1^{[j]}, \ldots, t_{N^{[j]}}^{[j]}; N^{[j]}\right]$ takes the form

$$\widehat{\lambda}_i^{[j]}(t) = \sum_{l=1}^{N^{[j]}} K_h\left(t - t_l^{[j]}\right)\mathbf{1}\left(Y_j = \omega_i\right), \;\; i = 1, 2\,. \tag{2.21}$$

Here $K_h(t) = h^{-1}K(t/h)$, where the kernel $K(t)$ is assumed to be a symmetric probability density function. Examples of proper kernel functions include the so-called Epanechnikov kernel

$$K(t) = \frac{3}{4}\left(1 - t^2\right)\mathbf{1}\left(|t| \leq 1\right) \tag{2.22}$$

and the Gaussian kernel

$$K(t) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}t^2\right). \tag{2.23}$$

The crucial tuning parameter h is called the bandwidth as it controls the level of smoothing via the scaled kernel $K_h(t)$.

The parameter τ_i can be estimated (from a single sample) by $\widehat{\tau}_i^{[j]} = N^{[j]}$. Therefore (2.21) yields the following estimate of the shape density

$$\widehat{p}_i^{[j]}(t) = \frac{1}{N^{[j]}}\sum_{l=1}^{N^{[j]}} K_h\left(t - t_l^{[j]}\right)\mathbf{1}\left(Y_j = \omega_i\right). \tag{2.24}$$

As we have already pointed in Section 2.3, the estimates $\widehat{\lambda}_i^{[j]}(t)$, $\widehat{p}_i^{[j]}(t)$ cannot be consistent by merely increasing T. To overcome this problem one can utilize the observed multiple training vectors and aggregate the single-sample estimates $\left\{\widehat{\lambda}_i^{[j]}(t)\right\}$, $\left\{\widehat{p}_i^{[j]}(t)\right\}$. This leads to the following aggregated kernel estimate of $p_i(t)$

$$\widehat{p}_i(t) = \frac{1}{L_i}\sum_{j=1}^{L}\widehat{p}_i^{[j]}(t)\mathbf{1}\left(Y_j = \omega_i\right). \tag{2.25}$$

Moreover, the aggregated estimate $\widehat{\tau}_i$ of τ_i is defined in (2.17). Plugging $\widehat{p}_i(t)$ and $\widehat{\tau}_i$, $i = 1, 2$ into (2.19) we obtain the kernel classification rule. The following result gives the sufficient conditions for the Bayes risk consistency property established in Theorem 2 in the context of the kernel classification rule.

Theorem 3. Let the class intensity functions be Lipschitz continuous on $(0, \infty)$. Assume that the kernel function is also Lipschitz continuous on $[-1, 1]$. Suppose that the bandwidth h depends on L in such a way that

$$h(L) \to 0 \text{ and } Lh^3(L) \to \infty.$$

Then, we have

$$\widehat{\mathbf{R}}_{L,T} \to \mathbf{R}_T^* \ (P),$$

as $L \to \infty$.

The proof of Theorem 3 is given in [49].

It must be noted that the conditions imposed on the class intensity functions and the kernel function assure the uniform convergence of the kernel estimate required in Theorem 2. However, the convergence only holds at the interior points $[\delta, T-\delta]$ for some $\delta > 0$. This is due to the inherent boundary problem of the standard kernel estimate. In our case the boundary consists of two points, i.e., $t = 0$ and $t = T$. Further discussion on the effect of the boundary on the kernel classifier performance and the impact of boundary-correcting methods is deferred to Section 2.4.2.

The selection of the bandwidth h is a critical issue in determining the accuracy of the kernel classification rule. In practical applications one can specify the bandwidth using some resampling techniques like cross-validation [50, 53]. In our experimental studies we choose separate bandwidth for each class. This is done by finding the maximum of the log-likelihood of the kernel estimate of the class shape densities. Hence, let $\widehat{p}_i(t;h)$ be the kernel estimate in (2.25) specified by the bandwidth h. Then, for the given validation set of size q (per class) we chose the bandwidth as follows

$$\widehat{h}_i = \arg\max_h \sum_{l=1}^{p} \sum_{r=1}^{N^{[l]}} \log\left(\widehat{p}_i(t_r^{[l]};h)\right), \quad i = 1,2, \tag{2.26}$$

where $t_r^{[l]}$ is r-th observation of the l-th validation sample. Note that here $\widehat{p}_i(t;h)$ is the kernel estimate determined from the training set of size $L_i - q$.

2.4.1 Kernel classifier convergence to the Bayes classification rule

To show that the plug-in kernel classification rule risk converges to the Bayes risk (2.12) as the observation window length T and the training set size L increase, we conducted to a simulated data study. In all experiments the kernel classifier is given by (2.19) with the estimated $\widehat{\tau}_i$, $\widehat{p}_i(t;h)$ specified by (2.17) and (2.25), respectively. The Gaussian kernel (2.23) is used, whereas the bandwidth is selected according to the log-likelihood heuristic (2.26). When selecting the bandwidth, we consider a grid of ten evenly logarithmically spaced points $h \in \left[10^{-1}, 10^{1}\right]$. Additionally, we employ a 5-fold cross validation in order to avoid biasing the optimal bandwidth $\widehat{h}_i$ with test data.

Similarly to Section 2.2.1, we use Monte Carlo simulations to illustrate the convergence $\widehat{\mathbf{R}}_{L,T} \to \mathbf{R}_T^*$ as $L \to \infty$ and also the limit behavior of $\widehat{\mathbf{R}}_{L,T}$ as a function of T for a fixed value of L. To do so we need to slightly modify the previously established experimental protocols to account for the fact that now there is separate training data used to estimate $\widehat{\tau}_i$, $\widehat{p}_i(t;h)$, and test data to evaluate the classification rule (2.19) on.

Simulation 3. Denote $\mathbf{D}_L = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_L, Y_L)\}$ as a sample of L independent observations of the spiking processes assigned to the training data. Additionally, let

$\mathbf{D}_P = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_P, Y_P)\}$ be a sample of P independent observations of the spiking processes assigned to the test data. Without the loss of generality, we assume $\pi_1 = \pi_2 = 0.5$, which implies that the factors $\frac{L}{2}$ and $\frac{P}{2}$ represent the number of examples sampled per class for the training and test data, respectively. Then, the simulation procedure can be summarized as follows:

1) Specify the number of simulation repetitions ξ; intensity function pair $\lambda_1(t)$, $\lambda_2(t)$; sequence observation interval $[0, T]$; and the total number of sampled spike trains for the training (L) and test (P) data.
2) Construct a simulation grid $\{L_1, L_2, \ldots, L_K\} \in (0, L]$ specifying $k \in \{1, \ldots, K\}$ points at which to evaluate the classification rule.
3) For every repetition of the experiment:
   a) Simulate $\frac{L}{2}$ and $\frac{P}{2}$ spike trains according to the intensity function $\lambda_1(t)$, and additionally $\frac{L}{2}$ and $\frac{P}{2}$ examples according to $\lambda_2(t)$.
   b) At each point L_k of the grid:
      i. Select a subset of spike trains that is valid for the current simulation evaluation point, i.e., $\mathbf{D}_{L_k} = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_{L_k}, Y_{L_k})\} \subset \mathbf{D}_L$.
      ii. Estimate $\widehat{\tau}_i$ (2.17) and $\widehat{p}_i(t; h)$ (2.25) on $\mathbf{D}_{L_k}$.
      iii. Compute the classifier prediction according to the rule (2.19) for every pair $(\mathbf{X}_j, Y_j) \in \mathbf{D}_P$.
      iv. Compute empirical risk (2.12) based on classifier decisions for the P spike trains.
4) Compute the mean and standard deviation of the empirical risk over ξ repetitions.

Then the mean empirical risk over ξ repetitions $\mathbb{E}\left[\mathbf{R}_{L,T}\right]$ is the Monte Carlo estimate of $\widehat{\mathbf{R}}_{L,T}$ for fixed T. Similarly to Simulation 1, for every repetition of the experiment, this simulation protocol reuses some sequences whenever the effective sample size L_k increases. Conversely, the test data $\mathbf{D}_P$ remains fixed regardless of the training data size.

Let us first focus on the simulation results obtained for the intensity function specified by (2.15). Figure 2.6a shows the convergence of the empirical kernel rule risk to the Bayes risk (established in Section 2.2.1) for different values of the intensity function pair parameters ϕ_1, ϕ_2 versus L and for a fixed $T = 10$. We set $P = 10^4$ based on the result established in the aforementioned Section. Clearly, as the difficulty of the problem increases (i.e., the point processes sampled according to the two intensity functions become more similar to one another), the rate of convergence decreases.

Next, we wish to investigate the behavior of $\widehat{\mathbf{R}}_{L,T}$ as a function of T for a fixed value of L. Doing so requires modifying Simulation 2 to use the kernel classifier fitted on training data.

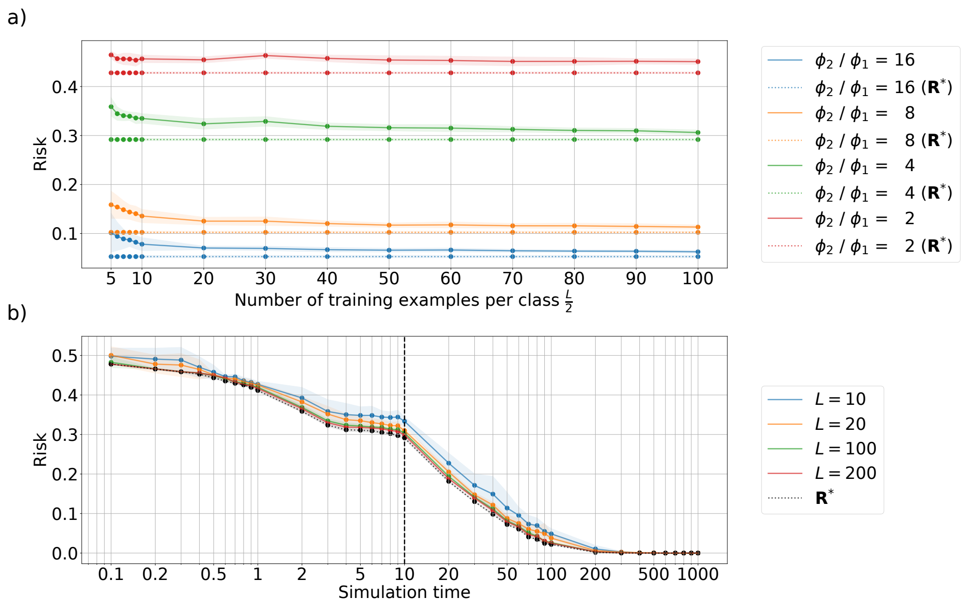


Figure 2.6: The results of Monte Carlo simulations for the two-class classification problem for the intensity function (2.15) parameterized by ϕ. a) The average risk $\mathbb{E}\left[\mathbf{R}_{L,T}\right]$ versus L for $T = 10$ for different values of intensity function pairs parameterized by ϕ_1, ϕ_2. The horizontal dashed lines denote Bayes risk $\mathbf{R}^*_T$ for the associated intensity function pair. b) The average risk $\mathbb{E}\left[\mathbf{R}_{L,T}\right]$ versus T across different values of L for $\phi_2/\phi_1 = 4$. Dashed vertical line at $T = 10$ denotes the simulation space slice presented in the top row. The curve for Bayes risk is added for reference.

Simulation 4. The simulation procedure can be summarized as follows:

1) Specify the number of simulation repetitions ξ; intensity function pair $\lambda_1(t)$, $\lambda_2(t)$; sequence observation interval $[0, T]$; and the total number of sampled spike trains for the training (L) and test (P) data.
2) Construct a simulation grid $\{T_1, T_2, \ldots, T_K\} \in [0, T]$ specifying $k \in \{1, \ldots, K\}$ points at which to evaluate the classification rule.
3) For every repetition of the experiment:
   a) Simulate $\frac{L}{2}$ and $\frac{P}{2}$ spike trains according to the intensity function $\lambda_1(t)$, and additionally $\frac{L}{2}$ and $\frac{P}{2}$ examples according to $\lambda_2(t)$.
   b) At each point T_k of the grid:
      i. Select a subset of events that is valid for the current simulation evaluation point, i.e., find $\mathbf{X}_{j,T_k} = \left[t_1^{[j,T_k]}, \ldots, t_{N^{[j,T_k]}}^{[j,T_k]}; N^{[j,T_k]}\right]$ such that $t_{N^{[j,T_k]}}^{[j,T_k]} \leq T_k$, for all $\mathbf{X}_j \in \mathbf{D}_L \cup \mathbf{D}_P$.
      ii. Estimate $\widehat{\tau}_i$ (2.17) and $\widehat{p}_i(t; h)$ (2.25) on $(\mathbf{X}_{j,T_k}, Y_{j,T_k}) \in \mathbf{D}_L$.
      iii. Compute the classifier prediction according to the rule (2.19) for every pair $(\mathbf{X}_{j,T_k}, Y_{j,T_k}) \in \mathbf{D}_P$.
      iv. Compute empirical risk (2.12) based on the classifier decisions for the P spike trains.

4) Compute the mean and standard deviation of the empirical risk over ξ repetitions. Then the mean empirical risk over ξ repetitions $\mathbb{E}\left[\mathbf{R}_{T,P}\right]$ is the Monte Carlo estimate of $\widehat{\mathbf{R}}_{L,T}$.

In contrast to the analysis presented in Section 2.2.1, when evaluating Simulation 4 we choose to vary L while keeping the intensity function pair parameters constant. This is to highlight that the rate of convergence of $\widehat{\mathbf{R}}_{L,T}$ to the Bayes risk depends on both L and T. We choose the parameters such that the classification problem is difficult enough to observe the behavior of $\widehat{\mathbf{R}}_{L,T}$ across both L and T, but not so difficult that it does not reach zero risk for some T.

Figure 2.6b depicts $\mathbb{E}\left[\mathbf{R}_{L,T}\right]$ for the intensity function pair (2.15) for $\phi_2/\phi_1 = 4$ versus T with the size of training data ranging from $L = 10$ to $L = 200$. The Bayes risk $\mathbf{R}^*_T$ is also plotted for comparison. The observed convergence $\mathbb{E}\left[\mathbf{R}_{L,T}\right] \to 0$ is analogous to the findings for the Bayes risk (Figure 2.3b). Also, the small value of the difference $\mathbb{E}\left[\mathbf{R}_{L,T}\right] - \mathbf{R}^*_T$ for all T should be noted. Additionally, we observe a small variability of the empirical risk with respect to the training data size L.

Next, we analyze the value of the optimal bandwidth selected according to the log-likelihood method versus T. For brevity, in Figure 2.7a we show only the results for $\widehat{h}_1$, noting that the curves obtained for $\widehat{h}_2$ are analogous. We observe an increase in $\widehat{h}_i$ with T, which aligns with the notion that as the observation window increases, the distribution of events in time becomes sparser, necessitating wider kernels. On the other hand, the obtained results also show that $h(L) \to 0$ as L increases. Another way to view this property is to analyze the model log-likelihood versus h for fixed T (Figure 2.7b).

To show that these findings hold for different classes of intensity functions, we run Simulations 3-4 on intensity functions introduced in Table 2.1. We choose to repeat their description in Table 2.2, given that the analysis is conducted only on a single parameterized intensity function pair. The results for these functions are summarized in Figure 2.8. In all cases the average empirical risk $\mathbb{E}\left[\mathbf{R}_{L,T}\right] \to 0$ as T increases. Additionally, it can be observed that $\mathbb{E}\left[\mathbf{R}_{L,T}\right] \to \mathbf{R}^*_T$ as L increases. Note that in Figure 2.8e the increase in risk for $T \in [10^2, 10^3]$ is an artifact of the simulation that arises due to a finite resolution of shape density estimation (2.25) for an intensity function with numerous discontinuities (Table 2.2e).

Finally, let us show a counter-example when the proposed algorithm fails to converge. Again, consider the intensity function 2.16 which does not satisfy the assumptions A1 and A2. Figure 2.9 depicts $\lambda_1(t; c, d) = \lambda\,(t; 300, 20)$ and $\lambda_2(t; c, d) = \lambda\,(t; 600, 40)$. Note that the empirical risk does not converge to the Bayes risk despite the latter being close to zero for some T. In fact, the average empirical risk $\mathbb{E}\left[\mathbf{R}_{L,T}\right]$ is the smallest around the maximum of (2.16) at $t = 0.5$. Afterwards $\mathbb{E}\left[\mathbf{R}_{L,T}\right]$ slightly increases and reaches a plateau because no new events can be observed.

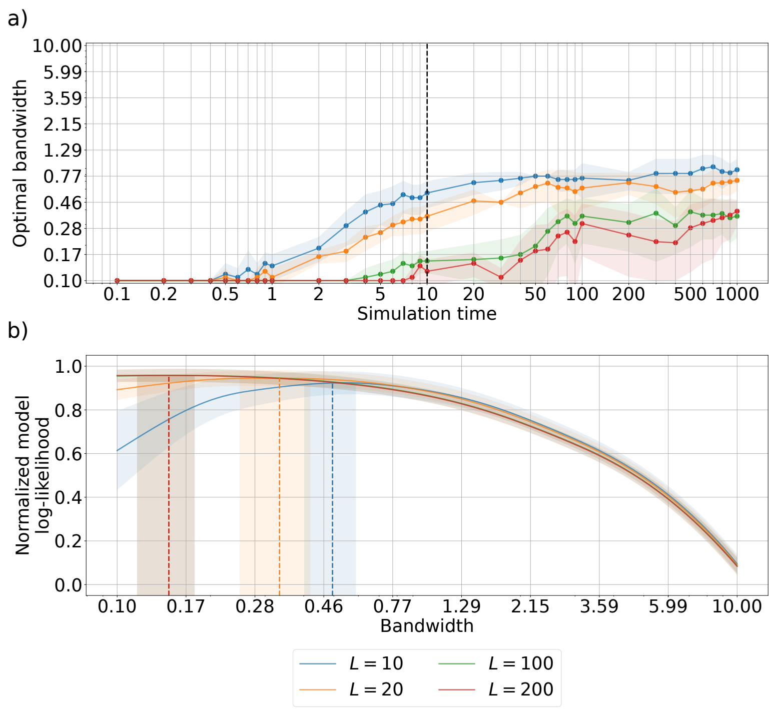


Figure 2.7: The results of Monte Carlo simulations for the two-class classification problem for the intensity function pair (2.15) specified by $\phi_2/\phi_1 = 4$. a) The average optimal bandwidth $\mathbb{E}\left[\widehat{h}_1\right]$ versus T for different values of L, estimated using 5-fold cross validation. The vertical dashed line at $T = 10$ denotes the simulation space slice presented in the bottom row. b) The average normalized model log-likelihood on test data versus h for different values of L. The vertical dashed lines denote function maxima. Note that the curves for $L = 100$ and $L = 200$ overlap.

Table 2.2: Additional set of intensity functions used in the simulated data study for the kernel classification rules risk convergence to the Bayes risk.

a)	$\lambda(t;r) = r$	$r_2/r_1 = 2$
b)	$\lambda(t;r) = (r+1) + r\cos\left(\frac{2\pi}{30}t + 2.6\right) + \cos\left(\frac{2\pi}{28}t + 4.5\right)$	$r_2/r_1 = 2$
c)	$\lambda(t;\phi) = \left[3.1 + 3\cos\left(\frac{\pi}{3\sqrt{2}}t + \phi\right)\right]^{1/2}$	$\phi_2/\phi_1 = 4$
d)	$\lambda(t;\phi) = 1.3\exp\left[\cos\left(\frac{\pi}{3\sqrt{2}}t + \phi\right)\right]$	$\phi_2/\phi_1 = 4$
e)	$\lambda(t;\phi) = 0.1 + 0.5 \mod [t+\phi, 2\pi]$	$\phi_2/\phi_1 = 4$

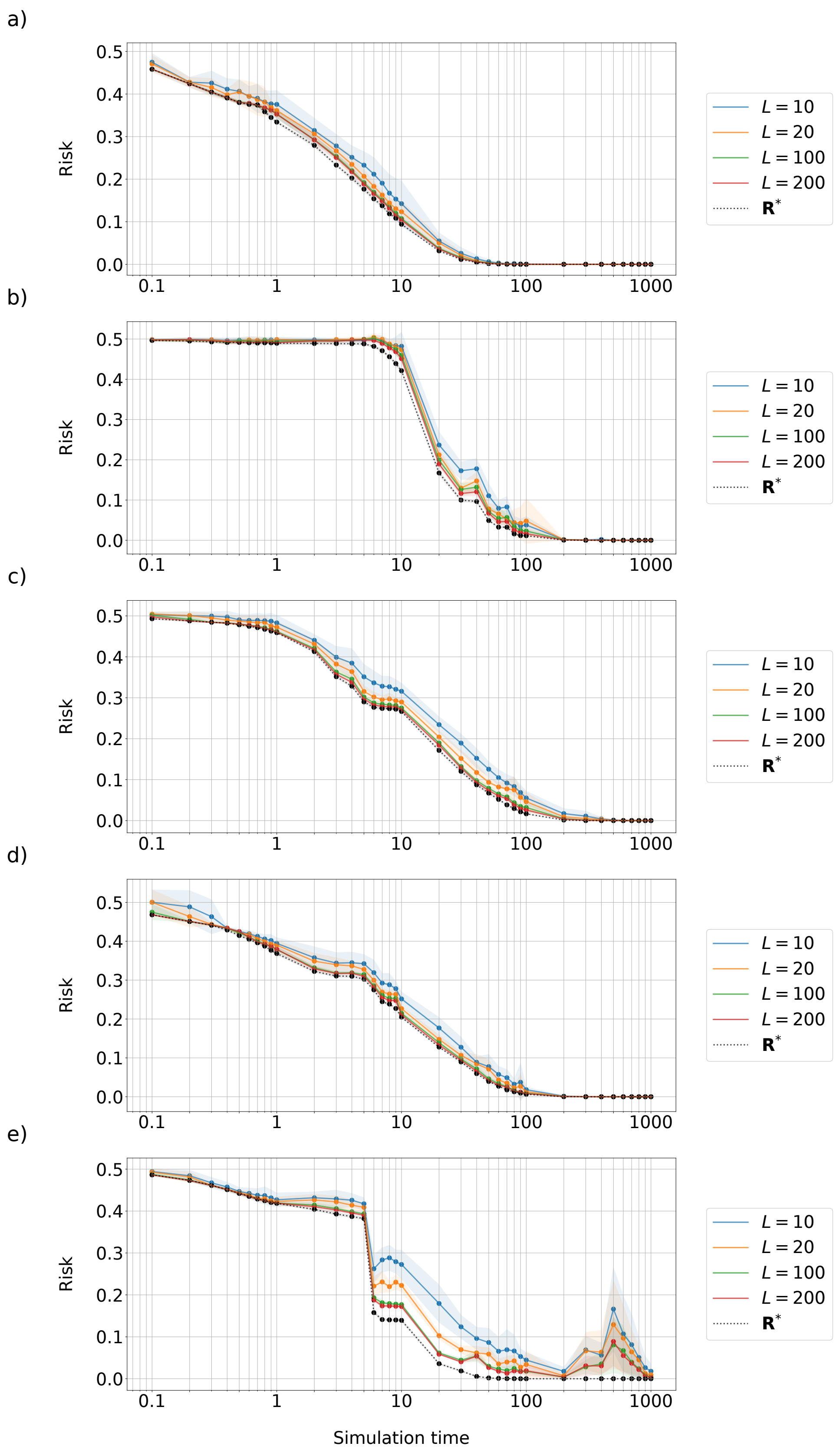


Figure 2.8: The average risk $\mathbb{E}\left[\mathbf{R}_{L,T}\right]$ versus T for for different values of L. Each row corresponds to a different two-class classification problem parameterized by an intensity functions presented in Table 2.2.

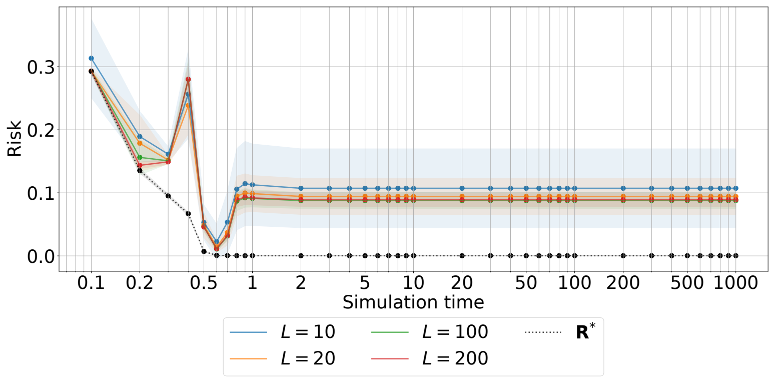


Figure 2.9: The average risk $\mathbb{E}\left[\mathbf{R}_{L,T}\right]$ versus T for different values of L for an exponential intensity function (2.16). Note that for a given set of parameters c, d the Bayes risk is zero for $T > 1$, whereas $\mathbb{E}\left[\mathbf{R}_{L,T}\right]$ is not.

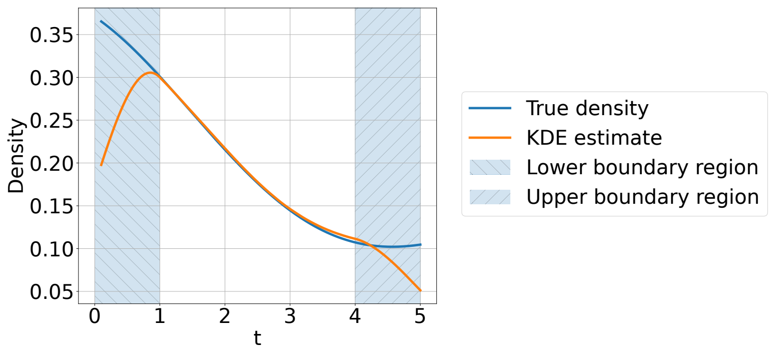


Figure 2.10: Kernel density estimation results with bandwidth $h = 1$ for a random variable with bounded support.

2.4.2 Impact of boundary correction on algorithm performance

For a compactly supported probability density function $g(t)$ kernel density estimation (KDE) – which forms the basis of shape density estimation (2.25) part of the algorithm – is known to produce estimates $\widehat{g}(t)$ that are significantly more biased near the boundaries than in the interior [55, 56]. Therefore, this same effect must occur when estimating the point process intensity function using the proposed kernel method. The lower boundary is, of course, equal to zero as time must be nonnegative, whereas the upper boundary T results from the sequence cut-off time after which we assume no events occur. These concepts are illustrated on Figure 2.10. Importantly, the purpose of this study is to evaluate the impact of boundary correction methods on the properties of our plug-in classification rules, and not propose a new method of boundary correction.

2.4.2.1 Applying BC-KDE to point process intensity estimation

Let us first define the analyzed boundary correction methods in the general kernel density estimation context, noting that adapting them to point process shape density estimation (2.25) is straightforward. The kernel density estimator of the density function $g(t)$ defined on $[0, T]$ is given by the following formula

$$\widehat{g}(t) = \frac{1}{L}\sum_{j=1}^{L} K_h\left(t - t_j\right) , \tag{2.27}$$

where $K_h(u) = h^{-1}K(u/h)$ is the scaled kernel function with the smoothing parameter h (bandwidth). Denote

$$\begin{aligned} a_k(c) &= \int_{-\infty}^{c} u^k K(u) du \\ b(c) &= \int_{-\infty}^{c} \left[K(u)\right]^2 du \end{aligned} , \tag{2.28}$$

and assume that $K(u)$ is a twice-differentiable symmetric and unimodal function such that the following conditions are fulfilled:

$$\begin{cases} a_0(\infty) = 1 \\ a_1(\infty) = 0 \\ a_2(\infty) > 0 \end{cases} . \tag{2.29}$$

An example of such kernel function are the Epanechnikov (2.22) and Gaussian (2.23) kernels.

For $g(t)$ bounded from below at $t_{\mathrm{LB}} = 0$, let $p_{\mathrm{LB}}(t) = t/h$ be the relative position indicator of t with respect to the lower boundary $t_{\mathrm{LB}} = 0$. Denote $p = p_{\mathrm{LB}}(t)$ and $h = h(L)$. If $h \longrightarrow 0$ and $Lh \longrightarrow \infty$ as $L \longrightarrow \infty$, then the KDE bias and variance are given as [57]

$$\begin{aligned} \mathbb{E}\left[\widehat{g}(t)\right] &\approx a_0(p)g(t) - ha_1(p)g'(t) + \frac{1}{2}h^2 a_2(p)g''(t) \\ \mathrm{Var}\left[\widehat{g}(t)\right] &\approx (Lh)^{-1} b(p)g(t) \end{aligned} . \tag{2.30}$$

These expressions describe the bias and variance both in the boundary and interior regions. Note that for a compactly supported kernel function (such as the Epanechnikov kernel), the middle term of the expectation $ha_1(p)g'(t)$ equals zero in the interior region. For a density function $g(t)$ with an upper boundary $t_{\mathrm{UB}} = T$ the relative position indicator becomes $p_{\mathrm{UB}}(t) = (T - t)/h$. If $g(t)$ has only the upper boundary, $p_{\mathrm{UB}}(t)$ can be plugged into (2.30) in place of p. This implies that the set of equations (2.30) is also valid in case $g(t)$ is bounded from above and below, but these boundaries do not overlap (i.e., there exists an interior). When the interior region is empty, these expressions for bias and variance need to be modified [58].

For our analysis, it is sufficient to set $t_{\mathrm{LB}} = 0$ and $t_{\mathrm{UB}} = T$. Then, we consider the following boundary correction methods:

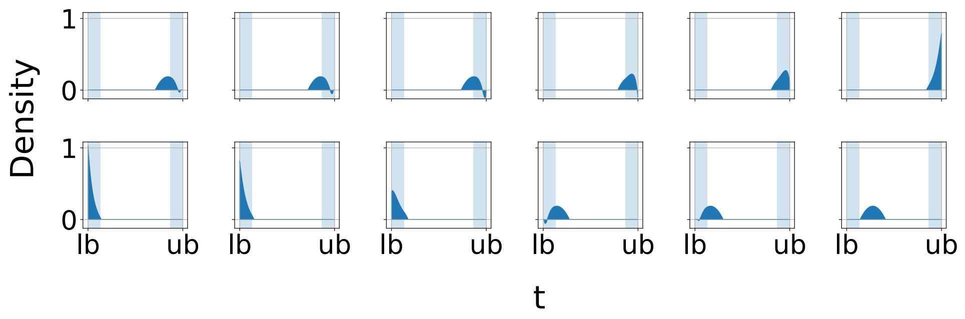


Figure 2.11: Examples of jackknived Epanechnikov kernels with bandwidth $h = 4$ constructed near the upper (top row) and lower (bottom row) boundary for data supported on $t \in [0, 30]$.

1) <u>density renormalization</u> (also known as "cut-and-normalize" method) [55]

$$\widehat{g}_{\text{NRM}}(t) = \frac{\widehat{g}(t)}{a_0(p_{\text{LB}}) \cdot a_0(p_{\text{UB}})} . \tag{2.31}$$

2) <u>reflection</u> [59, 60]

$$\widehat{g}_{\text{REF}}(t) = \frac{1}{L} \sum_{j=1}^{L} \left[K_h\left(t + t_j\right) + K_h\left(t - t_j\right) + K_h\left(t - \left[2T - t_j\right]\right) \right] . \tag{2.32}$$

3) <u>generalized jackknifing</u> with a linear kernel combination [57, 58]

$$\begin{aligned}
s(u; p) &= \frac{a_2(p) - a_1(p)u}{a_0(p)a_2(p) - [a_1(p)]^2} \quad , \quad a_0(p)a_2(p) \neq [a_1(p)]^2 \\
s_{\text{LB}}\left(t; t_j, h\right) &= s\left(\frac{t - t_j}{h}; \frac{t}{h}\right) \\
s_{\text{UB}}\left(t; t_j, h, T\right) &= s\left(-\frac{t - t_j}{h}; \frac{T - t}{h}\right) \\
K_{\text{JCK}}\left(t; t_j, h, T\right) &= s_{\text{LB}}\left(t; t_j, h\right) \cdot s_{\text{UB}}\left(t; t_j, h, T\right) \cdot K_h\left(t - t_j\right) \\
\widehat{g}_{\text{JCK}}(t) &= \frac{1}{L} \sum_{j=1}^{L} K_{\text{JCK}}\left(t; t_j, h, T\right)
\end{aligned} . \tag{2.33}$$

The shape of the jackknived kernel K_{JCK} depends on its position relative to boundaries. An example of a family of jackknived Epanechnikov kernels is presented in Figure 2.11. Among the enumerated methods, the *generalized jackknifing* correction provides the best theoretical improvement of the bias in the boundary region [57]. The other two methods should in principle provide a similar, albeit smaller, improvement. From a computational complexity perspective, the best method seems to be the *density renormalization* because it is applied only after the density has been estimated using (2.27). On the other hand, both *reflection* and *generalized jackknifing* modify the density estimation function at every KDE evaluation point, making these methods scale adversely with the number of events in the sequence and the time axis resolution. Additionally, should the upper boundary increase for any reason, both *reflection* and *generalized jackknifing* need to be recomputed in the old boundary region. Nevertheless, we still consider these algorithms in our analysis (despite the drawbacks in terms of computational complexity) as using them might bring some benefits over the *density renormalization* method.

To assess the behavior of these boundary correction methods for the point process intensity function estimation, a preliminary experiment was conducted. In each case the point process intensity shape function was estimated from 1000 replicates. Figure 2.12 present some results of the study on simulated data sampled according to the intensity function (2.15) for progressively larger interior. In general, all boundary correction methods reduce the estimation bias in the boundary regions, with the *generalized jackknifing* method being noticeably more accurate than the other approaches. What is more interesting, however, is that when using automatic bandwidth selection according to the log-likelihood maximization rule (2.26), *generalized jackknifing* completely fails to find the correct value of the bandwidth when the interior region can potentially be empty (i.e., both boundary regions overlap). This occurs when choosing the largest candidate value for h on the optimization grid causes the boundary regions to encompass the entire signal. This finding was consistent across different simulated data scenarios.

Additionally, we found that the *density renormalization* method was robust to different analysis parameters and chosen intensity functions. We therefore experimented with using this algorithm in the bandwidth selection step of the algorithm (2.26), but applying the other methods during the final shape function estimation step. The result of this procedure is presented in the bottom row of Figure 2.12. We found that the values of bandwidth determined by this procedure lead to more accurate estimates of the intensity function, which could have a positive impact on the classifier performance. Henceforth we shall call this procedure *stable bandwidth selection*, or *stable* version of the algorithm.

2.4.2.2 Applying BC-KDE to spike train data classification

In order to assess the impact of boundary correction on classification using our proposed approach, we conducted experiments on spike trains generated according to intensity functions introduced in Section 2.2.1. We focused our efforts on analyzing the rate of convergence to the Bayes risk in terms of the number of training examples L for a fixed sequence length T. We showed in Section 2.4.1 that this classifier converges to the Bayes risk for a sufficiently large L. In this study T is set to be small so that the relative impact of boundary effects on the classification performance is large. This constraint means that incorrectly choosing the bandwidth h may cause the interior to be empty.

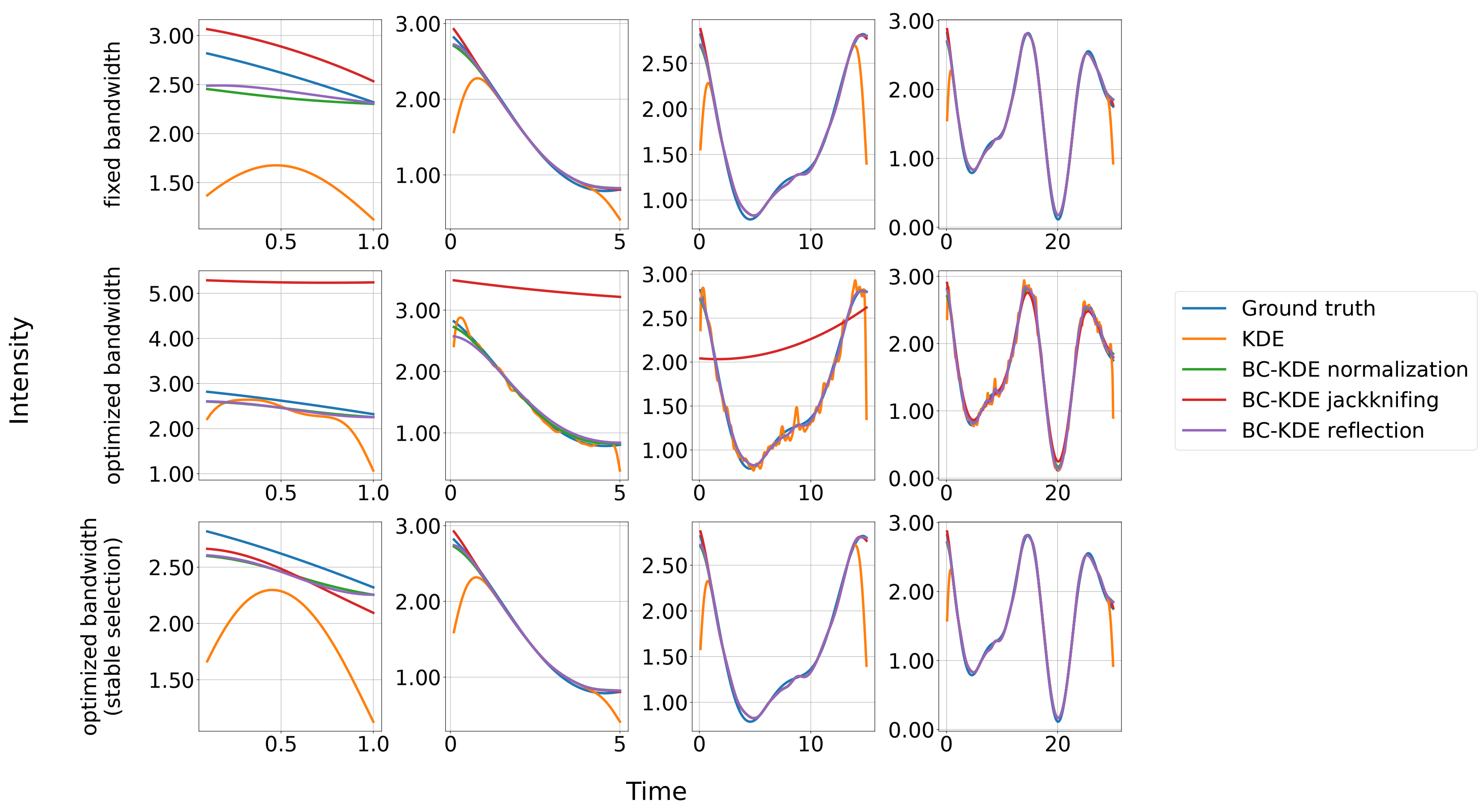


Figure 2.12: The effect of different boundary correction methods on point process intensity function estimation for data simulated according to (2.15) for progressively larger interior. Gaussian kernel was used as the base kernel. Top row: bandwidth parameter is set the same for all methods. Middle row: automatically-selected bandwidth according to (2.26). Bottom row: same as middle row, but in all cases BC-KDE *density renormalization* was used during bandwidth optimization.

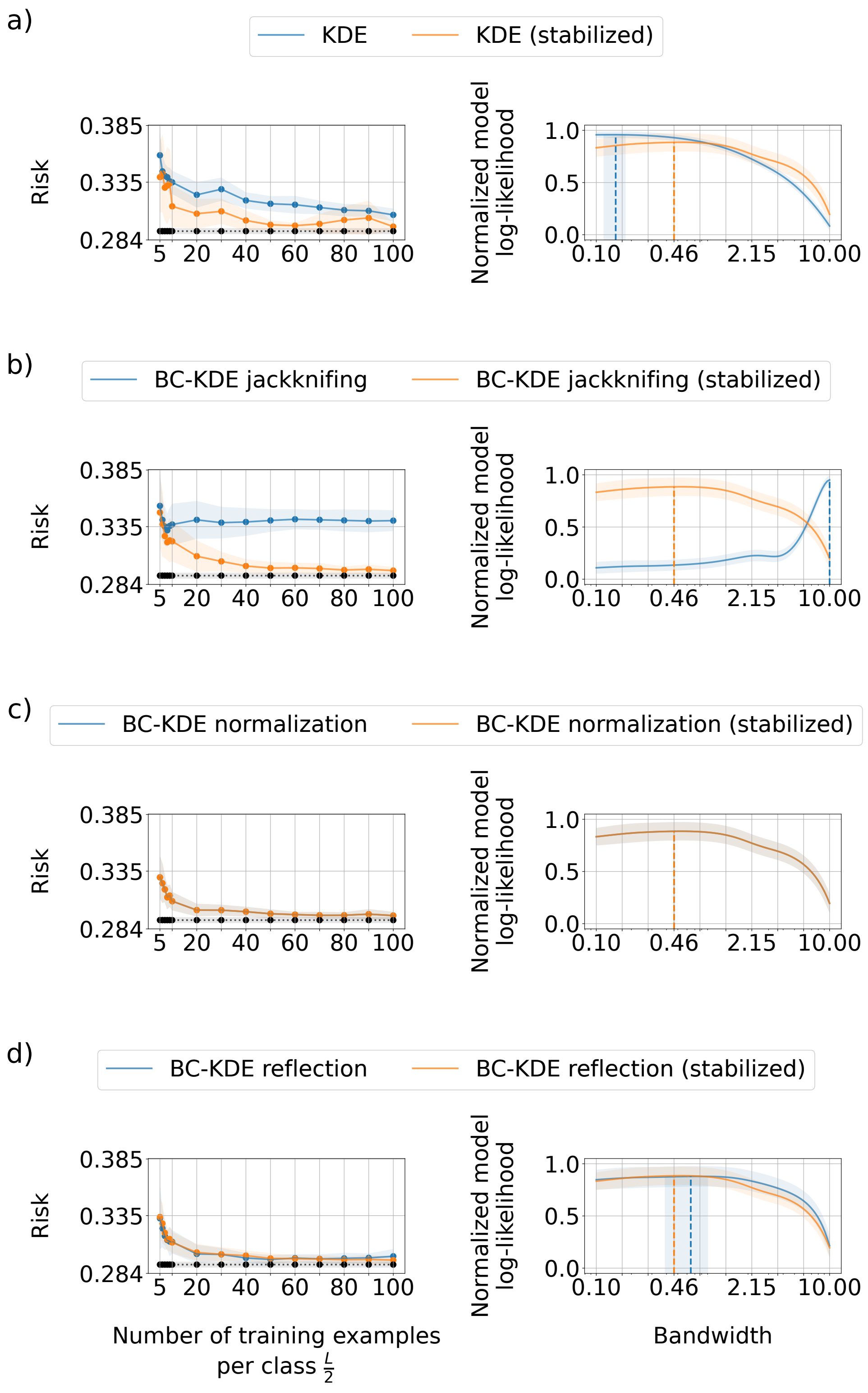


Figure 2.13: Results of the classifier convergence study with respect to the number of training examples L for data simulated according to (2.15), depending on the applied boundary correction method. Left column: the average risk $\mathbb{E}\left[\mathbf{R}_{L,T}\right]$ versus L compared to the Bayes risk baseline for fixed $T = 10$. Right column: the bandwidth selection curves for class ω_1 with $T = 10$ and $L = 200$. The vertical dashed lines denote function maxima. Note that the *density normalization* method and all *stable* version of the algorithm selected the same bandwidth.

An example of the results obtained for spike train simulated according to (2.15) is shown in Figure 2.13. In this case it can be observed that all methods other than *jackknifing* improve the convergence rate with respect to L compared to the KDE baseline. As shown in the previous Section, the most likely reason for *jackknifing* method failing to converge is that the automatic bandwidth selection settles, incorrectly, on the value of h that is the largest for a given optimization grid, significantly overestimating $\widehat{h}$. Note that for $L = 200$ each model selected a different bandwidth.

In order to reason about the overall impact of the boundary correction methods, we need to aggregate the results obtained for all intensity function pairs and algorithms. Let

$$\mathbf{R}_g^{[Q]} = \left|\widehat{\mathbf{R}}_{L,T}^{[Q]} - \mathbf{R}_T^*\right| \tag{2.34}$$

be the estimated risk gap $\mathbf{R}_g$ between the classification risk $\widehat{\mathbf{R}}_{L,T}$ for a model Q fitted on L training examples with the upper boundary set to T, and the Bayes risk $\mathbf{R}_T^*$. We define the relative improvement $\mathrm{RI}^{[Q,B]}$ of the model Q compared to the baseline model B as

$$\mathrm{RI}^{[Q,B]} = 1 - \frac{\mathbf{R}_g^{[Q]}}{\mathbf{R}_g^{[B]}} . \tag{2.35}$$

We compute this factor for all proposed boundary correction methods by taking the uncorrected KDE as the baseline B. Additionally, we also compute this factor for the *KDE-stable* variant where boundary correction by *density normalization* was used only to select the bandwidth (keeping the rest of the algorithm unchanged). Figure 2.14 summarizes the obtained results by algorithm name aggregated over all intensity function pairs used in the simulated data study (Table 2.2). In general the *reflection* and *density normalization* methods slightly improve classification performance. By contrast, *jackknifing*-based correction makes the risk gap significantly larger. Based on the raw results, such as those presented in Figure 2.13 and the earlier qualitative analysis, we reason that this method is incompatible with the chosen optimal bandwidth selection heuristic (2.26), causing the model to fail to fit the shape density function. We note that stabilizing the bandwidth selection algorithm significantly changes the results for *jackknifing* and *reflection* methods. The former no longer has a net negative impact on the classifier, while the latter achieves a median RI (or mRI) of 0.26. Additionally, the KDE-*stable* model performs worse than the baseline, which suggests that choosing a more accurate bandwidth does not always lead to a better kernel classifier.

2.5 Applications – Twitter bot detection

So far we have laid theoretical groundwork for the kernel-method-based approach for the two-class classification problem. The proposed methodology was empirically evaluated on

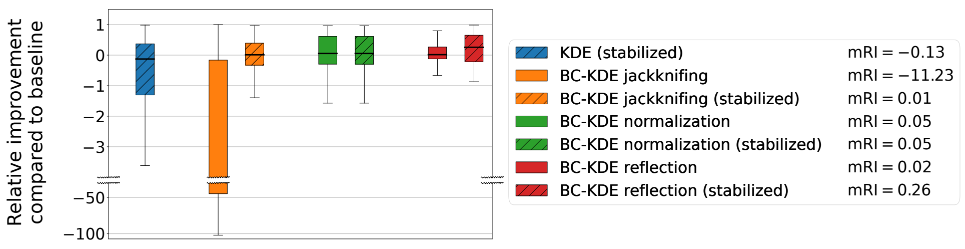


Figure 2.14: Box plots of the observed relative improvement of choosing a BC-KDE algorithm over the baseline KDE model. The results are grouped by algorithm type and aggregated over all intensity function pairs used in the simulated data study (Table 2.2). The label mRI denotes the median value of the RI factor (2.35). Note that RI is unbounded from below and can have a large magnitude when model performance is significantly worse than the baseline.

simulated data and analyzed in terms of the impact of the training set size, observation window length, kernel bandwidth selection, and applied boundary correction method. In this Section we extend our analysis to real-life data. Naturally, care must be taken to select a dataset that satisfies the assumptions and constraints imposed by our approach. To construct the kernel estimate of shape density function we require that events observed within each spike train sequence are independent of one another. Additionally, the number of events in each sequence must increase with the observation window T. Lastly, we require that time is the primary carrier of information of the signal.

Thus, we choose to evaluate our approach by fitting our classifier on a real-life dataset of legitimate and automated Twitter user activity [5]. We choose this data for several reasons. First of all, each record is described only in terms of the time of event without any auxiliary information (such as tweet content and sentiment). Secondly, spike trains formed for each record have a long sequence duration, allowing us to analyze the performance of the classifier versus T. Lastly, the events occur at timescales differing by many orders of magnitude. This allows us to assess the feasibility of the proposed approach in scenarios commonly encountered when processing this type of data.

2.5.1 An overview of Twitter bot detection

Social bots are automated agents that interact with humans and mimic human behavior [61]. In social media environment such bots may aggregate information from various sources, automatically respond to custom queries, or even generate content that satisfies some constraints. However, some bot behavior and intent is purposefully misleading or downright malicious. One example that is difficult to detect is political discourse polarization by spreading misinformation and promoting confirmation bias in like-minded people (i.e., creating an echo chamber) [62, 63]. Taking into account the increasingly more sophisticated programming behind the malicious social bots, the field of bot detection research lays the foundation to what can be described as an arms race against novel automated agents [64]. One particular impediment to this research is the lack of a standard benchmark that could be used to evaluate and compare different approaches [65].

Bot detection systems can be divided into three categories: crowdsourcing, feature-based classification, and social network analysis. Crowdsourcing techniques rely on the ability of hired human experts and volunteers to manually annotate bots based on their profiles, or on sending survey requests to the users and analyzing their replies [64]. Feature-based classification aims to automate the process by analyzing user-level features, usually by aggregating data pertaining to the user itself and to their activity [66, 67, 68]. Note that while this method can include features based on the social network structure formed by the user, it still analyzes only one user at a time [69]. Furthermore, user-focused methods are not limited to supervised learning from labels. It is also possible to perform unsupervised clustering of users based on the temporal similarity of their activity, regardless of their relationship in the social network [5]. This circumvents the time-consuming and costly process of labeling the data. Additionally, the unsupervised systems do not become outdated when new generations of bots, exhibiting patterns of behavior not present during model training, become more prevalent. However, they treat all unclustered examples as legitimate users, limiting their ability to detect bots exhibiting irregular behavior. The last of the three bot detection categories expands the scope of analysis to operate on a large group of users at once [70, 71, 72]. Similarly to the unsupervised user-level methods, they do not become obsolete due to data drift [71]. However, given that they analyze the actual community structure formed by users, they take more time to process information, making them more difficult to operate at scale [73].

In this work we focus on analyzing user data in terms of tweet and retweet actions as the only information available to the model. A tweet is the original message that can be re-broadcasted (i.e., retweeted) by other users. Analyzing retweet patterns can be utilized to predict which tweets will go viral or to identify the tweet target demographic [74]. Note that such analysis can also be used to determine the legitimacy of a retweet thread, or in other words: whether it occurred organically, or whether the activity was artificially inflated [75]. While it is more common to analyze retweets with group-based methods [76, 77, 78], automated behavior can also be made evident by analyzing user-level patterns of retweet activity [79, 69, 5]. The latter is the approach followed in our study.

Table 2.3: Attributes used to describe every record of the RT_{BUST} study Twitter dataset.

Field name	**Description**
user-id	Unique identifier associated with the retweeting user
user-screen_name	Name associated with *user-id*
user-created_at	Timestamp of the retweeting user creation
retweet-id	Unique identifier associated with the retweet
retweet-created_at	Timestamp of the retweet
originUser-id	Unique identifier associated with the tweeting user
originUser-screen_name	Name associated with *originUser-id*
originUser-created_at	Timestamp of the tweeting user creation
originTweet-id	Unique identifier associated with the tweet
originTweet-created_at	Timestamp of the tweet
is_bot	True/False label of whether the user is a bot, or NULL if the record is unlabeled

2.5.2 Dataset description

The RT_{BUST} study reported in [5] used Twitter Premium Search API to compile a list of all Italian retweets shared between 18 June 2018 and 1 July 2018. In this 2-week period there have been almost 10M retweets shared by 1.4M distinct users. The compiled dataset[a] consists of records of retweet timestamps associated with some original tweets. These records can be aggregated based on user id such that each user is characterized by a different number of tweet-retweet pairs. To supplement this vast collection of unlabeled records, authors of [5] manually annotated about 1000 accounts based on published annotation guidelines for datasets containing social bots [64].

The description of individual fields of every record of the raw dataset is provided in Table 2.3. Note that these records contain no information on the content of shared tweets (although they can be sourced based on the *originTweet-id* field). For this reason the goal is to classify legitimate and bot users based solely on retweet timestamps. Notably, every tweet-retweet pair is independent of any other pair. This is a reasonable assumption given the data; however, it seems plausible that graph community structure and programmed bot behavior play an important role in what gets retweeted, and when.

We limit the scope of our analysis to the labeled portion of this dataset and, in contrast to the original study, apply a supervised learning algorithm on individual data points. Let *retweet*

[a]The dataset is available at https://doi.org/10.5281/zenodo.2653137

delay be the difference between retweet and the origin-tweet timestamps, expressed in minutes.[b] This quantity can be treated as describing an observed event, hence we denote it as t in order to avoid introducing additional notation. We filter the records so that only those with both tweet and retweet timestamps occurring from 18 June 2018 to 1 July 2018 (exclusive) are kept, meaning that the retweet delay attribute is bounded from above to about $\tau_{\max} = 2 \cdot 10^4$ minutes (2 weeks). Additionally, only records with a delay of at least $\tau_{\min} = 10^{-1}$ minutes were considered for further analysis. Records with a retweet delay below the selected $\tau_{\min}$ can be considered a sign of either auto-retweeting bots, or legitimate users that were refreshing their feed as the tweet was posted and decided to retweet immediately. Either way, in our opinion, this does constitute a typical user behavior. As a result of this filtering step, 11.73% of all records were removed.

Finally, the records were grouped by the retweeting user and by label. This allowed us to construct a set of data points $\boldsymbol{X}$ with elements $\boldsymbol{x} = [t_1, \ldots, t_U; U]$ such that $\tau_{\min} < t_1 < \ldots < t_U < \tau_{\max}$ are the retweet delays bounded by observation time $\tau_{\max}$ and $U = U(\tau_{\max})$ is the length of the sequence. We obtain a nearly balanced data subset with 366 examples of legitimate users and 389 examples of bot users. These user tweet-retweet sequences have, on average, about 113 events with a maximum of 698. The remaining 46,883 cases are unlabeled and were not used in this study.

2.5.3 Data exploration & establishing a baseline classifier

Figure 2.15a presents a kernel-smoothed distribution of retweet delays of labeled data points. High degree of overlap between the two distributions implies that the classification problem is difficult to solve. Additionally, we can expect both small and large delays to play a role in differentiating the two classes. This is, however, a very simplified exploration of the problem, given that the dataset creators have identified several recurring patterns of bot behavior (see Figure 4 in [5]), which unfortunately they have not labeled in the dataset. From now on we will analyze the dataset as-is, aggregating all data points according to the label.

We create a simple classification baseline by treating as bots all accounts that have the number of retweets in this 2 week period higher than a fixed threshold established on the unlabeled portion of the dataset. This directly corresponds to the baseline of [5] which was based on user retweet rate (i.e., the number of retweets per unit of time). We evaluate this baseline in terms of average accuracy for different values of the (min-max normalized) retweet rate threshold. The obtained baseline is presented in Figure 2.15b. Choosing the third quantile of the unlabeled dataset distribution as the threshold, we obtain an accuracy score of 0.3417,

[b]Throughout the rest of this work, we express all units of time in terms of minutes, same as the original $\mathrm{RT}_{\mathrm{BUST}}$ study.

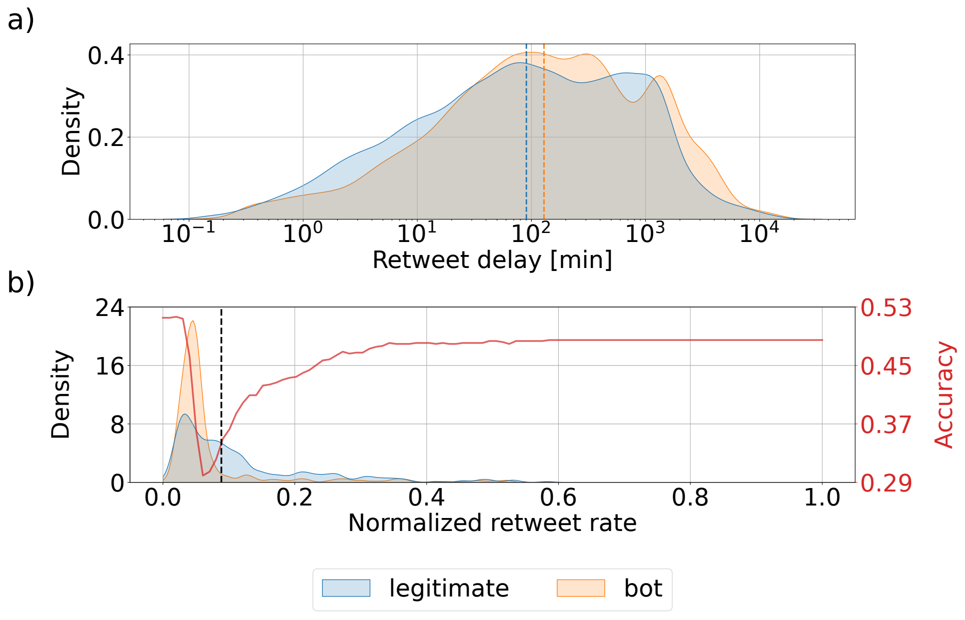


Figure 2.15: a) Retweet delay distribution for labeled data points. Vertical dashed lines denote median delay for each class. b) Normalized retweet rate distribution for the two classes and classifier performance for a retweet-rate-threshold-based rule. Vertical dashed line denotes the third quantile of retweet rate distribution of the unlabeled portion of the dataset. The obtained accuracy score for very low and very high threshold values is an indicator of a slight class label imbalance in this dataset (366 examples of legitimate users and 389 examples of bots).

which is similar to the value presented in the original study (0.3440). It is important to note that this score represents a worse-than-random binary classification performance and would be significantly higher if the threshold rule was flipped (i.e., classifying accounts with high retweet rate as legitimate instead of bots). Nevertheless, we keep this baseline for consistency with [5].

2.5.4 Applying the proposed method

According to the results of the simulated data study in Section 2.4, the classifier performance heavily depends on the size of the training set, as well as the observed sequence length. Therefore, it is reasonable to assume that the same holds for real-life data, although we need to take into consideration the cost of labeling data points, as well as the computational complexity of estimating the intensity function for all classes. Given the small sample size, we opt to use the stratified 5-fold cross-validation to evaluate the algorithm in lieu of a fixed training-test data split.

We have limited the scope of our analysis to the following four quantities:

1) the length of the retweet delay time series (T),

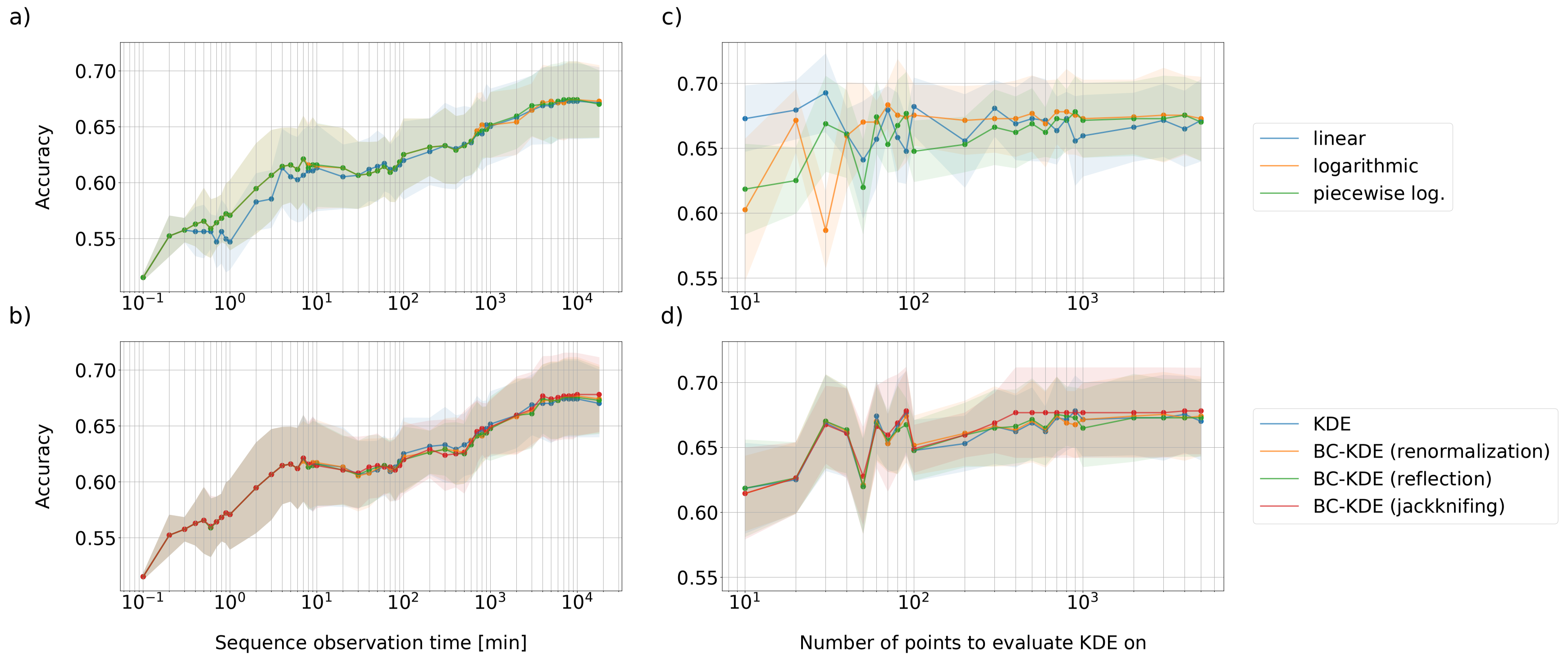


Figure 2.16: Performance of the proposed kernel classifier on the two-class Twitter bot detection problem. a) Accuracy versus retweet delay sequence observation time depending on the type of grid used to select KDE evaluation points. b) Accuracy versus retweet delay sequence observation time depending on the type of the KDE algorithm used to estimate the shape density function. c) Accuracy versus the number of points of the KDE grid depending on the type of grid used to select them. d) Accuracy versus the number of points of the KDE grid depending on the type of the KDE algorithm used to estimate the shape density function.

2) the number of points to evaluate the KDE on (k),
3) the type of the grid used to select KDE evaluation points:
    - linear,
    - logarithmic,
    - piecewise logarithmic (where the entire time-axis support is split into contiguous subregions and a separate logarithmic grid is defined for every subregion)[c],
4) the type of a KDE boundary correction method,

where the first two of the enumerated quantities directly impact the computational complexity of the proposed algorithm. This is an important aspect to consider for this specific dataset because a single data point (sequence) can be composed of events occurring at timescales differing by almost five orders of magnitude ($\tau_{\min} = 10^{-1} \leq t \leq \tau_{\max} = 2 \cdot 10^4$ [min]). All kernels are Gaussian (2.23).

Let us first summarize the results obtained in terms of the retweet delay sequence length while keeping the number of points to evaluate the KDE on fixed at $k = 5000$. Figure 2.16a shows how the choice of KDE evaluation point grid influences the classifier. The performance of the model steadily increases as more events are observed, although the choice of the grid seems to have an impact on the classifier only for extremely short sequences.

On the other hand, Figure 2.16b shows the results of the sequence length analysis in terms of the KDE boundary correction algorithm, for fixed $k = 5000$ and piecewise logarithmic grid. The boundary correction does not seem to have any noticeable impact on classifier performance, even at extremely short sequence lengths (which was noticeable in the simulated data study). Note the reduced variance at small sequence slice lengths.

The other set of experiments was conducted by varying the number of points to evaluate the KDE on while keeping $T = \max_{t \in X} t$ constant. Figure 2.16c summarizes the results for the type of grid analysis. Surprisingly, the linear grid seems to outperform the two logarithmic grid types at smaller number of grid points, and this property seems to be inverted for larger grid sizes. The performance for linear grid at small grid sizes could be explained by those points being selected for low density regions of small and large retweet delays (as per the exploratory analysis in Figure 2.15a) which is not the case for logarithmic grids. Finally, in Figure 2.16d we observe that there seems to be no significant difference in performance between BC-KDE algorithms when evaluated in terms of the KDE grid size.

We summarize our results in Table 2.4 and compare them to the baseline and different methods evaluated in the original study. We report the best result for our kernel-method-based algorithm among all analyzed parameter combinations, although, as evidenced in Figure 2.16,

[c]In the simulated data study the piecewise logarithmic grid has shown a good compromise between model performance and its computational complexity.

Table 2.4: Comparison of model performance on the bot detection task between different techniques.

Source	Technique	Accuracy	F_1-score
n/a	retweet rate baseline	34.40	35.59
[5]	Botometer	58.30	42.86
	HoloScope	49.08	0.96
	Social fingerprinting	71.14	75.82
[5]	RT_{BUST} (handcrafted features)	53.64	62.70
	RT_{BUST} (PCA)	51.54	66.49
	RT_{BUST} (TICA)	53.64	67.47
	RT_{BUST} (VAE)	87.55	86.87
our	KDE	67.81 ± 3.31	60.33 ± 4.08

what actually matters is the length of the sequence and the number of points to evaluate the model on (the more, the better). In terms of accuracy, our model has outperformed all techniques other than the Social fingerprinting [80] and the state-of-the-art results of RT_{BUST}-VAE. Unfortunately, it lags behind most other solutions in terms of F_1-score. Note that this measure focuses solely on the positive class (here: bot accounts), whereas accuracy takes into account both classes. For this reason F_1-score can be considered a more practical measure of performance on this classification problem, because it measures the relevance of the suspected bot account prediction. One possible way to improve the model performance with respect to F_1-score is to bias the classification rule (2.9) to make it more likely to predict a positive class label, unless the evidence suggests otherwise. Naturally, doing so is bound to decrease the accuracy of the model. Finally, among the presented results, RT_{BUST}-VAE emerges as the uncontested winner. However, we note that it is an unsupervised learning algorithm and is therefore able to infer different non-overlapping patterns of activity of distinct groups of users not present in the class-label-aggregated, supervised data on which we evaluate our model.

2.6 Summary

In this Chapter we described a novel methodology based on the Bayes decision theory for the two-class classification problem for a class of spike train data characterized by non-random intensity functions. The optimal Bayes rule was derived and its finite and asymptotic (with respect to the length of the observation interval) properties were established. Our theoretical findings were evaluated in a Monte Carlo simulation study. We found that the Bayes risk Monte Carlo estimate converges to zero as the length of the observation interval increases, with the rate of convergence depending on the difficulty of the classification problem and, under some conditions, on the asymptotic behavior of the class intensity functions. Notably, if the class intensity functions are compactly supported, then the convergence of the Bayes risk to zero is not guaranteed. This finding is supported by results of the simulation study for an intensity function pair counter-example.

Furthermore, we introduced a general class of plug-in empirical classification rules and formulated a set of sufficient conditions for their convergence (as the amount of data grows) to the Bayes risk. This optimality property was confirmed and verified for the plug-in kernel classifier derived from the aggregated data. The aggregated data was constructed by using replicates of the spiking process (i.e., different realizations of the same spiking process) to form the training and test sets for a two-class problem. Similarly to the analysis conducted for the optimal classification rule, we evaluated our theoretical findings in a finite sample study. The obtained results show that the empirical risk of the proposed kernel classifier converges to the Bayes risk as the training set size increases.

In addition to the kernel rules risk convergence study, the impact of boundary correction in the shape function estimation step of the proposed algorithm on the classifier performance was also assessed. We evaluated three different boundary correction methods and found that, in general, applying the *reflection* and *density renormalization* methods lead to a small improvement of the risk convergence rate. Conversely, the *generalized jackknifing* method tends to fail to automatically find the optimal value of the bandwidth when the interior region can potentially be empty (i.e., when both boundary regions overlap), which in turn causes the classifier predictions to be significantly less accurate than for the baseline model without any boundary correction. Interestingly, stabilizing the automatic bandwidth selection using the *density renormalization* method allowed us to observe a relative model improvement even when using the *generalized jackknifing* during the shape function estimation. However, it must be stressed that any noticeable impact (both positive and negative) of boundary correction on model performance is present only when the event stream observation period is sufficiently short. For longer sequences, where an interior region dominates, these effects are negligible.

Lastly, an exemplary model use-case scenario on real-life data was also presented. Our algorithm was applied to a labeled Twitter user activity dataset in order to determine whether each analyzed user is legitimate (i.e., human-controlled) or not. An interesting property of this specific data is that event times in each sequence can differ by almost five orders of magnitude. While the proposed approach does not outperform the incumbent state-of-the-art, it performs comparably to other approaches listed in that study. This performance gap shows that there is some benefit to using unsupervised approaches for bot detection, allowing the model to infer different non-overlapping patterns of activity of distinct groups of users, which are not identified in the class-label-aggregated data. Our model uses significantly fewer examples for training compared to the $\mathrm{RT}_{\mathrm{BUST}}$ methods (about 600 labeled examples vs. more than 63 thousand unlabeled cases). Notably, the dataset was manually labeled, which means that any mislabeled cases could have had a significantly larger impact on the supervised approach.

In all of our experiments – on real and simulated data alike – we explored various parameters that can be used to fine-tune the performance of the algorithm. We found that three quantities have a significant impact on the model: the length of the sequence under study, the training set size, and the bandwidth selection algorithm. The former two quantities directly correspond to the sample size in a classical kernel density estimation context (of course, increasing sequence length is only impactful if events are occurring in the new interval). We posit that applying boundary correction methods to the algorithm has no significant impact on the outcome due to the time series timescale, and due the bandwidth parameter being fixed for the entire sequence. For early-events the performance of the model is low because only a few events have actually been observed, regardless of any boundary correction. On the other hand, for late-events the bandwidth seems to be too small to have any significant impact on density estimation at extremely large timescales. It might be interesting to explore this concept and consider adaptive kernel bandwidth in future work.

As a final note, the two-class classification problem studied in this Chapter has a straightforward generalization to the multi-class situation with the class labels denoted as $\{\omega_1, \dots, \omega_c\}$. In fact, the Bayes rule in (2.9) for the $c-$class classification problem reads as

$$\mathbf{X} \in \omega_i \ \text{ if } \ \sum_{s=1}^{N} \log\left(\frac{\lambda_i(t_s)}{\lambda_k(t_s)}\right) \geq \gamma_{ik} \ \text{ for all } \ k = 1, \dots, c, \ \ k \neq i\,, \tag{2.36}$$

where $\gamma_{ik} = \int_0^T (\lambda_i(u) - \lambda_k(u))\, du + \log(\pi_k/\pi_i)$. Here $\{\lambda_i(t)\}$ are the class intensity functions and $\{\pi_i\}$ are the prior probabilities. It is worth noting that the evaluation of the corresponding Bayes risk as it was done in Section 2.2 is more involved.

Chapter 3

Spiking Neural Networks

In recent years artificial neural networks (ANN) have established themselves as a robust and versatile machine learning model, capable of solving increasingly complex problems due to advances in network algorithm design, specialized hardware, and the abundance of data. Despite this, it is difficult to apply these models to purely event data. Such data is non-uniformly distributed in time, particularly because – by definition – it can occur at any moment in time and is rarely periodic. Furthermore, event data is often incomplete, making it ambiguous as to whether nothing important had happened, or whether the information is truly missing. For these reasons, in order to be processed by the ANN, event data must be modified in a way that circumvents its event nature (e.g., by introducing additional features that encode time-of-event or no change; data resampling). A solution that combines recent advances in deep learning with the ability to process event data is a different type of neural network, called a spiking neural network (SNN). In contrast to their ANN counterparts, the SNN model biological neurons in a much more principled way, including their ability to process event streams similarly to biological networks by inducing a neural coding scheme using a corresponding training loss function. By operating directly in the event domain the spiking neural network is capable of leveraging information content of each and every event, which stands in stark contrast to the redundant information introduced as a preprocessing step of a typical ANN. This energy-efficiency is a key driving factor behind the development of event-centric algorithms like the SNN.

In this Chapter we explore a specific sub-type of SNN that processes signals based on the time of each individual event. We take into consideration the model design, properties of the trained network, as well as its applications. Section 3.1.1 introduces several concepts related to biological networks, whereas Section 3.1.2 provides a general overview of the SNN and shows how these concepts are incorporated in the model design. Section 3.1.4 briefly summarizes signal propagation rules of a time-to-first-spike SNN and additionally contains a short study that reproduces the results of the original paper. Thereafter, Section 3.2 discusses limitations identified for the chosen SNN model that stem from its assumptions. The proposed modifications improve upon the model by reducing its processing time, stabilizing training dynamics, allowing a finer control over the spiking activity, and proposing a framework for processing signals varying over time. Finally, Section 3.3 verifies the performance of the modified SNN model on a challenging Twitter bot classification task. Reusing the same dataset from the previous Chapter allows us to compare the SNN approach with the kernel classifier devised from the theory of point processes.

3.1 Introduction

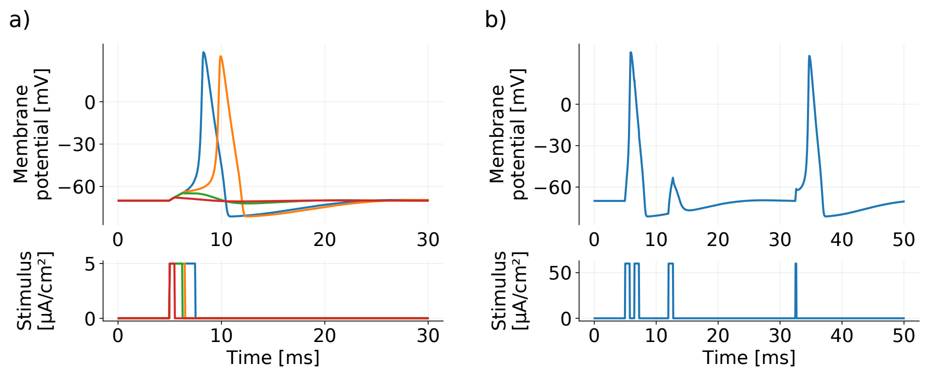
3.1.1 Properties of biological networks

A biological neural network is a structurally complex system composed of neurons connected by synapses, communicating with one another via electrical and chemical signals. Brain tissue is metabolically expensive, primarily due to the need to process various sensory information [81]. This suggests the presence of evolutionary pressure to achieve high energy efficiency in the network [82, 83]. Understanding how individual neurons respond to input stimuli and collectively solve complex tasks is the foundation of neurobiology research.

The biophysical mechanisms responsible for neuronal spiking activity are well-studied [84]. This knowledge can be used to construct neuron models that encapsulate the underlying processes at various levels of detail [11], with the choice of the model being application-specific. For instance, the simplistic Integrate-and-Fire (IF) model [85] can be chosen as the computational unit of an artificial spiking neural network due to its low computational complexity.[a] Conversely, the Hodgkin–Huxley model [86] realistically captures the biophysical phenomena and is more commonly encountered in computational neuroscience research.

In biological neurons the membrane potential describes the difference in the electrical potential between the cell interior and the surrounding extracellular medium. At rest, it is about −70 mV relative to the medium. Ion pumps located in the cell membrane maintain this potential difference. If in response to some input electrical stimulus the membrane potential rises above a certain voltage threshold, the cell responds by generating an action potential (Figure 3.1a). The generated signal can travel without attenuation over large distances along the axon. It is understood that the action potential occurrence time is the carrier of information in neural circuits, therefore all action potentials are customarily treated as identical, discarding any differences in duration, shape or amplitude. This leads to the so-called all-or-nothing principle that states that a neuron can either respond to its input stimulus or not (i.e., no other state is

[a]The simplest variant of an IF neuron model is described in detail in Section 3.1.4.



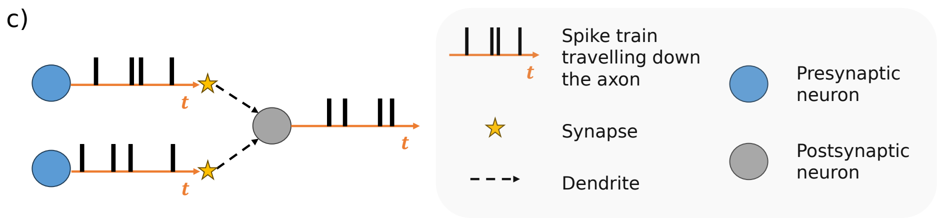


Figure 3.1: Overview of biological neuron properties and concepts. a) Neuron's membrane potential change in response to different input current stimuli, simulated according to the Hodgkin–Huxley model. All stimuli start at the same time. The rapid increase of the membrane potential present in the blue and orange curves is the *action potential*, which is the foundation of neural spike generation. The neuron's response depends on the total electrical charge transferred by the stimulus, therefore by varying the width of the rectangular signal it is possible to observe that some stimuli (red, green) are too weak to induce a spike response. Note that if the stimulus results in the action potential (blue, orange), it has (roughly) the same shape, regardless of the stimulus' intensity. This phenomenon is known as the *all-or-nothing principle*. b) Neuron's membrane potential change in response to a sequence of input current stimuli, simulated according to the Hodgkin–Huxley model. The first three rectangular pulses have the same intensity. Out of these three, only the first stimulus produces an action potential. After spike generation the neuron enters a *refractory period*, which limits its ability to respond to inputs. The second stimulus occurs during an *absolute* refractory period, meaning that no response can be elicited. The third stimulus is presented during a *relative* refractory period in which an action potential could be produced if the stimulus was much stronger. Note that the fourth stimulus is weaker than the others, but is still able to induce a spike response because it occurs outside the refractory period. c) Abstract representation of the connectivity and signal propagation in biological networks. Neurons communicate via neurotransmitters released in the *synapse* in response to spike trains (action potentials) propagated along the axon. Due to the importance of synapse in the information transfer, each neuron can be labeled either *pre-* or *postsynaptic*, depending on its position relative to the signal propagation direction. Note that this distinction is synapse-specific – a postsynaptic neuron can be a presynaptic neuron in other connections.

possible). Once the action potential is generated, the cell enters a brief refractory period in which it is unable to respond to further stimulation (Figure 3.1b).

The action potential travels along the axon until it reaches a connection with another neuron – the synapse (Figure 3.1c). Here the presence of an action potential of the presynaptic neuron might result in a release of a neurotransmitter that binds to the other side of the synapse. This binding causes either an inflow or an outflow of ions in the postsynaptic neuron, depending on the type of the synapse. In artificial neural networks this process is abstracted as a (synaptic) weight, with its sign and magnitude controlling the impact of one neuron on another. The probability of neurotransmitter release and the resulting postsynaptic neuron's conductance change can be influenced by the history of activity at the synapse. This process is known as activity-dependent synaptic plasticity, which plays a crucial role in learning and memory [87, 88]. It is regarded as the neuronal basis of unsupervised learning through spike-timing-dependent plasticity [89].

The study of neural coding attempts to describe how stimulus attributes are represented by action potentials through various temporal patterns [90]. The initial understanding was that cells primarily communicate using rate-coding, that is by varying the firing rate in response to their input [91, 92, 93]. Recently however, the importance of two other modes of operation has been established, namely timing [94, 95, 96] and synchrony coding [97, 98, 99]. The former is related to the precise occurrence time of events while the latter is more common in neural ensembles, describing spike coincidence between multiple event streams or phase-locked, periodic spike sequences. Another plausible hypothesis for information coding used by biological neurons is the rank order coding [100] which forgoes the precise timing information in favor of a relative ordering of spikes generated by presynaptic neurons. Such encoding results in sparse signal processing that is tolerant to variations as long as they preserve the relative order of spikes [101].

Studying neural code is challenging. In addition to the technical difficulties that arise during signal measurement [102], one must also attempt to isolate the features of the response under noisy conditions of an active neural circuit. Furthermore, the neural response varies from trial to trial, even if the exact same stimulus is presented. This implies that it is impossible to predict the timing of each spike deterministically [103]. The randomness of neural responses in *in vivo* measurements can be described in terms of the theory of point processes [104, 105, 106, 107].

3.1.2 Spiking neural networks

Spiking neural networks are considered to be the third generation of neural networks [108] – after artificial neural networks based on dense (or matrix-multiplication) layers and convolutional neural networks (CNN) which mainly use the convolution operation. The SNN process data using impulses which are asynchronously propagated through the entire network. This processing scheme stands in contrast to the classic artificial networks which require that all neurons within a single layer must finish their computation before the signal flows to the subsequent layer. Therefore, it is possible to obtain the model output as soon as the first output spike is available (that is: factoring-in the time delay introduced by the neurons), before the entire input sequence is observed (Figure 3.2). Note that such an early-spiking output is only a coarse approximation, requiring longer observation periods in order to obtain results with higher confidence. Furthermore, the spiking neurons need not respond to the input signal at all, which removes them from the overall model computation. This stands in contrast to the ANN and CNN models in which a neutral response (i.e., zero) still needs to be processed by the subsequent layers. Such properties of the SNN might explain why they are becoming popular candidates for energy-efficient processing units [109, 110].

Overall, the aim of the SNN is to model biological networks in a much more principled way. Depending on the level of detail captured by the SNN, its neurons can be sensitive to the relative timing of inputs, experience a refractory period or even model synaptic plasticity. Most methods used to obtain a functional SNN generally fall into one of three categories: network conversion (training an ANN and mapping each component of the source network to its spiking equivalent) [111, 112, 113, 114, 115], synaptic-plasticity-aware training (taking advantage of long-term potentiation and long-term depression effects) [116, 117], or training with backpropagation [118, 119, 120]. The latter approach necessitates formulating custom training rules that take into account the fact that the activation function of biological neurons – the all-or-nothing principle – is not differentiable. Hybrid conversion-backpropagation approaches have also been explored [121].

Network conversion of the ANN to the SNN attempts to map each neuron of the source network to its spiking equivalent. The main principle behind this process is that the spiking neuron's firing rate corresponds to an analog activation of the ANN neurons. The authors of [112] introduce spiking equivalents to common building blocks of deep neural networks, allowing them to achieve a very close performance to the ANN on a moderately difficult CIFAR-10 dataset. However, they note that the deeper the network is, the smaller the rate of firing becomes and the delay before obtaining reasonably good prediction increases.

In contrast to the network conversion methods, backpropagation-based techniques are not limited to the rate-based encoding (although they are certainly more popular in this context than other neural encoding schemes). One approach to training a rate-coding network using backpropagation is based on designing the training procedure of an ANN in such a way that attempts to model the behavior of the final spiking model [122]. This includes (but is not limited to): changing activation functions of neurons from ReLU (Rectified Linear Unit) to the smooth LIF (Leaky Integrate-and-Fire) or injecting noise to the output of each neuron. The latter trick takes advantage of the fact that rate-coding spiking neurons oscillate around a "true" mean firing rate. A large disadvantage of this approach is that it requires the driving training signal to be either static or uniformly distributed in time.

A more direct application of backpropagation to optimizing the weights of a spiking neural network assumes the differentiability of neuron membrane potential functions (with respect to to the entire presynaptic spiking history) and treats the discontinuity at the spike-generation

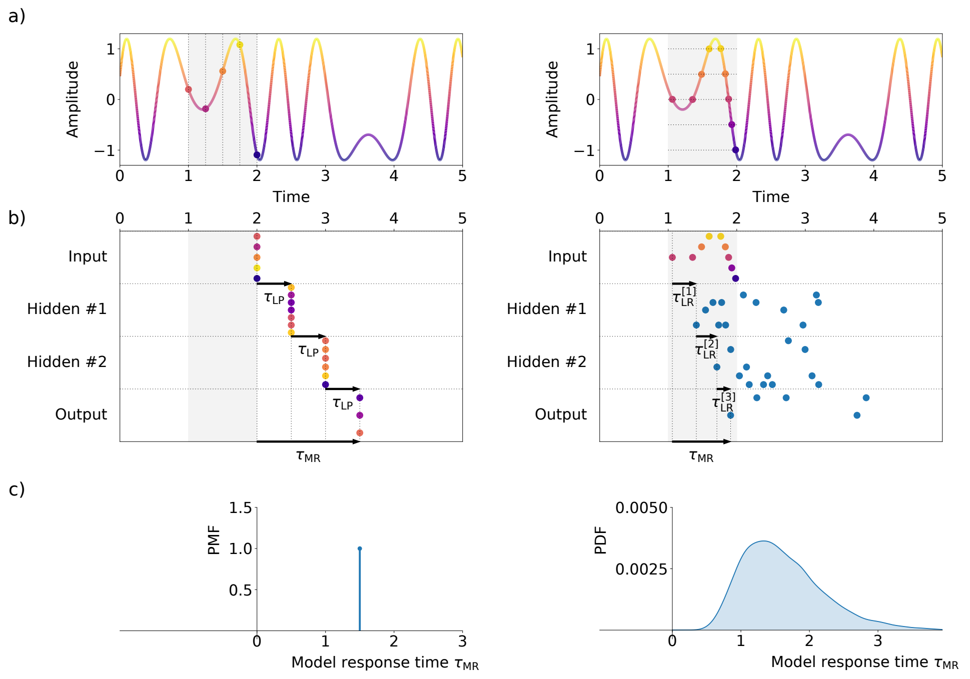


Figure 3.2: Comparison of signal propagation and response time of the ANN (left) and the SNN (right) models. Shaded gray area denotes the analysis window. a) Conversion of a time-varying signal to the representation expected by the model. Left: signal sampling at a fixed time resolution. Each input neuron observes the amplitude at a given point in time. Right: level-crossing sampling at fixed amplitude threshold levels. Each input neuron observes the time when the signal crosses a given threshold. b) Signal propagation through a three-layer network versus time. τ_{LP} denotes layer processing time, τ_{LR} is the layer response time, whereas τ_{MR} is the model response time. Note that τ_{LP}, τ_{LR} and τ_{MR} are exaggerated for illustrative purposes. Left: all neurons within a single layer must finish their computation before the signal flows to the next layer. Each neuron response is color-coded to denote amplitude. The responses in the input layer match the signal amplitude. τ_{LP} is independent of the input signal and is approximately identical for all layers in the network. Right: asynchronous propagation of spikes through the entire network. Layer and model response times τ_{LR}, τ_{MR} are measured relative to the first spike observed at layer or model input, respectively. Note that the SNN is able to produce a model response before the entirety of input signal (within the analysis window) is observed. The input layer responses are color-coded to denote the signal amplitude at event time. In subsequent layers there is no correspondence between the response and the input signal. c) Model response time distributions. For illustrative purposes the dependence of τ_{MR} on hardware running the model is ignored. Left: probability mass function. Model response time depends only on the model computational complexity. Right: probability density function. The distribution is supported on a semi-infinite interval because the left bound depends on the model computational complexity. τ_{MR} is different for each example, hence it is not possible to know in advance how much time exactly it will take to produce a response (although τ_{MR} is bounded from above by the width of the analysis window and neuron-specific time constants).

instant as noise [119]. Such approach has already proven itself to be effective in training the SNN, leading to efficient implementations in computational frameworks [120]. On the other hand, training a timing-sensitive SNN with backpropagation requires a different set of tricks [123], with the crucial one being that during training a neuron is not allowed to spike more than once until a new training example is presented (in other words: the refractory period is much longer than the length of a single training example).

There are several factors that inhibit research on the SNN. One of them is the availability of event data [6, 14, 10]. In case such dataset is unavailable (due to its proprietary nature or when the original signal exists in the analog domain), researchers can opt to transform existing sets of data to an event structure [21, 20, 19]. Note that the SNN models require that their input data is encoded using events, whereas carefully-designed ANN can process either the event-coded, or the original signal. Regardless of how the event data is obtained, it is possible to compare the SNN with nonspiking models trained on the same task. Another inhibiting factor is the computational complexity of simulating the SNN on typical hardware. For this reason, the research on spiking neural networks goes hand-in-hand with the development of dedicated neuromorphic hardware [124].

Despite being a relatively new field of research, the SNN have already been applied to a variety of tasks. In [125] a spiking neural network was used to train an electrocardiographic (ECG) arrhythmia classifier. Input data stream is converted to events related to a relative increase or decrease in signal amplitude (i.e., two event types). This naturally results in a variable firing rate over the data stream, with its maximum corresponding to a QRS complex. The input event sequence is then processed by a recurrent SNN that, interestingly enough, is not trained, instead relying on the initial weights sampled from some pre-defined distribution. Finally the output of the recurrent SNN is connected to a readout layer comprised of several leaky integrate-and-fire neurons, trained to discriminate target classes using the Support Vector Machine (SVM). During inference a winner-takes-all strategy is employed to determine a class based on the output spiking neuron membrane potential. A similar idea has been also applied to hand-gesture recognition from an electromiography (EMG) recording [126].

An example of a model employing a biologically-inspired learning rule is the sound classification pipeline presented in [117]. Events were extracted from the raw signal by fitting a Self-Organizing Map (SOM) on Mel-frequency cepstral coefficients (MFCC) computed for an audio signal frame. Spike trains represent SOM activations in each consecutive window and are used to drive output readout neurons trained using a modified Tempotron rule [116]. One can easily notice that the event stream defined in this way is uniformly distributed in time, which allowed the authors to compare the spiking architecture with recurrent neural networks trained on the same task.

Finally, a visual-scene detection model from [127] is trained by converting an output layer logistic regression classifier to the spiking domain. In contrast to previously presented applications, this model directly operates on the spiking data from a neuromorphic Dynamic Vision System (DVS) with raw spike trains convolved with a set of Gabor filters and subsampled prior to reaching readout layer.

3.1.3 Training the SNN with backpropagation

Training artificial spiking neural networks with backpropagation leverages existing methods, algorithms and best practices developed for artificial nonspiking neural networks and deep learning. In doing so it forgoes biological plausibility, evident in training methods based on spike-timing-dependent plasticity effects such as the ReSuMe algorithm [128, 129]. One of the first proposed gradient-descent-based learning rules for timing-encoding networks was the SpikeProp algorithm [130] and its variants. It trains a multilayer feedforward network of neurons firing a single spike. SpikeProp defines the learning rules in terms of the dynamics of a general Spike Response Model (SRM) of neurons. The original SpikeProp algorithm has since been extended to allow neurons in the input, hidden [131] and output [132] layers to spike multiple times in response to input patterns.

The SpikeProp paper established key traits of backpropagation algorithms present in subsequent works. In particular, the algorithm has the following characteristics: 1) the SNN is simulated over a finite time window with a fixed time step, recording changes in the internal state (membrane voltage, synaptic current) of all neurons during the forward pass and 2) approximating (smoothing) the spike-generation function to avoid a discontinuous derivative (unless this discontinuity is ignored and treated as noise, as in [119]). Backpropagation-based approaches for training time-coding SNN can thus be divided into two categories, depending on how much information from the forward pass is needed to compute the backward pass. In event-driven learning the error is propagated only through spikes [133, 134, 135, 136]. Most notably, EventProp [137] defines exact gradients and can be applied to neuron models without an analytical expression for the postsynaptic potential kernels. Nevertheless, similarly to other existing algorithms, EventProp still requires simulating the network in the forward pass in order to compute postsynaptic events [138]. As these algorithms propagate the error only through spike events, they are prone to failing to converge when the network does not generate enough events to process the signal end-to-end.

The other category that stands in contrast to the event-driven learning is the RNN-like learning (named after its similarity to nonspiking artificial recurrent neural networks), in which the error information is also propagated through the computation time steps which did not elicit a spike [139, 140]. The SuperSpike [141] and SLAYER [142, 143] models are examples of such algorithms. Surrogate gradient methods, commonly used in the RNN-like learning, are an alternative approach for overcoming the discontinuous derivative of the spike-generating function [144]. Typically, a standard backpropagation through time algorithm is used, as in the RNN, with one minor modification: a continuously differentiable function is used in the backward pass as a surrogate of the spike-generation function derivative. Finding the optimal surrogate gradient function is a topic of an ongoing research [145, 146]. Despite achieving a remarkable success in training deep SNN [147, 148], surrogate gradient methods represent an even further departure from energy-efficient, biologically-inspired learning.

Recently, there has been some research conducted on the single-spike time-to-first-spike SNN that specify learning rules which do not require simulating the network over time [123, 149, 150, 151]. Concretely, [123] trains an SNN with simplistic Integrate-and-Fire (IF) neurons by deriving locally exact gradients of the spike-generating function. A similar idea is explored in [150] where the instantaneous synaptic current kernel function is used instead of an exponentially decaying kernel. Following up on these works, [149] shows how this algorithm can be applied to the more general SRM neuron with an alpha function of synaptic transfer. Furthermore, [151] applies popular deep learning techniques such as max-pooling and batch normalization to this model type, which alleviates issues with training deeper neural architectures. However, to the best of our knowledge, there has been no research that shows how this model can be applied to actual spike trains rather than single spikes.

3.1.4 Signal propagation in the time-to-first-spike SNN

In the previous Section, the time-coding single-spike time-to-first-spike SNN group of models has been briefly introduced. These models are sensitive to the timing of input events and not their rate. Furthermore, they are truly event-centric, evaluating the state of the network at each generated event rather than at fixed points in time determined by the simulation grid. They are able to do so because they rely on a set of equations that determines when each postsynaptic neuron elicits its first spike. For these reasons, they were the focus of the studies conducted in this thesis.

Let us briefly summarize the model first described by Mostafa [123]. This type of network uses the IF neurons with exponentially decaying synaptic current kernels. The membrane voltage of an IF neuron with C presynaptic neuron connections is governed by the differential equation

$$\frac{dV(t)}{dt} = \sum_{c=1}^{C} w_c i_c(t)\,, \tag{3.1}$$

where w_c is the weight associated with the c-th synapse (channel), and $i_c(t)$ is the synaptic current driving signal. Assuming that every presynaptic neuron observes a single event at time t_c, the presynaptic current is

$$i_c(t) = \exp\left(-\frac{t - t_c}{\tau_{\text{syn}}}\right) u(t - t_c)\,, \tag{3.2}$$

with τ_{syn} being the synaptic current time constant and $u(t)$ being the step function. The solution to the system of equations defined by (3.1) and (3.2) is given by

$$V(t) = V_0 + \tau_{\text{syn}} \sum_{c=1}^{C} w_c \left[1 - \exp\left(-\frac{t - t_c}{\tau_{\text{syn}}}\right)\right] u(t - t_c)\,, \tag{3.3}$$

where $V_0 = V(0)$ is the initial membrane voltage. The neuron is said to fire at time t_{out} if the voltage crosses a threshold V_{thr} from below, i.e., $t_{\text{out}} = \arg\min_t V(t) \geq V_{thr}$, after which the IF neuron becomes unresponsive to input signals for some time, called the refractory period τ_{ref}. Once the refractory period subsides, the voltage is reset to zero and the summation over impulses in (3.3) can resume. For now let us consider the case $\tau_{\text{ref}} \longrightarrow \infty$, meaning that the neuron is unable to respond to new inputs following the output spike generation. This is an implicit assumption in single-spike SNN [123, 149, 150, 151].

Assuming without the loss of generality $V_0 = 0$ and that there exists a subset of input spikes that cause the postsynaptic neuron to fire

$$Q = \{c : t_c < t_{\text{out}}\}\,, \tag{3.4}$$

i.e., the *causal set* of input spikes, the solution for t_{out} is given in the implicit form

$$z_{\text{out}} = \frac{\sum_{c \in Q} w_c z_c}{\sum_{c \in Q} w_c - \frac{V_{\text{thr}}}{\tau_{\text{syn}}}}\,, \tag{3.5}$$

where

$$z_c(t) = \exp\left(\frac{t_c}{\tau_{\text{syn}}}\right),\quad z_{\text{out}}(t) = \exp\left(\frac{t_{\text{out}}}{\tau_{\text{syn}}}\right). \tag{3.6}$$

For completeness we assign $z_{\text{out}} = \infty$ when $Q = \varnothing$. A necessary condition for the postsynaptic neuron to fire is that the sum of weights of the causal set of neurons be strictly larger than the scaled threshold voltage [123], i.e., that

$$1 \leq z_{\text{out}} < \infty \iff \sum_{c \in Q} w_c > \frac{V_{\text{thr}}}{\tau_{\text{syn}}}\,. \tag{3.7}$$

Note that the condition in (3.7) assures that the right-hand side of (3.5) is positive. The formula (3.5) is differentiable with respect to the transformed input spike times $\{z_c\}$ and synaptic weights $\{w_c\}$ with partial derivatives

$$\frac{\partial z_{\text{out}}}{\partial z_c} = \begin{cases} \frac{w_c}{\sum_{c \in Q} w_c - \frac{V_{\text{thr}}}{\tau_{\text{syn}}}} & \text{if } c \in Q \\ 0 & \text{otherwise} \end{cases}, \tag{3.8}$$

$$\frac{\partial z_{\text{out}}}{\partial w_c} = \begin{cases} \frac{z_c - z_{\text{out}}}{\sum_{c\in Q} w_c - \frac{V_{\text{thr}}}{\tau_{\text{syn}}}} & \text{if } c \in Q \\ 0 & \text{otherwise} \end{cases}, \tag{3.9}$$

therefore it can be used to train a spiking network using the backpropagation algorithm.

The original paper showcases the earlier described spiking neural network model in terms of two classification problems. The following Sections briefly summarize the conducted reproducibility study on those same machine learning tasks. This was an important step in the overall context of analyzing this time-to-first-spike SNN: first it assessed the correctness of our own implementation (as no code repository was shared publicly by the authors of [123]), and further on, those results could be used as a reference point when discussing the limitations and extensions of the model in subsequent Sections.

3.1.4.1 XOR task

The first scenario used to evaluate the proposed spiking neural network model was an SNN version of training a dense layer to tackle the XOR task. A model behaving as a XOR logic gate must have the following properties:

1) Two boolean inputs (0 or 1),
2) One boolean output: returning the value 1 if either one of its inputs is 1 while the other is 0, and returning 0 otherwise.

Notably, XOR logic is nonlinear, i.e., it cannot be represented by a neural network without any nonlinear activation function acting on layer outputs. One possible interpretation of the XOR logic in a spiking context is determining *whether inputs spike at the same time or not.* Of course, it is debatable whether this means that the events must occur at the exact same time, or whether there is some tolerance. Importantly, both inputs must have an associated spike time, as having no events would make the model wait infinitely long before producing its decision. In any case, the authors of [123] considered only event coincidence in the strict sense, and built a two-layer model with 4 neurons in the hidden layer and 2 neurons in the output layer. As the entire SNN model operates on the principle of the time of the first spike (3.5), spikes produced by the output layer are compared with one another in order to produce the final decision, determining which output neurons spiked first. This comparison between output spikes constitutes an example of biological rank-order coding [100] in neural circuits. The XOR task along with the SNN model is summarized in Figure 3.3.

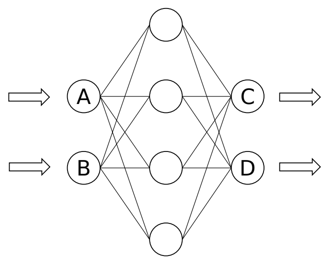
The loss function used to train the SNN has two components. The first one is a modification of the cross-entropy loss with softmax normalization:

$$L(z, y) = -\sum_{p=1}^{P} y_p \ln\left(\frac{\exp(-z_p)}{\sum_{p=1}^{P} \exp(-z_p)}\right), \tag{3.10}$$

a)

p	q	p XOR q
0	0	0
0	1	1
1	0	1
1	1	0

b)



c)

A	B	Expected output
t_0	t_0	$t_C > t_D$
t_0	t_1	$t_C < t_D$
t_1	t_0	$t_C < t_D$
t_1	t_1	$t_C > t_D$

Figure 3.3: a) Truth table of the XOR operator applied to two Boolean variables P, Q. b) Schematic of the SNN trained on the XOR task. c) Training objective of the XOR-SNN. Hyperparameters t_0 and t_1 are spike times associated with input neurons A and B, and are fixed throughout the entire training process. The network is trained so that outputs C and D spike at times t_C, t_D such that one of them is earlier than the other.

where:

- P – the number of output channels (for XOR task $P = 2$),
- y_p – a binary indicator (0 or 1) of the desired output channel p spiking first,
- z_p – the transformed spike time of the p-th output channel.

This loss function component encourages the model to use rank-order coding to represent the output, without explicitly specifying the time of each spike (which would be necessary when minimizing a loss function based on Mean Squared Error between the output spikes train and some reference). The second component of the loss function is a spike regularization term, which promotes network spiking activity by ensuring a nonnegative denominator of (3.5)

$$R_{\text{spiking}} = \sum_{h=1}^{H} R_h \,, \tag{3.11}$$

where $R_h = \max\left(0, \frac{V_{\text{thr}}}{\tau_{\text{syn}}} - \sum_{c=1}^{C_h} w_{ch}\right)$ and the index h runs over all neurons in the network H. The overall loss function minimized by the model is

$$L_{\text{total}}(z, y) = \frac{1}{N}\sum_{n=1}^{N} L_n(z, y) + \gamma R_{\text{spiking}} \,, \tag{3.12}$$

where $L_n(z, y)$ is the cross-entropy loss (3.10) for the n-th example of the batch of size N,[b] and γ is a hyperparameter.

Given a toy-problem nature of the XOR task, the model was trained with a total of four input examples, setting the two input spike parameters outlined in Figure 3.3 to $t_0 = 0$ and $t_1 = 2\ \tau_{\text{syn}}$. Notably, the first and the last case are the same (both inputs spiking at the same time), just shifted slightly in time. Additionally, the spike times are specified relative to the synaptic time constant τ_{syn} to avoid training convergence problems occurring when τ_{syn} is

[b]The term "batch" or "minibatch" denotes the collection of examples presented as network layer input at the same time. In this work we use these terms interchangeably.

chosen incorrectly relative to the input spikes. The IF-neuron-specific parameters were fixed to $\tau_{\text{syn}} = 1$ and $V_{\text{thr}} = 1$. The training proceeded over 1000 epochs with a learning rate $\lambda = 0.06$ and the synaptic regularization parameter $\gamma = 50$.

For such a simple spiking neural network model, it is possible to plot how the membrane voltage of each individual neuron changes in response to input spikes. An example of such spike-voltage plot for a trained model is presented in Figure 3.4. It shows that the network correctly resolves the XOR logic based on the concept of "early" and "late" input spikes. One can also notice that shifting the input spikes in time causes the model to respond in the exact same way, just at a different time. This means the proposed SNN has no notion of absolute time, instead operating on the time relative to the first input spike. As a result of this property, merely shifting spike trains in time cannot be used as a data augmentation method when training such an SNN, although for this specific model it does have a measurable impact on training dynamics. This topic shall be further discussed in Section 3.2.2. Overall, successfully training the SNN model to behave as a XOR logic gate shows that it is nonlinear, with its nonlinearity induced by the causal set of input spikes.

3.1.4.2 MNIST classification task

The other task used to evaluate the time-to-first-spike SNN model was the MNIST digit classification [152]. This dataset is composed of 28×28 grayscale images of ten digits. The MNIST dataset is almost class-balanced and has a predefined split into 60,000 training images and 10,000 test images. It is commonly used when evaluating models for image recognition due to its simplicity – newly proposed ideas that do not successfully train a model on MNIST have little hope of solving any reasonably challenging problem. Additionally, choosing a dataset that is widely known and easily accessible means that the results of research can be readily reproduced.

In order to analyze the MNIST images by the SNN model, they first need to be converted into the spiking domain. While there exists an event-based Dynamic Vision Sensor (DVS) variant of MNIST [153], using it prevents researchers from comparing the results with those achieved by nonspiking models. And so, the authors [123] chose to preprocess the images of the MNIST dataset in the following way:

1) Each 28×28 image is first flattened into a 784-element vector.
2) The vector representation of each image is binarized with a threshold of 50% of global maximum pixel intensity.
3) One of two time instants $t_0 = 0$ or $t_1 = 1.79\ \tau_{\text{syn}}$ is assigned to the value of each bit (white and black pixels, respectively), additionally setting $\tau_{syn} = 1$.

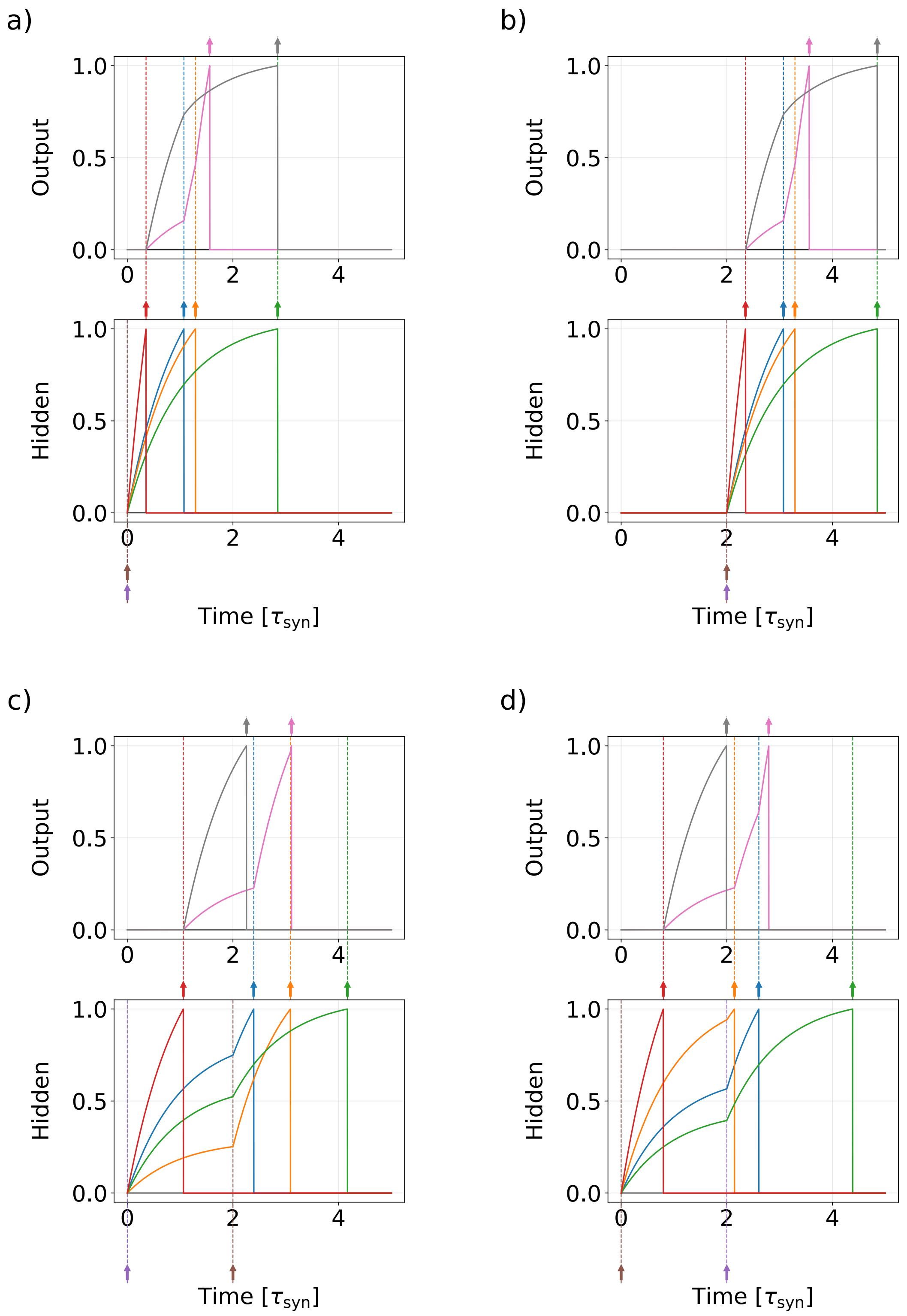


Figure 3.4: Spike-voltage plots of the time-to-first-spike SNN trained on the spiking XOR task. Each color is associated with a different neuron. Dashed lines and arrows denote spike events. a) 0 XOR 0 = 0, b) 1 XOR 1 = 0, c) 0 XOR 1 = 1, d) 1 XOR 0 = 1. Note that the voltage responses for scenarios (a) and (b) are exactly the same, just shifted in time.

The evaluated SNN was a three-layer feedforward 784-400-400-10 network (denoting the number of neurons in input, hidden, and output layers, respectively). The original study additionally considered a 784-800-10 network, but the former architecture was chosen as the focus of this reproducibility study as multilayer networks are more difficult to train due to the vanishing gradient problem [154]. The model was trained to minimize the loss function composed of three terms

$$L_{\text{total}} = \frac{1}{N}\sum_{n=1}^{N} L_n(z, y) + \gamma R_{\text{spiking}} + \lambda \sum_{h=1}^{H}\sum_{c=1}^{C_h} w_{ch}\,, \tag{3.13}$$

with the first two being the same as in the XOR task (3.12), whereas the last component is the L_2 regularization parameterized by λ. The model was optimized over 5000 iterations in batches of 128 examples, with a learning rate of 10^{-3}, a synapse regularization parameter $\gamma = 0.008$, and L_2 regularization parameter set to $\lambda = 5 \cdot 10^{-5}$. Similarly to the original study, the gradients are clipped so that the Frobenius norm of the gradient with respect to weights is at most equal 10. The parameters related to the IF-neuron were the same for all neurons in the network: $\tau_{\text{syn}} = 1$ and $V_{\text{thr}} = 1$.

We have successfully trained a network that achieves an F_1-score of 0.971, computed over the test set examples, which is equal to the result reported in the original study. Additionally, Table 3.1 shows that both models perform similarly to what other researchers report when training the SNN classifiers on the MNIST data. Finally, by analyzing the empirical distribution of spike times in the hidden and output layers, we verified the ability of the trained model to elicit an output spike as soon as the signal is propagated through the network, even if some of the neurons in the hidden layer produce a spike much, much later (Figure 3.5). This also shows that, in general, the time-to-first-spike SNN does not need many spikes to produce a response. This property shall be discussed in more detail in Section 3.2.3.

3.2 Overcoming the limitations of the model

The time-to-first-spike SNN model introduced in previous Sections presents an interesting concept of an artificial network able to process information on a level of each individual event. Moreover, it formulates a set of rules that allows training a network end-to-end in the spiking domain, which circumvents the need for input data to exist in a traditional, nonspiking domain. While it might be tempting to employ this model as-is to solve some nontrivial machine learning problem and report on the properties of the trained SNN, it is important to take a step back and assess its capabilities and limitations. Therefore, this Section presents an overview of identified shortcomings of the model that stem from algorithmic design, as well as established

Table 3.1: Classifier performance of different SNN and the nonspiking k-NN baseline on MNIST (best reported results from each paper).

Input encoding	Model type	Performance
-	k-NN Euclidean baseline [152]	0.950
firing rate	Spiking RBM [155]	0.926
	STPD-trained network [156]	0.950
	Spiking NN [157]	0.986
	Spiking CNN [157]	**0.991**
spike-time	Spiking NN (784-800-10) [123]	0.975
	Spiking NN (784-400-400-10) [123]	0.971
	Spiking NN (784-400-400-10) (our)	0.971

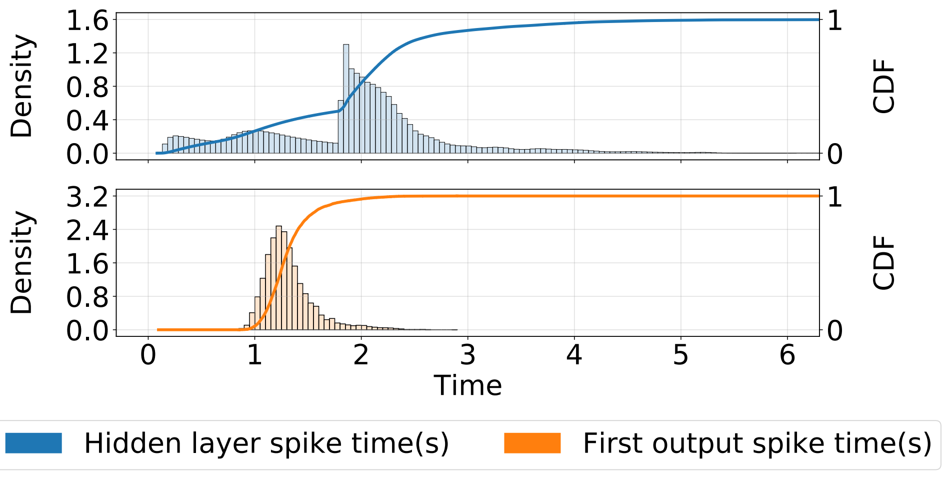


Figure 3.5: Histograms of spike times in the hidden and output layers across the MNIST test set images. Almost all first output layer spikes, i.e., the ones that determine the classifier prediction, are produced before the rest of the network elicits a response.

assumptions. Each limitation is discussed in detail and is accompanied by a proposed set of modifications that address the issue. The impact of the proposed changes is also assessed.

Throughout the rest of this document we refer to a single layer of the time-to-first-spike SNN as a *spiking dense layer*. This serves as a reminder that, similarly to the ANN, spiking neurons are also organized in layers, and that the computation conducted by a single layer is analogous to the dense (matrix multiplication) layer in nonspiking networks.

3.2.1 Reducing the layer processing time

One challenging aspect of the algorithm summarized in Section 3.1.4 is the requirement to find the causal set of input neurons for a single output neuron. In practical applications multiple neurons form a single layer, and multiple sequences are passed as layer input at the same time during training to take advantage of minibatch stochastic gradient descent-based optimization of network weights. In such scenarios one could take a naïve approach to find the causal sets of all output neurons for all input sequences, but – as presented in the flowchart in Figure 3.6a – that would necessitate creating several loops that would negatively impact the computational complexity of the algorithm. Importantly, when searching for the causal set it is not enough to find the (chronologically) first set of inputs that would result in the output spike generation. It is necessary to check additional input spikes to see if they could have contributed to the spike generation (or even suppress the output altogether). A concrete example of this nontrivial problem is presented in Figure 3.6b.

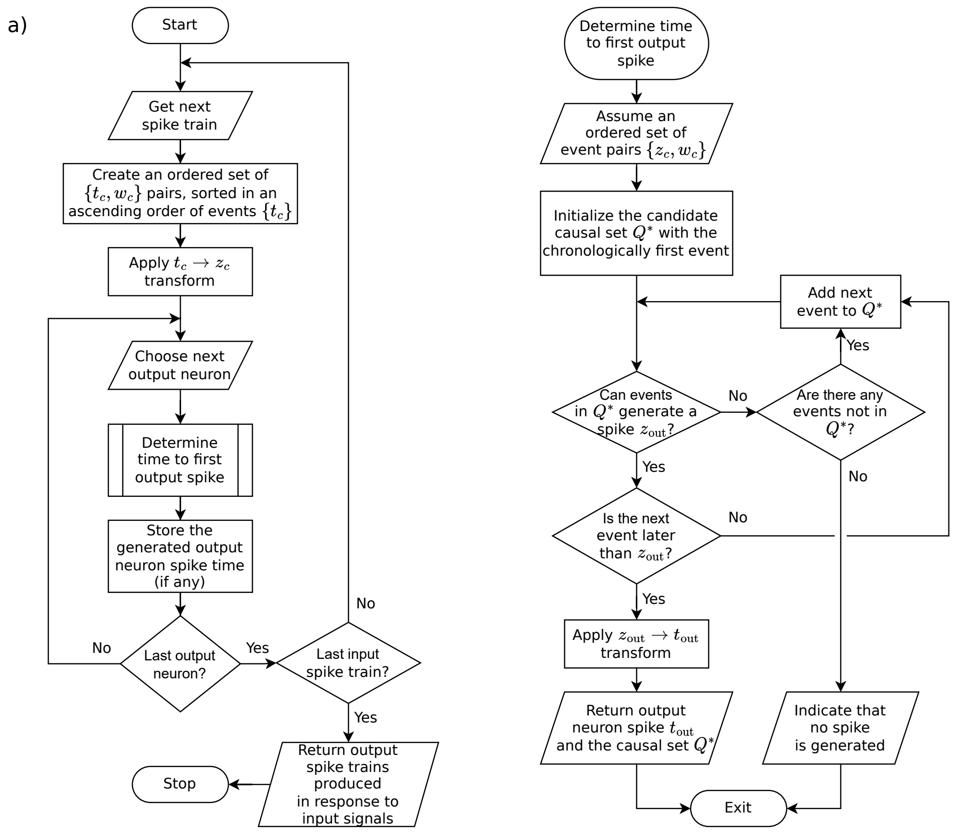
However, it must be noted that searching for the causal set is neuron- and example-specific, and so the two outermost sequential loops in Figure 3.6a could be substituted with running the algorithm in parallel. Furthermore, instead of iteratively finding the combination that would result in a causal set, we could check *all* combinations and return the the one that results in the earliest t_{out}. These observations form a conjecture that it is possible to propose a vectorized implementation of the spiking dense layer that can be plugged into modern deep learning frameworks such as TensorFlow [158] or PyTorch [159].

The aim of the following Sections is to describe an algorithm that does the computation in a single pass over data, without any explicit, high-level loops. The main challenge with vectorizing this algorithm is the causal set selection step, because it depends not only on the weights matrix, but also the actual input sequence. Hence, in contrast to regular dense layers, the spiking dense layer must be described in terms of third-order tensors instead of matrices.

Note that the presented algorithm has been developed independently from the one briefly mentioned in [151], albeit the general structure of the algorithm is the same. Our work is much more verbose, describing the algorithm in detail in terms of tensor algebra with a focus on edge-case scenarios. We also quantify the improvement over the original approach.

3.2.1.1 Vectorized algorithm highlights – computation of the causal set

Before diving deeper into the tensor algebra description of the vectorized algorithm, let us provide a high-level overview of the computation, highlighting how it relates to the original description in [123]. As mentioned earlier, iteratively finding the input event combination that would result in a causal set is the computational bottleneck of the network. For this reason, we


a)
Start
Get next spike train
Create an ordered set of $\{t_c, w_c\}$ pairs, sorted in an ascending order of events $\{t_c\}$
Apply $t_c \to z_c$ transform
Choose next output neuron
Determine time to first output spike
Store the generated output neuron spike time (if any)
Last output neuron?
No
Yes
Last input spike train?
No
Yes
Return output spike trains produced in response to input signals
Stop
Determine time to first output spike
Assume an ordered set of event pairs $\{z_c, w_c\}$
Initialize the candidate causal set Q^* with the chronologically first event
Add next event to Q^*
Can events in Q^* generate a spike z_{out}?
No
Yes
Are there any events not in Q^*?
Yes
No
Is the next event later than z_{out}?
No
Yes
Apply $z_{\text{out}} \to t_{\text{out}}$ transform
Return output neuron spike t_{out} and the causal set Q^*
Indicate that no spike is generated
Exit

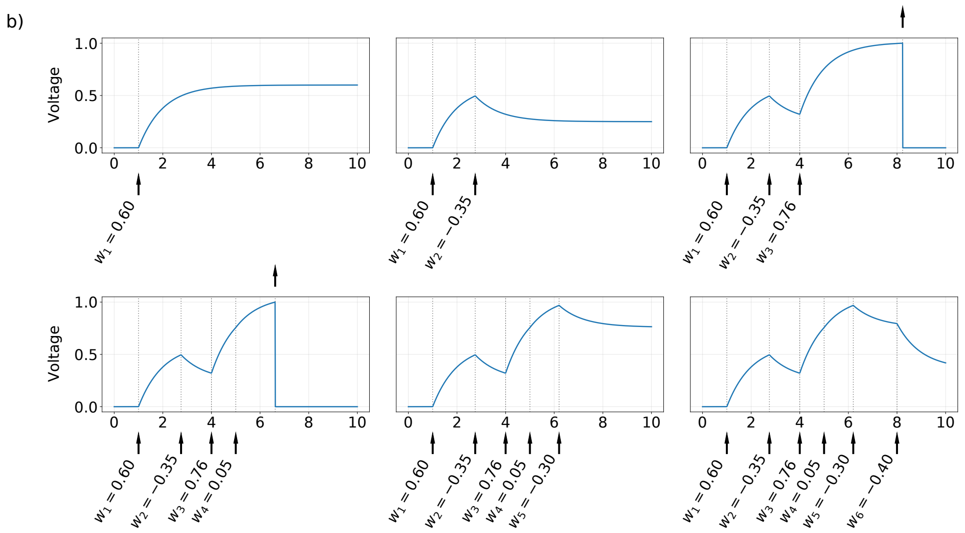


Figure 3.6: a) Flowchart of the naïve implementation of the spiking dense layer. The two exit conditions at the bottom right imply that $Q \triangleq Q^*$ when $t_{\text{out}} < \infty$ and $Q = \varnothing$ otherwise. b) Iterative search for the causal set of inputs for an output neuron parameterized by $\tau_{\text{syn}} = 1$ and $V_{\text{thr}} = 1$. X-axis denotes time as multiples of τ_{syn}. Each plot describes the change in membrane voltage as additional event pairs $\{z_c, w_c\}$ are considered when constructing Q^*. Note that for this input sequence the neuron does not generate an output spike, despite there being two combinations (with 3 or 4 events) that could elicit a spike.

shall focus on this part of the algorithm in this Section. The two outermost loops – repeating the aforementioned computation for every output neuron and every input example pairwise combination – can be unrolled (executed in parallel) because their results are independent of any other pair.

For now let us assume that the event sequences are already sorted and transformed according to (3.6), resulting in vectors $\widetilde{\mathbf{z}}$ for the input spikes and $\widetilde{\mathbf{w}}$ for the synaptic weights (with the tilde diacritic highlighting that vectors are indexed such that events in $\mathbf{z}$ are sorted).[c] This is a necessary preprocessing step regardless of the choice of the algorithm. Then, Algorithm 3.1 presents the pseudo-code description of the iterative causal set search. We have already shown in Figure 3.6b that the iterative search cannot stop early even if the denominator of the current candidate output spike z^*_{out} is positive, i.e., the necessary condition in (3.7).[d] That is because it must also be larger than all events indexed by the current candidate causal set $\widetilde{Q}^*$ and, at the same time, smaller than the next input event $\widetilde{u}$. This requirement arises from the sufficient condition implied by the definition of the actual causal set in (3.4).

Algorithm 3.1: Iterative causal set search for a single output neuron observing an event sequence.

```
Input: $\widetilde{\mathbf{z}}$ # vector of sorted input events of length $C$
Input: $\widetilde{\mathbf{w}}$ # vector of synaptic weights associated with elements in $\widetilde{\mathbf{z}}$
Output: $\widetilde{Q}$ # causal neuron set
Output: $z_{\text{out}}$ # transformed output spike time
Begin:
        $Q^* \leftarrow \{1\}$ # initialize the candidate causal set with the first event
        $z_{\text{out}} \leftarrow \infty$ # assign non-event as the candidate output spike
        For $c = 1, \ldots, C$:
                $z^*_{\text{out}} \leftarrow \frac{\sum_{k \in \widetilde{Q}^*} \widetilde{w}_k \widetilde{z}_k}{\sum_{k \in \widetilde{Q}^*} \widetilde{w}_k - \frac{V_{\text{thr}}}{\tau_{\text{syn}}}}$ # new output spike proposal
                # find the next input spike
                If $c = C$:
                        $\widetilde{u} \leftarrow \infty$
                Else:
                        $\widetilde{u} \leftarrow \widetilde{z}_{(c+1)}$
                End If
                # check the necessary and sufficient conditions
                If $\sum_{k \in \widetilde{Q}^*} \widetilde{w}_k > \frac{V_{\text{thr}}}{\tau_{\text{syn}}} \wedge \widetilde{z}_c < z^*_{\text{out}} < \widetilde{u}$:
                        $z_{\text{out}} \leftarrow z^*_{\text{out}}$ # current candidates result in a valid output spike
                        Break
                Else:
                        $\widetilde{Q}^* \leftarrow \{1, \ldots, c\}$ # update the candidate causal set
                End If
```

[c]The purpose of the tilde diacritic is twofold. Firstly, it makes it explicit that the quantities we compute are in the “sorted-events domain”. To properly assign the credit to events during backpropagation it is important to keep track of these indices. Secondly, it makes this description consistent with Sections 3.2.1.2.2-3.2.1.2.3 which actually use both the sorted and the original event domains.

[d]Scalars z^*_{out} and z_{out} are written without the tilde diacritic because their values do not depend on the ordering of elements in $\mathbf{z}$.

```
        End For
        If $z_{\text{out}} < \infty$:
            $\widetilde{Q} \leftarrow \widetilde{Q}^*$
        Else:
            $\widetilde{Q} \leftarrow \varnothing$ # the causal set is empty
        End If
        Return $\widetilde{Q}, z_{\text{out}}$
End
```

To show how a different approach can be used in lieu of the iterative search over events, we first note that knowing the entire input event sequence alongside the synaptic weights is enough to determine all possible output spike times. Before showing how this information can be leveraged to obtain the actual output spike time, let us first introduce several element-wise operators. Denote

$$\mathbf{b} = \text{cumsum}\left(\mathbf{a}\right) \tag{3.14}$$

as the *cumulative sum* of vector $\mathbf{a}$ such that

$$b_i = \sum_{j \le i} a_j \,. \tag{3.15}$$

Furthermore, the *Hadamard product* between vectors $\mathbf{a}$ and $\mathbf{b}$ is

$$\mathbf{e} = \mathbf{a} \odot \mathbf{b}\,, \tag{3.16}$$

with elements

$$e_i = a_i b_i \,. \tag{3.17}$$

Similarly, the *Hadamard division* is

$$\mathbf{f} = \mathbf{a} \oslash \mathbf{b} \tag{3.18}$$

with elements

$$f_i = \frac{a_i}{b_i} \,. \tag{3.19}$$

With these operators we can define $\widetilde{\mathbf{d}} = \text{cumsum}\left(\widetilde{\mathbf{w}}\right) - \frac{V_{\text{thr}}}{\tau_{\text{syn}}}$ and $\widetilde{\mathbf{m}} = \text{cumsum}\left(\widetilde{\mathbf{w}} \odot \widetilde{\mathbf{z}}\right)$. Then,

$$\widetilde{\mathbf{g}} = \widetilde{\mathbf{m}} \oslash \widetilde{\mathbf{d}} \tag{3.20}$$

represents the neuron response time in (3.5) computed for all combinations of input events (i.e., $\widetilde{g}_1$ is the output spike time in response to event $\widetilde{z}_1$; $\widetilde{g}_2$ is the response to events $\widetilde{z}_1$ and $\widetilde{z}_2$, etc.). As mentioned earlier, not all combinations of input events elicit a response, thus it is necessary to determine which elements of $\widetilde{\mathbf{g}}$ represent valid responses – ones that would generate an actual spike.

Let $\widetilde{\mathbf{u}}$ represent the time of the next input spike relative to $\widetilde{\mathbf{z}}$ with elements

$$\widetilde{u}_c = \begin{cases} \infty & \text{if} \quad c = C \\ \widetilde{z}_{(c+1)} & \text{if} \quad c < C \quad \wedge \quad \widetilde{z}_{(c+1)} > \widetilde{z}_c \\ -\infty & \text{otherwise} \end{cases}, \tag{3.21}$$

where C is the number of events in $\mathbf{z}$. This definition captures the following scenarios:

- $\widetilde{u}_c = \infty$ implies that $\widetilde{z}_c$ is the last event of the spike sequence (i.e., the "next event" does not exist),
- $\widetilde{u}_c = -\infty$ means that two consecutive events $\widetilde{z}_c, \widetilde{z}_{(c+1)}$ occur at the same time.

Note that $\widetilde{\mathbf{u}}$ must contain exactly one non-event indicator. Then, the set of indices that denote the valid elements of $\widetilde{\mathbf{g}}$ is

$$\widetilde{\Omega} = \{c : \ \widetilde{d}_c > 0 \ \wedge \ \widetilde{g}_c > 0 \ \wedge \ \widetilde{z}_c < \widetilde{g}_c < \widetilde{u}_c\}\,. \tag{3.22}$$

The three conditions enforce that

1) the sum of weights is larger than the scaled voltage threshold (3.7),
2) the output spike candidate has a valid domain so that the inverse transform of (3.6) mapping $z \longrightarrow t$ exists, and
3) the spike causality principle holds, i.e., the candidate spike $\widetilde{g}_c$ occurs later than the most recent input spike $\widetilde{z}_c$, but earlier than the next input spike $\widetilde{u}_c$,

respectively. It follows that the smallest index in $\widetilde{\Omega}$, that is $c^* = \min \widetilde{\Omega}$, can be used to obtain the actual output spike. Therefore, the causal set is

$$\widetilde{Q} = \begin{cases} \{c : c \leq c^*\} & \text{if} \quad \widetilde{\Omega} \neq \varnothing \\ \varnothing & \text{otherwise} \end{cases}, \tag{3.23}$$

and similarly $z_{\text{out}} = \widetilde{g}_{c^*}$ if $\widetilde{\Omega} \neq \varnothing$ and $z_{\text{out}} = \infty$ otherwise.

The interpretation of vectors $\widetilde{\mathbf{z}}$, $\widetilde{\mathbf{w}}$ and $\widetilde{\mathbf{g}}$ is that they fully describe the neuron input-output relationship for all combinations of input events in $\mathbf{z}$. This is similar to simultaneously computing all neuron membrane voltage curves in Figure 3.6b for different subsets of events. In this specific example the elements $\widetilde{g}_3$ and $\widetilde{g}_4$ are positive (meaning that these combinations of events could generate a response), but ultimately $\widetilde{\Omega} = \varnothing$ due to the spike causality principle.

Algorithm 3.2 summarizes the aforementioned vectorized computation. Note that computing the next spike according to (3.21) is a matter of shifting the input sequence and checking for simultaneous events. Contrast this with the description in Algorithm 3.1. By removing the high-level loop that iterates over events we obtained an algorithm that is easier to implement. Additionally, as we will show in Section 3.2.1.3, the vectorized computation is about three orders of magnitude faster than the naïve algorithm. A small drawback of this solution is that it computes all candidate output spike responses $\widetilde{\mathbf{g}}$, even when it is evident that the neuron will not generate a spike in response to the event, or when only the first few events are actually needed to elicit such response. In such scenarios the iterative search might be preferable. While this leaves room for further improvement, we note that according to our simulations, this algorithm already significantly outperforms the naïve implementation (Section 3.2.1.3).

So far we have shown how the causal set search can be realized in a single pass over an event sequence. Next, this computation must be repeated for every output neuron and every input example pairwise combination. Sections 3.2.1.2.2-3.2.1.2.3 present one possible approach to computing the causal set (and the output spike time) for all of these pairs in parallel. This description utilizes third-order tensors, making it compatible with modern deep learning framework. Furthermore, it also highlight edge-case scenarios and presents solutions to additional challenges that arise when dealing with third-order tensors of fixed size (i.e., due to the variable number of events in each example of the minibatch).

Algorithm 3.2: Vectorized causal set search for a single output neuron observing an event sequence.

```
Input: $\widetilde{\mathbf{z}}$ # vector of sorted input events of length $C$
Input: $\widetilde{\mathbf{w}}$ # vector of synaptic weights associated with elements in $\widetilde{\mathbf{z}}$
Output: $\widetilde{Q}$ # causal neuron set
Output: $z_{\text{out}}$ # transformed output spike time
Begin:
        $\widetilde{\mathbf{m}} \leftarrow \text{cumsum}(\widetilde{\mathbf{w}} \odot \widetilde{\mathbf{z}})$
        $\widetilde{\mathbf{d}} \leftarrow \text{cumsum}(\widetilde{\mathbf{w}}) - \frac{V_{\text{thr}}}{\tau_{\text{syn}}}$
        $\widetilde{\mathbf{g}} \leftarrow \widetilde{\mathbf{m}} \oslash \widetilde{\mathbf{d}}$
        $\widetilde{\mathbf{u}} \leftarrow \text{next}(\widetilde{\mathbf{z}})$ # according to (3.21)
        $\widetilde{\Omega} \leftarrow \{c : \widetilde{d}_c > 0 \ \wedge \ \widetilde{g}_c > 0 \ \wedge \ \widetilde{z}_c < \widetilde{g}_c < \widetilde{u}_c\}$
        If $\widetilde{\Omega} = \varnothing$:
                $\widetilde{Q} \leftarrow \varnothing$
                $z_{\text{out}} \leftarrow \infty$
        Else:
                $c^* \leftarrow \min \widetilde{\Omega}$
                $\widetilde{Q} \leftarrow \{c : c \leq c^*\}$
                $z_{\text{out}} \leftarrow \widetilde{g}_{c^*}$
        End If
        Return $\widetilde{Q}, z_{\text{out}}$
End
```

3.2.1.2 Vectorized algorithm details – computation of the output spike times for a batch of event sequences

3.2.1.2.1 Preliminaries

A *tensor* is a multidimensional array. The *order* of a tensor denotes the number of indices needed to address a single element of a tensor. For example,

$$\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3} \tag{3.24}$$

is a third-order tensor in $\mathbb{R}$. An n-th order tensor has *n modes*. The (i_1, i_2, i_3)-th element of the tensor $\mathcal{A}$ is addressed as $(\mathcal{A})_{i_1 i_2 i_3}$ or $a_{i_1 i_2 i_3}$ for indices $i_1 \in \{1, \ldots, I_1\}$, $i_2 \in \{1, \ldots, I_2\}$ and $i_3 \in \{1, \ldots, I_3\}$. Tensors are generalizations of scalars (zeroth-order tensors), vectors (first-order tensors) and matrices (second-order tensors) to an arbitrary number of modes. Vectors are written as boldface lowercase letters (e.g. $\mathbf{a}$), matrices are boldface capital letters (e.g. $\mathbf{A}$) and tensors are Euler script capital letters (e.g. $\mathcal{A}$).

Tensor *fibers* are one-dimensional sections of a tensor formed when all but one index are fixed. A colon is used to indicate all elements of a mode. Similarly, a tensor *slice* is a two-dimensional section with all but two indices fixed. For example, a third-order tensor $\mathcal{A}$ has

- column ($\mathbf{a}_{:jk}$), row ($\mathbf{a}_{i:k}$) and tube fibers ($\mathbf{a}_{ij:}$), also knows as mode-1, mode-2 and mode-3 fibers, respectively;
- horizontal ($\mathbf{A}_{i::}$), lateral ($\mathbf{A}_{:j:}$) and frontal slices ($\mathbf{A}_{::k}$).

Note that the notation indicates that fibers are vectors whereas slices are matrices.

In addition to the main indices denoted by its order, each tensor can have arbitrarily many *singular indices*. A classical example is the row vector $\mathbf{a} \in \mathbb{R}^{1 \times J}$ and its transpose, the column vector $\mathbf{a}^T$. Note that according to the established convention, row and column vectors are denoted by boldface lowercase letters as if they were actual vectors (i.e., without the singular index). Singular indices are redundant and need not be explicitly specified when addressing tensors elements. For example, the matrix $\mathbf{A} \in \mathbb{R}^{J_1 \times 1 \times J_2}$ has elements

$$\mathbf{A} = \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1J_2} \\ z_{21} & z_{22} & \cdots & z_{2J_2} \\ \vdots & \vdots & \ddots & \vdots \\ z_{J_11} & z_{J_12} & \cdots & z_{J_1J_2} \end{bmatrix}. \tag{3.25}$$

For consistency, we denote third-order tensors with a singular dimension by boldface capital letters, which is the same notation used for matrices.

Tensor *matricization* refers to a specific mapping of tensor's elements such that they are arranged in a matrix. The *n-mode matricization* arranges mode-n fibers as columns of a matrix.

To give a concrete example, let $\mathcal{A} \in \mathbb{Z}^{4\times3\times2}$ be a tensor with frontal slices

$$\mathbf{A}_{::1} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}, \quad \mathbf{A}_{::2} = \begin{bmatrix} -1 & -2 & -3 & -4 \\ -5 & -6 & -7 & -8 \\ -9 & -10 & -11 & -12 \end{bmatrix}. \tag{3.26}$$

Then,

$$\begin{aligned}
\mathbf{A}_{(1)} &= \begin{bmatrix} 1 & 2 & 3 & 4 & -1 & -2 & -3 & -4 \\ 5 & 6 & 7 & 8 & -5 & -6 & -7 & -8 \\ 9 & 10 & 11 & 12 & -9 & -10 & -11 & -12 \end{bmatrix}, \\
\mathbf{A}_{(2)} &= \begin{bmatrix} 1 & 5 & 9 & -1 & -5 & -9 \\ 2 & 6 & 10 & -2 & -6 & -10 \\ 3 & 7 & 11 & -3 & -7 & -11 \\ 4 & 8 & 12 & -4 & -8 & -12 \end{bmatrix}, \\
\mathbf{A}_{(3)} &= \begin{bmatrix} 1 & 5 & 9 & 2 & 6 & 10 & 3 & 7 & 11 & 4 & 8 & 12 \\ -1 & -5 & -9 & -2 & -6 & -10 & -3 & -7 & -11 & -4 & -8 & -12 \end{bmatrix}
\end{aligned} \tag{3.27}$$

are mode-1, mode-2 and mode-3 matricizations of $\mathcal{A}$, respectively. The number of rows in the resulting matrix is equal to dimensionality of the mode-n fiber. Again, the notation indicates that the matricization result is a matrix.

The *n-mode product* between tensor $\mathcal{A} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_n\times\cdots\times I_N}$ and matrix $\mathbf{M} \in \mathbb{R}^{J\times I_n}$ is defined as

$$\mathcal{B} = \mathcal{A} \times_n \mathbf{M} \tag{3.28}$$

where $\mathcal{B} \in \mathbb{R}^{I_1\times\cdots\times I_{n-1}\times J\times I_{n+1}\times\cdots\times I_N}$ has elements

$$b_{i_1\cdots i_{n-1} j i_{n+1}\cdots i_N} = \sum_{i_n=1}^{I_n} a_{i_1\cdots i_n\cdots i_N} m_{j i_n}\,. \tag{3.29}$$

Furthermore, let $\mathbf{1}_J \in \{1\}^{1\times J}$ be a (row) *vector of ones*. Then, we can formally define *n-mode broadcasting* of a tensor with a singular n-th index $\mathcal{A} \in \mathbb{R}^{I_1\times\cdots\times I_{n-1}\times1\times I_{n+1}\times\cdots\times I_N}$ as a function

$$\mathrm{bcast}_n\left(\mathcal{A}; J\right) = \mathcal{A} \times_n \mathbf{1}_J^T \tag{3.30}$$

with the resulting elements

$$\left(\mathrm{bcast}_n\left(\mathcal{A}; J\right)\right)_{i_1\cdots i_{n-1} j i_{n+1}\cdots i_N} = a_{i_1\cdots i_{n-1} i_{n+1}\cdots i_N}\,. \tag{3.31}$$

Broadcasting repeats tensor elements J-times across what used to be the singular n-th index. The concept of (array) broadcasting is used extensively in scientific computation and deep learning frameworks [160]. Similarly, let us define *n-mode cumulative sum* as a function acting on tensor $\mathcal{B} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times \cdots \times I_N}$ such that

$$\mathcal{C} = \mathrm{cumsum}_n\left(\mathcal{B}\right) \tag{3.32}$$

has elements

$$c_{i_1 \cdots i_n \cdots i_N} = \sum_{j=1}^{i_n} b_{i_1 \cdots j \cdots i_N} \,. \tag{3.33}$$

Finally, let us introduce several element-wise operators. Let $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, $\mathcal{B} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, $\mathcal{C} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\mathcal{D} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ be tensors with the same dimensionality. The *Hadamard product* between tensors $\mathcal{A}$ and $\mathcal{B}$ is

$$\mathcal{C} = \mathcal{A} \odot \mathcal{B} \,, \tag{3.34}$$

with elements

$$c_{i_1 \cdots i_N} = a_{i_1 \cdots i_N} b_{i_1 \cdots i_N} \,. \tag{3.35}$$

Similarly, the *Hadamard division* is

$$\mathcal{D} = \mathcal{A} \oslash \mathcal{B} \tag{3.36}$$

with elements

$$d_{i_1 \cdots i_N} = \frac{a_{i_1 \cdots i_N}}{b_{i_1 \cdots i_N}} \,. \tag{3.37}$$

3.2.1.2.2 Forward pass

Let us now describe the spiking dense layer computation, noting that the following description is valid for any layer in the network and so to simplify the notation we avoid using a dedicated layer index. Let $\mathbf{Z} \in [1, \infty]^{N \times 1 \times C}$ be a matrix with elements

$$\mathbf{Z} = \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1C} \\ z_{21} & z_{22} & \cdots & z_{2C} \\ \vdots & \vdots & \ddots & \vdots \\ z_{N1} & z_{N2} & \cdots & z_{NC} \end{bmatrix} \tag{3.38}$$

where the index n runs over the number of examples in a batch N, and the index c runs over the number of input neurons in the layer C. The element z_{nc} represents the transformed spike time (3.6) of the c-th neuron for the n-th input spike train. In principle, spike train sequences selected to be a part of the same batch need not have the same number of events, however they do need to have the same number of elements in order to construct matrix $\mathbf{Z}$. To account for that, we pad them with the non-event indicator ($z_{nc} = \infty$) along index c. Similarly, the matrix $\mathbf{W} \in \mathbb{R}^{1\times P\times C}$ with elements

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1C} \\ w_{21} & w_{22} & \dots & w_{2C} \\ \vdots & \vdots & \ddots & \vdots \\ w_{P1} & w_{P2} & \dots & w_{PC} \end{bmatrix} \tag{3.39}$$

describes the synaptic weights between the c-th input neuron and the p-th output neuron.

Let $\mathbf{S} \in \mathbb{Z}_+^{N\times 1\times C}$ be a matrix of indices such that the mapping

$$\widetilde{\mathbf{Z}} = s\left(\mathbf{Z}; \mathbf{S}\right) \tag{3.40}$$

reorders the elements of matrix $\mathbf{Z}$ according to $\mathbf{S}$ resulting in a matrix $\widetilde{\mathbf{Z}}$ with all of its elements sorted in an ascending order along the index c. The mapping produced by (3.40) can be inverted

$$\mathbf{Z} = s^{-1}\left(\widetilde{\mathbf{Z}}; \mathbf{S}\right) \tag{3.41}$$

to produce the original matrix $\mathbf{Z}$. Importantly, the function (3.40) (and its inverse) can be applied to any matrix $\mathbf{A}$ that has the same dimensionality as $\mathbf{S}$ in order to construct a matrix $\widetilde{\mathbf{A}}$ with its elements reordered according to $\mathbf{S}$. The inverse mapping function will be used in the backward pass through the layer. Throughout the rest of this Section the tilde character above matrices (tensors) denotes the aforementioned sorting index reference.

Additionally, let $\widetilde{\mathbf{U}} \in (\{-\infty\} \cup [1, \infty])^{N\times 1\times C}$ be a matrix denoting the time of the next input spike relative to $\widetilde{\mathbf{Z}}$ with elements

$$\widetilde{u}_{nc} = \begin{cases} \infty & \text{if} \quad c = C \quad \wedge \quad \widetilde{z}_{nc} < \infty \\ \widetilde{z}_{n(c+1)} & \text{if} \quad c < C \quad \wedge \quad \widetilde{z}_{n(c+1)} > \widetilde{z}_{nc} \\ -\infty & \text{otherwise} \end{cases} . \tag{3.42}$$

This definition captures the following scenarios:

- $\widetilde{u}_{nc} = \infty$ implies that $\widetilde{z}_{nc}$ is the last element of the spike sequence,
- $\widetilde{u}_{nc} = -\infty$ means that two consecutive events $\widetilde{z}_{nc}, \widetilde{z}_{n(c+1)}$ occur at the same time, or that both of them are non-event indicators ($\widetilde{z}_{nc} = \infty$). The latter notion is also valid for $c = C$, in which case $\widetilde{z}_{n(C+1)} = \infty$ is implied.

Note that each row of $\widetilde{\mathbf{U}}$ must contain exactly one non-event indicator, unless the corresponding "sequence" is completely empty.

According to the naïve version of the algorithm, we need to pair each input event $\{z_{nc}\}$ with a set of synaptic weights $\{w_{pc}\}$ connected to that input neuron. To do so, recall that matrices

$\mathbf{Z} \in [1, \infty]^{N \times 1 \times C}$ and $\mathbf{W} \in \mathbb{R}^{1 \times P \times C}$ have singular indices. We can apply broadcasting to produce third-order tensors $\mathcal{Z} \in [1, \infty]^{N \times P \times C}$ and $\mathcal{W} \in \mathbb{R}^{N \times P \times C}$ such that

$$\mathcal{Z} = \text{bcast}_2 (\mathbf{Z}; P) \tag{3.43}$$

and

$$\mathcal{W} = \text{bcast}_1 (\mathbf{W}; N) \ . \tag{3.44}$$

Repeating elements across the singular dimensions has a fairly straightforward interpretation. For tensor $\mathcal{Z}$ the spike train sequence observed by input neurons is obviously the same, regardless of the choice of the output neuron p. Similarly for tensor $\mathcal{W}$ – the synaptic weights do not depend on the input example n. For completeness, we can broadcast the matrix $\mathbf{S}$ to

$$\mathcal{S} = \text{bcast}_2 (\mathbf{S}; P) \tag{3.45}$$

such that the mapping

$$\widetilde{\mathcal{Z}} = s (\mathcal{Z}; \mathcal{S}) \tag{3.46}$$

reorders that elements of $\mathcal{Z}$ so that they are sorted along index c, and that the inverse mapping

$$\mathcal{Z} = s^{-1} \left(\widetilde{\mathcal{Z}}; \mathcal{S} \right) \tag{3.47}$$

exists. Finally, denote

$$\widetilde{\mathcal{U}} = \text{bcast}_2 \left(\widetilde{\mathbf{U}}; P \right) \ . \tag{3.48}$$

Having computed the sorted spike-times tensor $\widetilde{\mathcal{Z}}$ and the weights tensor projected according to the sorting indices $\widetilde{\mathcal{W}}$, the next step is to find all possible candidates for the output spikes, which for a single output neuron is given in an implicit form (3.5) conditioned on the causal set Q. First, let us denote the cumulative sum of $\widetilde{\mathcal{W}}$ along the index c as tensor $\widetilde{\mathcal{L}} \in \mathbb{R}^{N \times P \times C}$

$$\widetilde{\mathcal{L}} = \text{cumsum}_3 \left(\widetilde{\mathcal{W}} \right) \tag{3.49}$$

with elements $\widetilde{\ell}_{npc}$. Similarly, the membrane current tensor $\widetilde{\mathcal{M}} \in \mathbb{R}^{N \times P \times C}$ represents the cumulative sum along index c of the Hadamard product $\widetilde{\mathcal{W}} \odot \widetilde{\mathcal{Z}}$. Let us explicitly define the elements of $\widetilde{\mathcal{M}}$ as

$$\widetilde{m}_{npc} = \begin{cases} \sum_{i=1}^{c} \widetilde{w}_{npi} \widetilde{z}_{npi} & \text{if} \quad \widetilde{z}_{npc} < \infty \\ \text{sgn} \left(\widetilde{\ell}_{npc} \right) \cdot \infty & \text{otherwise} \end{cases} , \tag{3.50}$$

where $\text{sgn}(x)$ is the signum function. The second case is a heuristic that prevents obtaining an undefined result $(\infty - \infty)$ in the cumulative sum when $\widetilde{z}_{npc} = \infty$. Then, the output spike candidate denominator tensor $\widetilde{\mathcal{D}} \in \mathbb{R}^{N \times P \times C}$ is

$$\widetilde{\mathcal{D}} = \widetilde{\mathcal{L}} - \frac{V_{\text{thr}}}{\tau_{\text{syn}}} , \tag{3.51}$$

whereas the spike candidate tensor $\widetilde{\mathcal{G}} \in \mathbb{R}^{N \times P \times C}$ is constructed as

$$\widetilde{\mathcal{G}} = \widetilde{\mathcal{M}} \oslash \widetilde{\mathcal{D}} . \tag{3.52}$$

The next step aims to determine which elements of the candidate output events tensor $\widetilde{\mathcal{G}}$ are valid, i.e., find all input spike combinations that could results in an output spike generation. The set of index triplets $\{n, p, c\}$ that denote the valid elements of $\widetilde{\mathcal{G}}$ is

$$\widetilde{\Omega} = \{n, p, c : \ \widetilde{d}_{npc} > 0 \ \wedge \ \widetilde{g}_{npc} > 0 \ \wedge \ \widetilde{z}_{npc} < \widetilde{g}_{npc} < \widetilde{u}_{npc}\} . \tag{3.53}$$

The three conditions enforce that

1) the sum of weights is larger than the scaled voltage threshold (3.7),
2) the output spike candidate has a valid domain so that the inverse transform of (3.6) mapping $z \longrightarrow t$ exists, and
3) the spike causality principle holds, i.e., the candidate spike $\widetilde{g}_{npc}$ occurs later than the most recent input spike $\widetilde{z}_{npc}$, but earlier than the next input spike $\widetilde{u}_{npc}$,

respectively. The set of valid indices $\widetilde{\Omega}$ is analogous to the causal set Q from (3.4).

Note that neither $\widetilde{\Omega}$ nor Q say anything about the earliest valid spike candidate (although in the naïve implementation this is an implicit result of the *for*-loop operation). To determine the earliest possible output spike, we first construct a matrix $\widetilde{\mathbf{Q}} \in \mathbb{Z}_{0+}^{N \times P \times 1}$ with elements

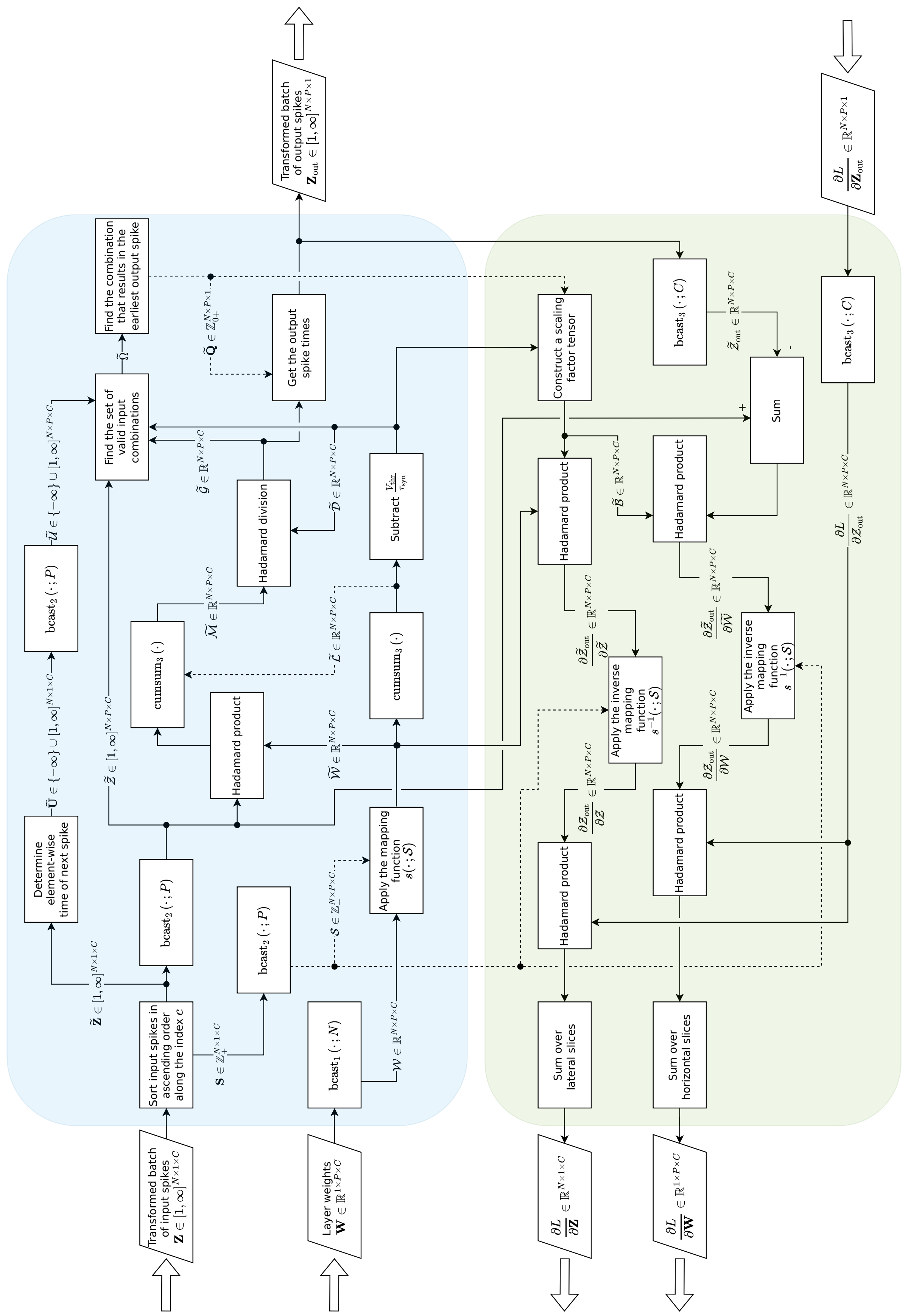
$$\widetilde{q}_{np} = \begin{cases} 0 & \text{if} \quad \{k : \{n, p, k\} \subset \widetilde{\Omega}\} = \emptyset \\ \min\{k : \{n, p, k\} \subset \widetilde{\Omega}\} & \text{otherwise} \end{cases} . \tag{3.54}$$

The matrix $\widetilde{\mathbf{Q}}$ can be thought of as an indicator of the actual (generated) output spike according to all possible valid input spike combinations, which is why it is a matrix and not a third-order tensor. Then, the output spike matrix $\mathbf{Z}_{\text{out}} \in [1, \infty]^{N \times P \times 1}$ has elements

$$z_{\text{out}_{np}} = \begin{cases} \widetilde{g}_{npk} \quad \text{where} \quad k = \widetilde{q}_{np} & \text{if} \quad \widetilde{q}_{np} > 0 \\ \infty & \text{otherwise} \end{cases} . \tag{3.55}$$

This step concludes the forward pass through the layer. Note that $\mathbf{Z}_{\text{out}}$ need not be described in terms of the sorting index reference $\mathbf{S}$ because the index corresponding to the input dimension is singular.

The algorithm is summarized in Figure 3.7a, which additionally shows that some of the computed values can be reused in the backward pass. In contrast to the flowchart for the naïve algorithm presented in Figure 3.6a, there are no loops, which significantly reduces processing time.



Transformed batch of input spikes $\mathbf{Z} \in [1,\infty]^{N\times 1\times C}$
Layer weights $\mathbf{W} \in \mathbb{R}^{1\times P\times C}$
Sort input spikes in ascending order along the index c
$\tilde{\mathbf{z}} \in [1,\infty]^{N\times 1\times C}$
$\mathbf{S} \in \mathbb{Z}_+^{N\times 1\times C}$
$\text{bcast}_1(\cdot\,;N)$
$\mathcal{W} \in \mathbb{R}^{N\times P\times C}$
$\text{bcast}_2(\cdot\,;P)$
$\mathcal{S} \in \mathbb{Z}_+^{N\times P\times C}$
Apply the mapping function $s(\cdot\,;\mathcal{S})$
Determine element-wise time of next spike
$\tilde{\mathbf{U}} \in \{-\infty\}\cup[1,\infty]^{N\times 1\times C}$
$\tilde{\mathcal{Z}} \in [1,\infty]^{N\times P\times C}$
Hadamard product
$\widetilde{\mathcal{W}} \in \mathbb{R}^{N\times P\times C}$
$\text{cumsum}_3(\cdot)$
$\tilde{\mathcal{L}} \in \mathbb{R}^{N\times P\times C}$
$\widetilde{\mathcal{M}} \in \mathbb{R}^{N\times P\times C}$
Hadamard division
$\tilde{\mathcal{D}} \in \mathbb{R}^{N\times P\times C}$
Subtract $\frac{V_{\text{thr}}}{\tau_{\text{syn}}}$
$\tilde{\mathcal{U}} \in \{-\infty\}\cup[1,\infty]^{N\times P\times C}$
$\tilde{\mathcal{G}} \in \mathbb{R}^{N\times P\times C}$
Find the set of valid input combinations
$\tilde{\Omega}$
Find the combination that results in the earliest output spike
$\tilde{\mathbf{Q}} \in \mathbb{Z}_{0+}^{N\times P\times 1}$
Get the output spike times
Transformed batch of output spikes $\mathbf{Z}_{\text{out}} \in [1,\infty]^{N\times P\times 1}$
$\frac{\partial L}{\partial \mathbf{Z}_{\text{out}}} \in \mathbb{R}^{N\times P\times 1}$
$\text{bcast}_3(\cdot\,;C)$
$\tilde{\mathcal{Z}}_{\text{out}} \in \mathbb{R}^{N\times P\times C}$
Construct a scaling factor tensor
Sum
+
-
$\tilde{\mathcal{B}} \in \mathbb{R}^{N\times P\times C}$
$\frac{\partial L}{\partial \mathcal{Z}_{\text{out}}} \in \mathbb{R}^{N\times P\times C}$
$\frac{\partial \tilde{\mathcal{Z}}_{\text{out}}}{\partial \tilde{\mathcal{Z}}} \in \mathbb{R}^{N\times P\times C}$
$\frac{\partial \tilde{\mathcal{Z}}_{\text{out}}}{\partial \widetilde{\mathcal{W}}} \in \mathbb{R}^{N\times P\times C}$
Apply the inverse mapping function $s^{-1}(\cdot\,;\mathcal{S})$
$\frac{\partial \mathcal{Z}_{\text{out}}}{\partial \mathcal{Z}} \in \mathbb{R}^{N\times P\times C}$
$\frac{\partial \mathcal{Z}_{\text{out}}}{\partial \mathcal{W}} \in \mathbb{R}^{N\times P\times C}$
Sum over lateral slices
Sum over horizontal slices
$\frac{\partial L}{\partial \mathbf{Z}} \in \mathbb{R}^{N\times 1\times C}$
$\frac{\partial L}{\partial \mathbf{W}} \in \mathbb{R}^{1\times P\times C}$

Figure 3.7: Vectorized signal propagation through the spiking dense layer. Dashed arrows pointing towards operator blocks denote information that changes the output element-wise or influences the order of elements. Top: forward pass. Bottom: backward pass.

3.2.1.2.3 Backward pass

The purpose of the backward pass through the layer is to compute gradient of the loss function L (such as the cross-entropy loss (3.10)) with respect to the layer weights $\mathbf{W}$ and the input signal $\mathbf{Z}$. The former is used to update the weights prior to the next optimization step, whereas the latter flows to the preceding layer. With a slight abuse of notation, let $\frac{\partial L}{\partial \mathbf{Z}_{\text{out}}} \in \mathbb{R}^{N \times P \times 1}$ be a matrix with elements

$$\frac{\partial L}{\partial \mathbf{Z}_{\text{out}}} = \begin{bmatrix} \frac{\partial L}{\partial z_{\text{out}_{11}}} & \frac{\partial L}{\partial z_{\text{out}_{12}}} & \cdots & \frac{\partial L}{\partial z_{\text{out}_{1P}}} \\ \frac{\partial L}{\partial z_{\text{out}_{21}}} & \frac{\partial L}{\partial z_{\text{out}_{22}}} & \cdots & \frac{\partial L}{\partial z_{\text{out}_{2P}}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial z_{\text{out}_{N1}}} & \frac{\partial L}{\partial z_{\text{out}_{N2}}} & \cdots & \frac{\partial L}{\partial z_{\text{out}_{NP}}} \end{bmatrix} . \tag{3.56}$$

This is the signal that flows into the backward pass from the next layer. We wish to compute the partial derivatives $\frac{\partial L}{\partial \mathbf{Z}} \in \mathbb{R}^{N \times 1 \times C}$ and $\frac{\partial L}{\partial \mathbf{W}} \in \mathbb{R}^{1 \times P \times C}$. From the chain rule of derivatives we have

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{Z}} &= \sum_{p=1}^{P} \left(\frac{\partial L}{\partial \mathcal{Z}_{\text{out}}} \odot \frac{\partial \mathcal{Z}_{\text{out}}}{\partial \mathcal{Z}} \right)_{:p:} \\ \frac{\partial L}{\partial \mathbf{W}} &= \sum_{n=1}^{N} \left(\frac{\partial L}{\partial \mathcal{Z}_{\text{out}}} \odot \frac{\partial \mathcal{Z}_{\text{out}}}{\partial \mathcal{W}} \right)_{n::} \end{aligned} . \tag{3.57}$$

This shows that the partial derivatives can be computed from third-order tensors by summing them across the appropriate slices. The partial derivative $\frac{\partial L}{\partial \mathcal{Z}_{\text{out}}} \in \mathbb{R}^{N \times P \times C}$ can be obtained by broadcasting the backward pass input signal (3.56)

$$\frac{\partial L}{\partial \mathcal{Z}_{\text{out}}} = \text{bcast}_3 \left(\frac{\partial L}{\partial \mathbf{Z}_{\text{out}}}; C \right) . \tag{3.58}$$

All that is left to do is compute $\frac{\partial \mathcal{Z}_{\text{out}}}{\partial \mathcal{Z}} \in \mathbb{R}^{N \times P \times C}$ and $\frac{\partial \mathcal{Z}_{\text{out}}}{\partial \mathcal{W}} \in \mathbb{R}^{N \times P \times C}$.

Recall from the partial derivatives formulae derived for a single IF neuron (3.8)-(3.9) that they are both set to zero for neurons which were not a part of the causal set used to produce the output spike in the forward pass. Additionally, both equations share a common denominator term, which is also present in (3.5). This means that some of the results obtained during the forward pass computation described in Section 3.2.1.2.2 can be reused. Therefore, let us first introduce the scaling factor tensor $\widetilde{\mathcal{B}} \in \mathbb{R}^{N \times P \times C}$ which can be constructed from the spike candidate denominator tensor $\widetilde{\mathcal{D}}$ (3.51) according to the elements of the output spike indicator matrix $\widetilde{\mathbf{Q}}$ (3.54)

$$\widetilde{b}_{npc} = \begin{cases} \left(\widetilde{d}_{npk} \right)^{-1} & \text{where} \quad k = \widetilde{q}_{np} \quad \text{if} \quad c \leq k \quad \wedge \quad \widetilde{q}_{np} > 0 \\ 0 & \text{otherwise} \end{cases} . \tag{3.59}$$

Note that the element $\left(\widetilde{d}_{npk}\right)^{-1}$ which satisfies the aforementioned condition is guaranteed to be positive (3.7). Furthermore, this condition ensures that all nonzero elements of $\widetilde{\mathcal{B}}$ are associated with a combination of inputs that are a part of a causal set.

In order to compute the partial derivatives with respect to input spikes and synaptic weights we first need to ensure that $\mathbf{Z}_{\text{out}}$ is of the same dimensionality as tensors $\widetilde{\mathcal{B}}$, $\widetilde{\mathcal{Z}}$ and $\widetilde{\mathcal{W}}$. Therefore, we broadcast the matrix to produce a tensor $\widetilde{\mathcal{Z}}_{\text{out}} \in \mathbb{R}^{N \times P \times C}$

$$\widetilde{\mathcal{Z}}_{\text{out}} = \text{bcast}_3\left(\mathbf{Z}_{\text{out}}; C\right) . \tag{3.60}$$

Note that $\mathcal{Z}_{\text{out}} \triangleq \widetilde{\mathcal{Z}}_{\text{out}}$. Then, the partial derivative of $\widetilde{\mathcal{Z}}_{\text{out}}$ with respect to $\mathcal{S}$-sorted input neurons

$$\frac{\partial \widetilde{\mathcal{Z}}_{\text{out}}}{\partial \widetilde{\mathcal{Z}}} = \widetilde{\mathcal{B}} \odot \widetilde{\mathcal{W}} , \tag{3.61}$$

whereas the partial derivative of $\widetilde{\mathcal{Z}}_{\text{out}}$ with respect to $\mathcal{S}$-sorted weights is

$$\frac{\partial \widetilde{\mathcal{Z}}_{\text{out}}}{\partial \widetilde{\mathcal{W}}} = \begin{cases} \widetilde{\mathcal{B}} \odot \left(\widetilde{\mathcal{Z}} - \widetilde{\mathcal{Z}}_{\text{out}}\right) & \text{if} \quad \widetilde{b}_{npc} > 0 \\ 0 & \text{otherwise} \end{cases} . \tag{3.62}$$

This system of equations is designed to prevent undefined elements which might occur when

$$\widetilde{b}_{npc} = 0 \quad \wedge \quad \left(\widetilde{z}_{npc} = \pm\infty \quad \vee \quad \widetilde{z}_{\text{out}_{npc}} = \pm\infty\right)$$

or

$$\widetilde{z}_{npc} = \pm\infty \quad \wedge \quad \widetilde{z}_{\text{out}_{npc}} = \pm\infty .$$

Then, the sought-after partial derivatives $\frac{\partial \mathcal{Z}_{\text{out}}}{\partial \mathcal{Z}}$ and $\frac{\partial \mathcal{Z}_{\text{out}}}{\partial \mathcal{W}}$ can be obtained by applying the inverse mapping function (3.47)

$$\begin{aligned} \frac{\partial \mathcal{Z}_{\text{out}}}{\partial \mathcal{Z}} &= s^{-1}\left(\frac{\partial \widetilde{\mathcal{Z}}_{\text{out}}}{\partial \widetilde{\mathcal{Z}}}; \mathcal{S}\right) \\ \frac{\partial \mathcal{Z}_{\text{out}}}{\partial \mathcal{W}} &= s^{-1}\left(\frac{\partial \widetilde{\mathcal{Z}}_{\text{out}}}{\partial \widetilde{\mathcal{W}}}; \mathcal{S}\right) \end{aligned} . \tag{3.63}$$

Plugging (3.58) and (3.63) into (3.57) concludes the backward pass. The algorithm for the backward pass is summarized in Figure 3.7b.

3.2.1.3 Comparison with the naïve algorithm

In order to assess the feasibility of training networks with the vectorized algorithm introduced in the previous sections, its time-performance was compared with the naïve version. Both of them were implemented in the TensorFlow [158] deep learning framework.

Simulation 5. The evaluation protocol can be summarized as follows:

1) For every combination of values of the {*input_shape*}, {*output shape*} and {*batch size*}:
   a) For every value of the {*random_seed*}:
      i. Construct the *naïve* and *vectorized layers* with {*input_shape*} input neurons and {*output_shape*} output neurons, with weights sampled according to the {*random_seed*}.
      ii. For every {*sample_index*}:
         A. Construct {*batch_size*} examples by sampling a spike $x \sim \mathrm{U}(0, 5)$ for every input neuron according to the {*random_seed*}.
         B. For both layer types: measure the time it takes to process sampled examples in the forward and backward pass, averaged over {*num_reps*} repetitions.

It must be stressed that in the aforementioned algorithm evaluation protocol each input neuron is associated with a single spike. This means that the naïve implementation cannot take advantage of early stopping of the innermost *for*-loop when no more spikes occur. Conversely, the vectorized implementation is, in general, unable to distinguish between inputs with and without spikes.[e] Therefore, for a more fair comparison between the two implementations, input signal sparsity was not considered in the analysis. Furthermore, the evaluation protocol has no optimization objective and each set of examples is passed through the layer {*num_reps*} times without updating the weights. Thus, backpropagation computation time comparison is not included in this analysis.

The set of parameters used to compare the vectorized and naïve algorithms is summarized in Table 3.2. The values of the {*input_shape*}, {*output_shape*} and {*batch_size*} were chosen on a $\log_2$-scale in order to capture the range of values typically used to train nonspiking dense networks. Repeating the simulations with a different random seed used to sample the set of weights averages the results over different initial states of the network, which might differ in how easy it is to propagate the signal through the layer. Repeated sampling of input examples further averages the results over different signals observed by the layer, although, given that all input neurons produce a spike, this primarily has an effect on the spike-sorting component of the algorithm. All simulations were conducted on the same device which was otherwise idle (i.e., no other CPU-intensive tasks were underway) and so repeating each run {*num_reps*} times should, in principle, mitigate the effect of hardware task-scheduling & prioritization on the results.

First, let us present the results on how the two algorithms scale with the overall number of neurons in the network and the number of examples passed as their input. Figure 3.8 presents those results as contour lines of the processing time surfaces in the $\log_2$-$\log_{10}$ space. The

[e]Several special cases for which the level of computation can be reduced are discussed in Section 3.2.4.

Table 3.2: An overview of the parameters used in the algorithm evaluation protocol.

Parameter	**Range of values or a number of**
input_shape output_shape	2, 4, 8, 16, 32, 64, 128, 256
batch_size	4, 8, 16, 32, 64, 128
random_seed (no.) sample_index (no.) num_reps (no.)	10

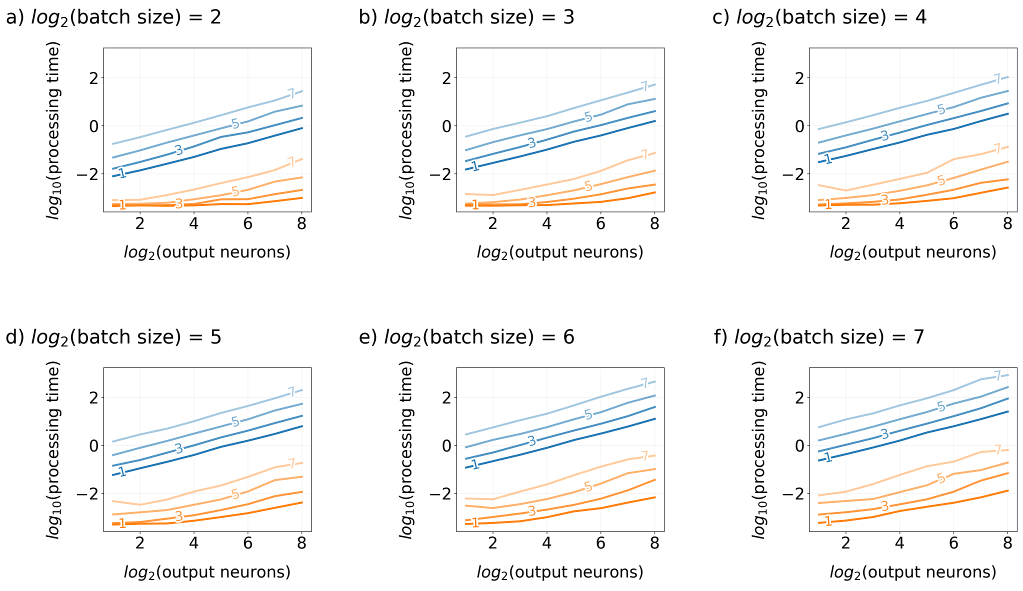


Figure 3.8: Contour lines of the processing time surfaces in the $\log_2$-$\log_{10}$ space for the naïve (in blue) and vectorized (in orange) spiking layer implementations. The results were obtained by varying the batch size and the number of input and output neurons according to the algorithm evaluation protocol in Table 3.2. The number on each curve corresponds to $\log_2$(input neurons). The processing time prior to the log-transform is expressed in seconds.

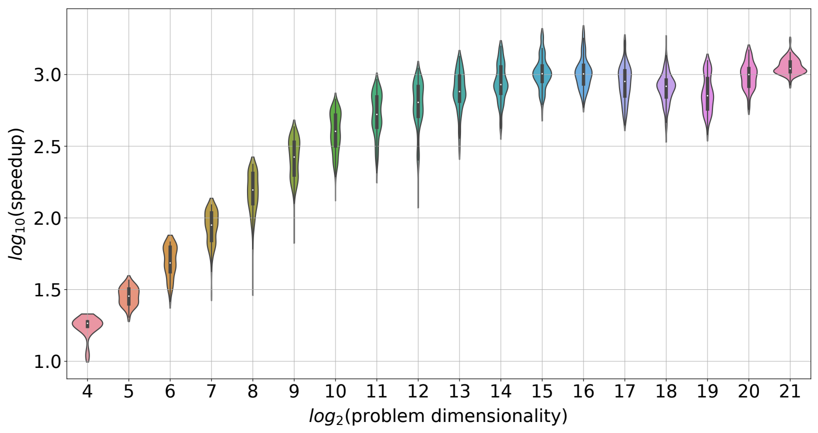


Figure 3.9: Violin plot of the average speedup of the processing time by the vectorized implementation over the naïve approach at different values of the problem dimensionality.

$\log_2$ component naturally corresponds to the range of values chosen in the evaluation protocol (Table 3.2), whereas the $\log_{10}$ component allows us to display the results at different processing time scales. For the naïve implementation we observe that the surfaces described by the contour lines are approximately flat, with the processing time gradually increasing with the number of input and output neurons. Furthermore, surfaces obtained at different batch sizes are roughly parallel to one another. This finding is in line with what one might expect from an algorithm composed of three nested *for*-loops. The result also shows that when both the batch size and the number of neurons is large (although in the broader context of nonspiking neural networks this number of neurons is still relatively small), then this layer takes hundreds of seconds to process a batch of examples. This makes the naïve implementation impractical for training spiking neural networks of even a moderate size. By contrast, the surfaces for the vectorized implementation show some interesting properties:

1) At sufficiently small batch sizes and number of neurons the processing time is approximately constant (at this resolution). This might be explained by the input spike-sorting component of the algorithm having the biggest impact on the processing time in this region.
2) The surfaces become concave and approximately parallel to one another when the number of neurons and batch sizes are both large enough. In this region the overall complexity of the spike-search algorithm dominates, as was the case for all surfaces observed for the naïve implementation.

Note that the vectorized implementation takes significantly less time to process any single batch of examples, regardless of the number of input and output neurons.

It might be interesting to assess how much faster is the proposed algorithm than the baseline at different points of the evaluation protocol grid. To do that, we first introduce the *problem dimensionality*, defined as $\dim = N \cdot P \cdot C$, as the overall number of elements that contribute to the computation through a spiking layer. We aggregate all results obtained for the same value of the problem dimensionality, and then compute a ratio of the processing time for the naïve approach to the vectorized implementation (i.e., speedup), for all results. This analysis is summarized as a violin plot in Figure 3.9. It shows that the vectorized implementation is faster than the naïve version for the entirety of the problem dimensionality grid. Furthermore, at sufficiently large values of the batch size and the number of neurons in the layer, the proposed approach is consistently three orders of magnitude faster than the baseline. This makes it feasible to train the spiking neural network and is a foundation on which all experimental work described in subsequent Sections of this thesis builds upon.

3.2.2 Numerical instability resulting from absolute time event representation

An interesting property of the discussed model is that it has no notion of absolute time. As shown in the XOR task (Figure 3.4), shifting both input spikes in time by a constant t_δ also shifts the model response by the same value. This includes not only spikes generated by the model, but also the integrated membrane voltage, for all neurons in the model. In fact, this property can be trivially derived by shifting all input spike times $\{t_c\}$ by a constant t_δ

$$t_c^\delta = t_c + t_\delta \implies z_c^\delta = \exp\left(\frac{t_c + t_\delta}{\tau_{\text{syn}}}\right) = z_c z_\delta\,, \tag{3.64}$$

which when plugged into (3.5) results in

$$z_{\text{out}}^\delta = \frac{\sum_{c\in Q} w_c\,(z_c z_\delta)}{\sum_{c\in Q} w_c - \frac{V_{\text{thr}}}{\tau_{\text{syn}}}} = z_{\text{out}} z_\delta \implies t_{\text{out}}^\delta = t_{\text{out}} + t_\delta\,, \tag{3.65}$$

with a caveat that $\forall_c\ t_c^\delta \geq 0$. It must be noted however, that the aforementioned absolute-time invariance is only applicable during model inference on a dedicated hardware. In practice the model is simulated using a programming language of choice, therefore there exists an upper limit to the applied shift t_δ before encountering the limits of floating-point data format. Additionally, the presence of the scaling factor z_δ in (3.65) means that $\frac{\partial z_{\text{out}}^\delta}{\partial w_c} = z_\delta \frac{\partial z_{\text{out}}}{\partial w_c}$ (note that $\frac{\partial z_{\text{out}}}{\partial z_c}$ in (3.8) is invariant to the applied time shift). This implies that the model is not time invariant during training as the final set of trained weights will be different.

Overall, this suggests that not accounting for the magnitude of absolute time reference has an adverse effect on the training procedure, and might ultimately lead to a failed model. For instance, recall the simple XOR-SNN introduced in Section 3.1.4.1. Figure 3.10 (top) shows that when the model is trained with progressively larger values of t_δ, its convergence to a stable solution becomes less likely. Note that the numerical instability emerges at relatively

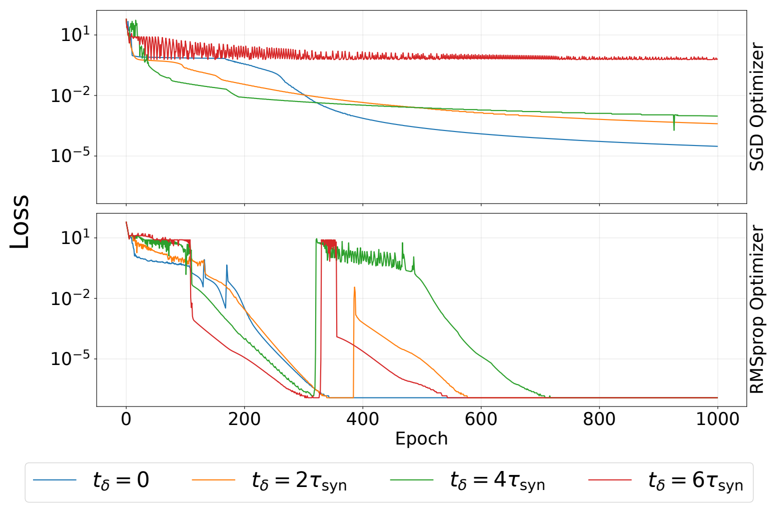


Figure 3.10: Training loss curves of the XOR-SNN model for different values of time-shift t_δ applied to input spikes. Top: model trained using the SGD optimization algorithm. Bottom: model trained using the RMSprop optimization algorithm.

small shifts, making the model impractical for analyzing real data. We found that substituting the SGD optimization algorithm with the RMSprop [161], which focuses on the sign of the gradient instead of its magnitude, helps with convergence, as shown in Figure 3.10 (bottom). Nevertheless, changing the optimization algorithm has an impact only on the learned set of weights and does nothing to help with the magnitude of transformed spikes $\{z_c\}$ becoming too large.

Importantly, in practical applications the input spike sequences are unlikely to all be shifted by the same t_δ, or occur at similar timescales. An example of the latter scenario is the Twitter dataset analyzed in Section 2.5 with sequences composed of events ranging from minutes to weeks since the time reference. It follows that mitigating the absolute time reference problem requires taking into account both of these effects.

Recall from (3.64) and (3.65) that the time shift constant t_δ manifests in the layer's forward-backward computation as a scaling factor. This means that the spike sequence can be shifted in time so that the first spike observed by the layer is at a relative time $t = 0$ (or equivalently $z = 1$) without meaningfully changing the algorithm that finds the output spike time t_{out} (z_{out}). The actual time of the layer output spikes can be obtained by shifting them all in time by the same constant. We shall denote an SNN model as *time-aligned* if all layers of the network observe events starting at relative time $t = 0$. The computation through a time-aligned model is different during training and inference:

- When training the model the information about the absolute time reference is – in general – optional, and so layer outputs are not shifted forward in time. This stems from the fact that during training the ordering of spikes relative to each other and the first input spike is important, not their absolute time. Importantly, if the chosen loss function is not time-invariant (as is the case for the modified cross-entropy loss (3.10)), then its inputs must also be time-aligned. In the backward pass through the model, the derivatives are not scaled by the computed layer time-shifts. This ensures that training the time-aligned SNN results in the same set of layer weights, regardless of the initial time-shift applied to the input signal. Figure 3.11a summarizes signal propagation through a multilayer time-aligned SNN.

- During inference we want to preserve the absolute time reference. Therefore, all spikes are shifted before and after the layer computation. This approach is summarized in Figure 3.11b. An alternative approach would be to process the signal as one would do in the training-mode (i.e., shift only to $t = 0$ without a forward-shift of the resulting output spikes), accumulate the time-shifts across all layers, and restore the absolute time reference only after obtaining the output spikes of the entire model.

Overall, a time-aligned model is time-shift invariant in both training- and inference-time at a negligible computation cost. Note that the computed j-th layer time-shift $t_\delta^{[j]}$ must be finite. If the layer observes no spikes, then $t_\delta^{[j]} \triangleq 0$.

Figure 3.11c presents a comparison between loss curves obtained by training a XOR-SNN model with and without time-alignment versus the time-shift t_δ applied to input spikes. Note that the loss curves for all time-aligned models are identical to one another, meaning that the proposed approach makes the model time-invariant during training. There are three factors that contribute to the observed difference between XOR-SNN and time-aligned XOR-SNN for $t_\delta = 0$. First of all, the time-aligned model shifts spikes that are used in the loss function computation, and so the two models have different losses after only a few iterations. Secondly, the backward pass of the time-aligned SNN ignores the scaling factors induced by layer time-shifts, which has a major impact on the computed partial derivatives. Lastly, recall from Section 3.1.4.1 that the XOR-SNN model is trained with four examples in total, two of which are identical to each other in all but the absolute spike time. Time-aligned SNN removes this input signal time-shift, which reduces the effective number of examples from four to three, making the training dataset no longer class-balanced and more difficult to learn from. This explains why such a simple model converges quite late in the training process.

In order to mitigate the issues with training the model on data occurring at different timescales, we propose to substitute the spike transform function $z(t)$ (3.6) with $z(g(t))$, where the function $g(t)$ is

- monotonically increasing, so that the relative order of events is preserved, and
- bijective on $\mathbb{R}_0^+$, so that there exists an inverse mapping from z to t via $g^{-1}(t)$.

For practical reasons discussed throughout this Section, we additionally require $\forall_{t>0}\ g(t) \leq t$.

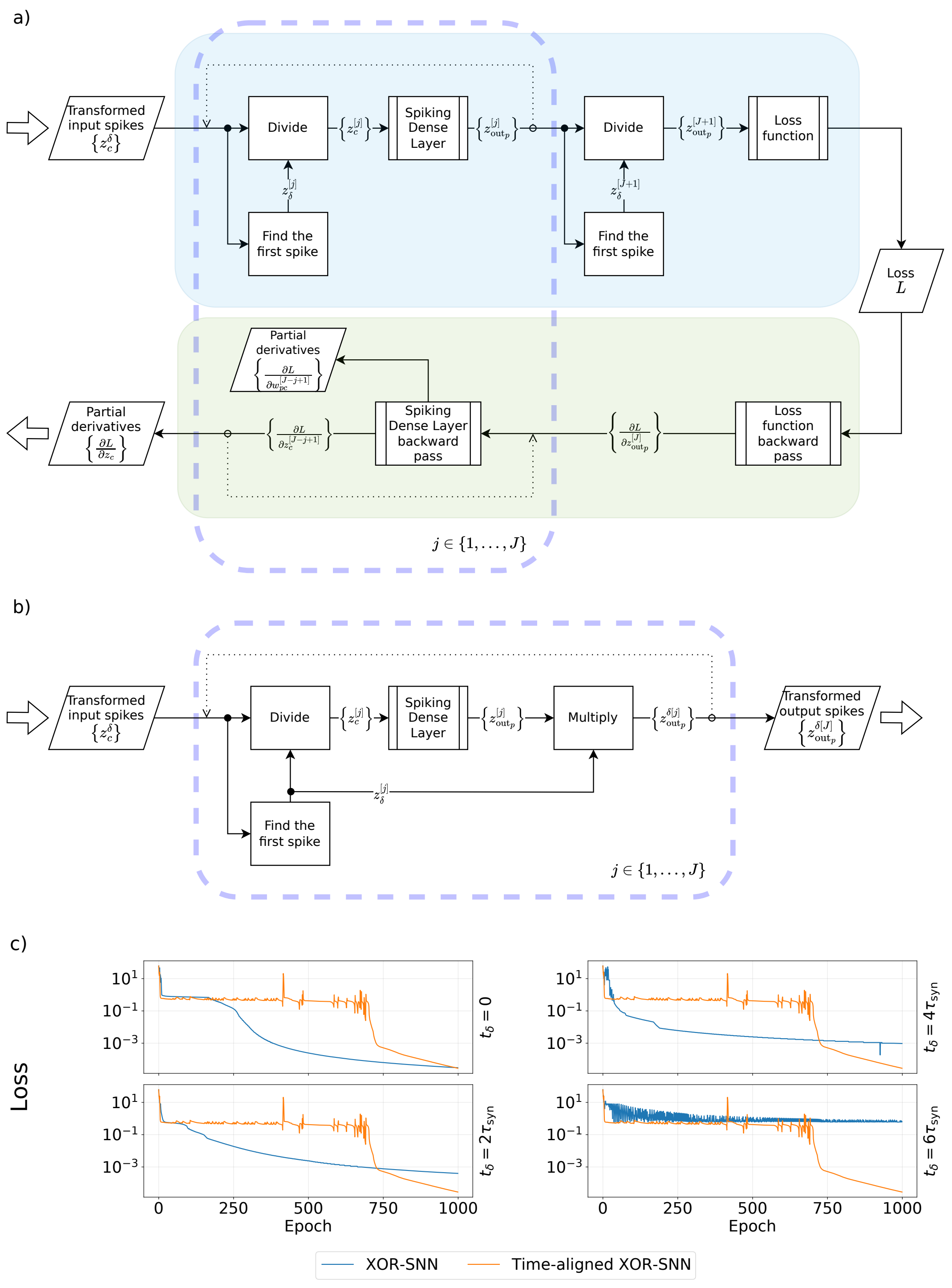

a)
Transformed input spikes $\{z_c^\delta\}$
Divide
$\{z_c^{[j]}\}$
Spiking Dense Layer
$\{z_{\text{out}_p}^{[j]}\}$
$z_\delta^{[j]}$
Find the first spike
Divide
$\{z_{\text{out}_p}^{[J+1]}\}$
Loss function
$z_\delta^{[J+1]}$
Find the first spike
Loss L
Partial derivatives $\left\{\frac{\partial L}{\partial w_{pc}^{[J-j+1]}}\right\}$
Partial derivatives $\left\{\frac{\partial L}{\partial z_c}\right\}$
$\left\{\frac{\partial L}{\partial z_c^{[J-j+1]}}\right\}$
Spiking Dense Layer backward pass
$\left\{\frac{\partial L}{\partial z_{\text{out}_p}^{[J]}}\right\}$
Loss function backward pass
$j \in \{1, \dots, J\}$
b)
Transformed input spikes $\{z_c^\delta\}$
Divide
$\{z_c^{[j]}\}$
Spiking Dense Layer
$\{z_{\text{out}_p}^{[j]}\}$
Multiply
$\{z_{\text{out}_p}^{\delta[j]}\}$
Transformed output spikes $\{z_{\text{out}_p}^{\delta[J]}\}$
$z_\delta^{[j]}$
Find the first spike
$j \in \{1, \dots, J\}$
c)
Loss
10^1
10^{-1}
10^{-3}
$t_\delta = 0$
$t_\delta = 2\tau_{\text{syn}}$
$t_\delta = 4\tau_{\text{syn}}$
$t_\delta = 6\tau_{\text{syn}}$
0
250
500
750
1000
Epoch
XOR-SNN
Time-aligned XOR-SNN

Figure 3.11: a) Flowchart of the signal propagation through a multilayer time-aligned SNN during training. The index $j \in \{1, \dots, J\}$ denotes different layers of the network. Dotted lines denote information that is passed as next layer input. b) Flowchart of the signal propagation through a multilayer time-aligned SNN during inference. c) Loss curves for XOR-SNN models trained with and without time-alignment versus time-shift t_δ applied to the input spikes. All models were trained with the same set of hyperparameters.

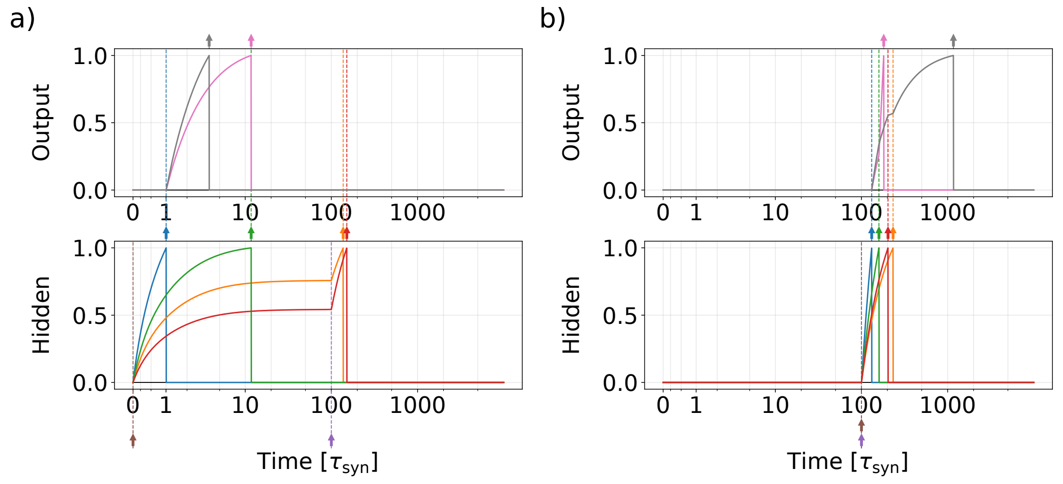


Figure 3.12: Selected spike-voltage plots of a XOR-SNN network responding to examples spiking at a) $\{t_0, t_1\}$ and b) $\{t_1, t_1\}$, for a model that uses time log-transform with $b = 10$.

An example of a simple function that satisfies these conditions is

$$g(t) = \log_b (t + 1) \ , \quad b > 1 \, . \tag{3.66}$$

Note that this function is applied not only to the input spikes, but also to all time-related constants of the model (for the IF neuron that means τ_{syn} and τ_{ref}). An important consequence of applying the log-transform approach to the input data is that it significantly lengthens the model response time, subject to the choice of the logarithm base b and model parameters V_{thr} and τ_{syn}. This is because the transform $g(t)$ becomes a part of the model and must be applied at both training- and inference-time. For example, Figure 3.12 presents two spike-voltage plots of a XOR-SNN model trained on examples with events occurring at either $t_0 = 0$ or $t_1 = 100\ \tau_{syn}$ for $b = 10$. With both inputs spiking at $t = t_1$ the model produces its last spike at about $t = 1100\ \tau_{syn}$, despite its inputs spiking much earlier.[f] Even though such model is able to converge to a solution whereas an unmodified SNN fails, it can be argued that this approach is impractical when low-latency responses are required of the model. Conversely, if processing the signal and computing a model prediction is all that is important (as is the case when applying the model to the Twitter bot classification problem in Section 3.3), then this seems to be a valid approach. It might be interesting to explore different approaches that mitigate the outlined problem of long model response time, such as introducing a new penalty term to the minimized loss function.

In conclusion, the problems and solutions discussed so far are merely workarounds to what seems to be a fundamental flaw of the model. Perhaps a different approach is needed, one that reformulates all learning rules in terms of interspike intervals (ISI) instead of absolute spike-times. Even then, those issues might still arise when events are sufficiently rare, causing ISIs

[f]Whether that spike is even necessary to solve the XOR problem is a topic of the discussion in the next Section.

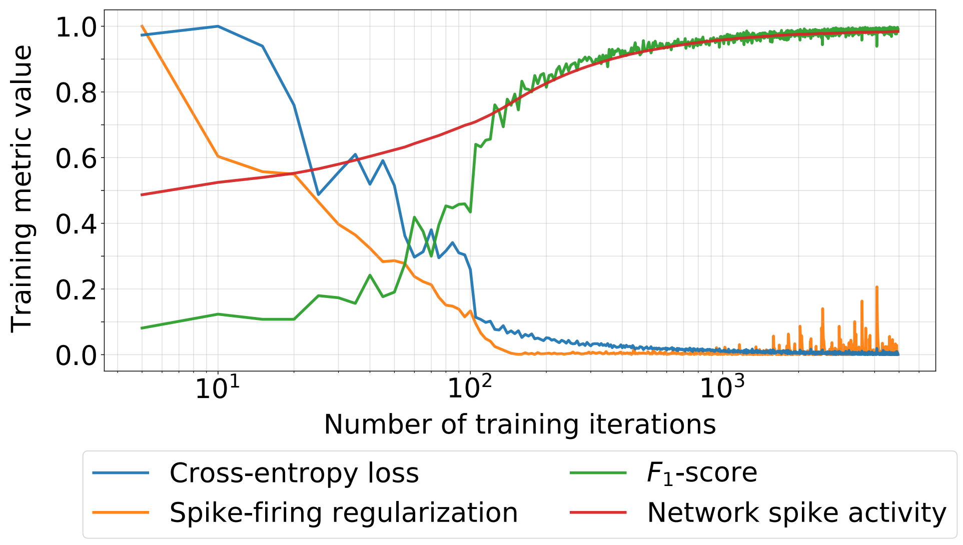


Figure 3.13: Training curves of a classification SNN model trained on MNIST data. Each loss function component (cross-entropy loss, spike-firing regularization) is paired with a score (F_1-score and network spike activity, respectively). The network spike activity score represents the number of neurons eliciting a spike relative to the number of neurons in the network. The loss components were min-max normalized.

to become too large. One could consider adaptively rescaling the layer inputs in order to keep them reasonably small, similarly to the batch normalization layer [162] in ANNs. While there exists a spiking equivalent to the batch normalization layer for the rate-coding SNN [112], it cannot be used in the time-coding setting of the discussed model. Overall, this seems to be an interesting topic to explore in the future.

3.2.3 Relaxing the neuron firing constraint

Minimizing the spike-firing penalty (3.11), which promotes the SNN activity, is crucial for the model to function properly. A randomly initialized set of layer weights more often than not prevents the model from propagating the signal through the spiking layers. Recall the SNN model trained on the MNIST digit classification task from Section 3.1.4.2. Figure 3.13 illustrates the training curves of this model. Note that the spike-firing penalty is extremely important during the initial training iterations, with its relative importance diminishing as training progresses. Only after the model is able to produce spikes at its output does the task-specific loss function component begin to play any role in the training process. Additionally, observe that the network activity score continues to increase alongside the F_1-score.

Recall from the reproducibility study in Section 3.1.4.2 that, in general, a significant number of hidden layer neurons generate a spike much later than in takes for the model to produce its first output spike. This implies that either these neurons inhibit the activity of subsequent layers, helping the model make a correct prediction in the process, or that they have no impact on the final decision. The latter scenario is energy-inefficient, but stems from the definition of the spike-firing penalty, which is applied to every neuron as long as it does not produce a spike. It can be argued that it is unnecessary to promote spiking activity when the model correctly solves the task at hand. We posit that the penalty term should be applied only when "the task is not solved". For classification models this entails that the output of the network is different from the expected ground truth. Such approach would result in models that are more efficient in terms of the number of spikes needed to propagate the signal through the network, compared to the original approach (which makes all neurons in the network produce at least one spike). The proposed heuristic is thus

$$R^*_{\text{spiking}} = \frac{1}{\sum_{n=1}^{N} \mathbf{1}\left(y_n \neq \hat{y}_n\right)} \sum_{n=1}^{N} \mathbf{1}\left(y_n \neq \hat{y}_n\right) R^{[n]}_{\text{spiking}}\,, \tag{3.67}$$

where n runs over all examples within a training minibatch N; $R^{[n]}_{\text{spiking}}$ is the spike-firing penalty term in (3.11) computed only for the n-th example, and $\mathbf{1}\left(A\right)$ is an indicator function. The argument of the indicator function is a short-hand notation for the output of the network $\hat{y}_n$ being different from the expected ground truth y_n. The proposed heuristic (3.67) dynamically scales the spike-firing penalty term during training – it acts as a mean penalty over the mini-batch examples if the network returns an incorrect output for all examples, and it assigns an increasingly larger weight to each incorrectly predicted example as the training progresses. For completeness

$$R^*_{\text{spiking}} = 0 \quad \text{if} \quad \underset{n\in\{1,\ldots,N\}}{\forall} \mathbf{1}\left(y_n \neq \hat{y}_n\right) = 0\,.$$

Note that having the correct network output does not necessarily mean that the task-specific loss term (e.g. cross-entropy loss (3.10)) for a given example is zero.

Figure 3.14 shows that applying this modified spike-firing penalty term to the XOR-SNN model produces a network that uses fewer spikes to achieve the desired logic. The neurons that do not fire in response to an input signal are called *quiescent* neurons. This is possible due to an implicit assumption that quiescent neurons produce a spike at $t_{\text{out}} = \infty$. In fact, Figure 3.14 shows that an SNN trained with this modified penalty term can have both completely quiescent (in green), as well as conditionally quiescent neurons (in gray).

The average number of spikes needed to process a signal by the model, compared to the actual number of neurons in the network, can be quantified. Let us formally define a group of quiescent neurons not firing in response to a given input signal $x(t)$ as

$$H_{q,x} = \{h \in H, t > 0 : V_{h,x}(t) < V_{thr}\}\,. \tag{3.68}$$

Then, the *network sparsity index* QN_x is a fraction of quiescent neurons $H_{q,x}$ to all neurons in

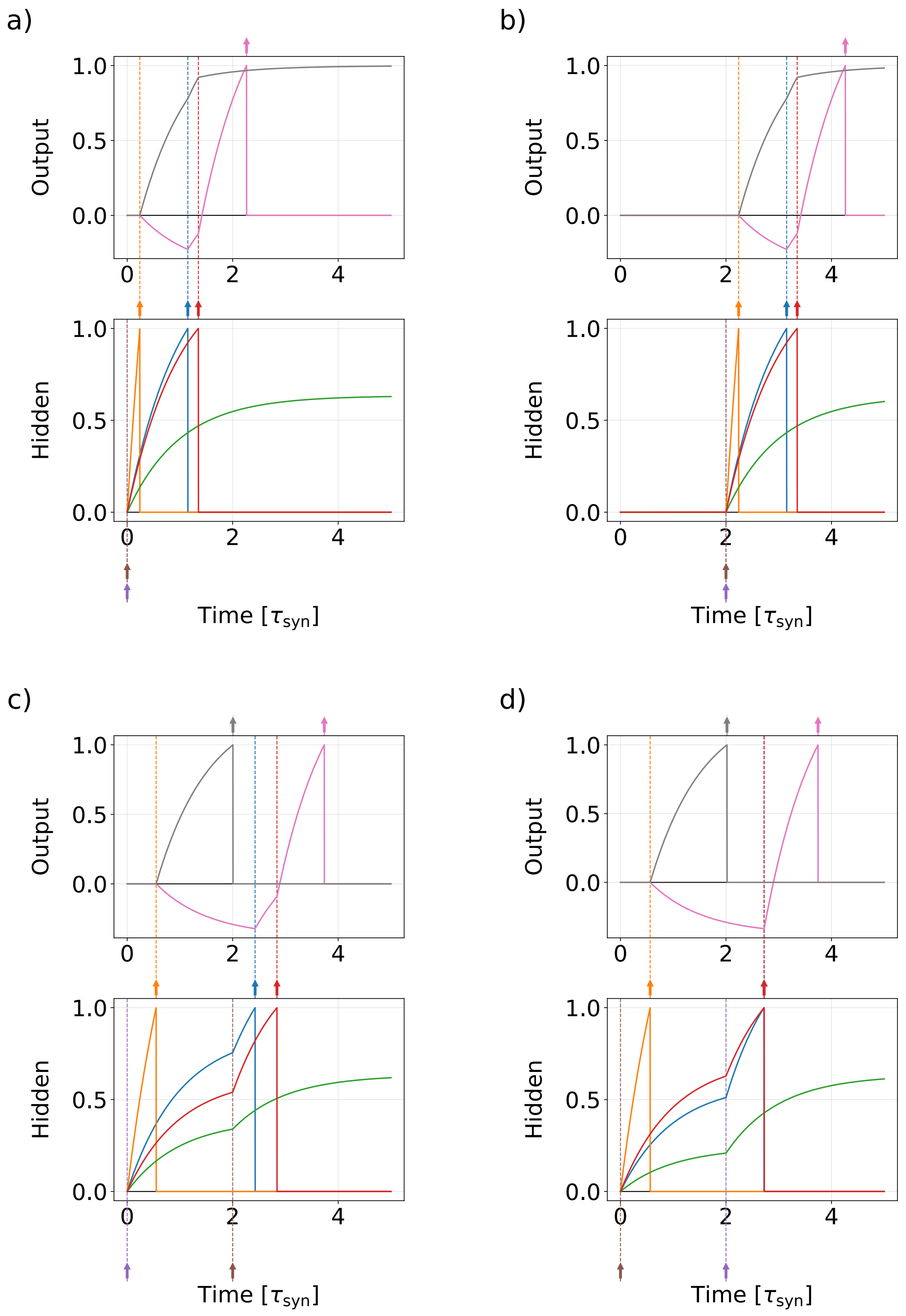


Figure 3.14: Spike-voltage plots of a time-to-first-spike SNN trained on the spiking XOR task with the modified spike-firing penalty term. a) 0 XOR 0 = 0, b) 1 XOR 1 = 0, c) 0 XOR 1 = 1, d) 1 XOR 0 = 1. Note that the gray curve in scenarios (a) and (b) does not cross the threshold V_{thr} = 1 (plots truncated for clarity).

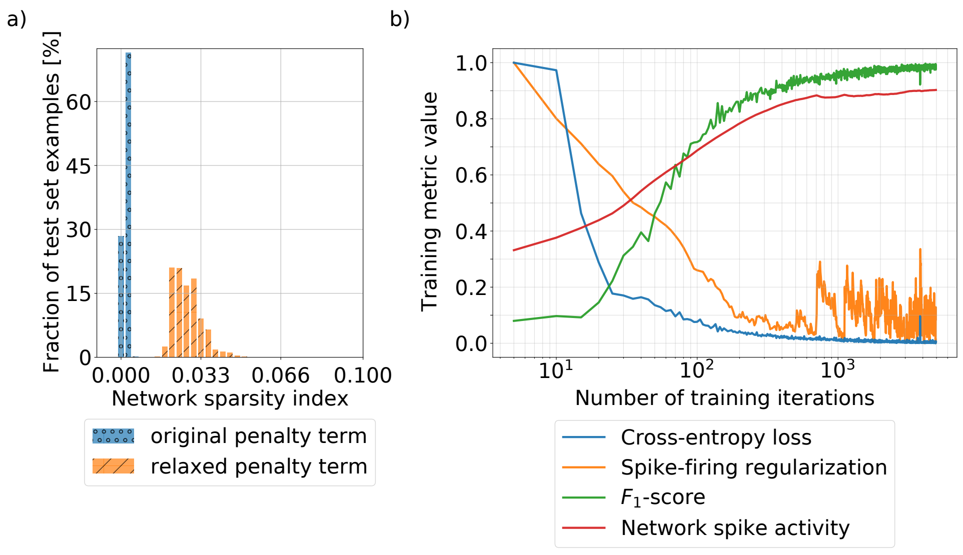


Figure 3.15: a) Network sparsity index empirical distributions for models trained on MNIST data depending on the spike-firing penalty term used during training. b) Training curves of a classification SNN model trained with the modified spike-firing penalty term on MNIST data.

hidden layers H

$$\mathrm{QN}_x = \frac{|H_{q,x}|}{|H|}\,. \tag{3.69}$$

With the help of the ratio QN_x it is possible to compare spiking activity sparsity of different models. To illustrate this concept, we train two models on the MNIST digit classification task. The only difference between the two models is that one is trained using the original spike-firing penalty (3.11), whereas the other employs the modified approach (3.67). All other hyperparameters were the same as in the preliminary study in Section 3.1.4.2. Both models achieved near-identical classification performance on test data (F_1-score of 0.971 vs. 0.970 for the modified penalty term). Figure 3.15a presents the hidden layer activation sparsity empirical distributions for the two models evaluated on test set data. It is clear that using the modified spike-firing penalty results in a model that uses fewer neurons to process examples. Moreover, the ratio QN_x for this model exhibits larger variance, suggesting that this neural activity sparsity is context-based, i.e., that a different subset of neurons will respond to each input signal. Furthermore, Figure 3.15b presents training curves for the model trained with the modified spike-firing penalty. When compared with the alternative (presented previously in Figure 3.13), we observe that the model is encouraged to minimize the cross-entropy term at earlier iterations, despite the network being relatively inactive at this stage. Note that while the network spike activity score continues to increase throughout the entire training procedure, it reaches a lower value than for the model trained with the original penalty term.

Overall, the proposed modification of the spike-firing penalty term allows finer control over the network spiking activity by not applying the penalty to cases which the network has already correctly solved. Interestingly, the network sparsity can be further influenced by choosing a different task-specific loss function itself, or by modifying the input data encoding. This can result in models processing the signal with even fewer events produced by the network. This topic is explored in further detail in Section 4.3.1.

3.2.4 Signal propagation rules with multiple inputs & multiple outputs (MIMO)

A significant limitation of the model is the inability to process information changing over time. It assumes that every neuron in the model, including input neurons, can elicit at most one spike (by implicitly setting $\tau_{\text{ref}} = \infty$). Our goal is to extend the signal propagation rules of the aforementioned model so that its hidden and output neurons are also able to produce multiple spikes each. For brevity, we use a *MIMO network* descriptor throughout this Section to refer to the model that supports signal propagation with multiple inputs and multiple outputs.

In case of multiple events arriving at the input synapse, the equation (3.1) becomes

$$\frac{dV(t)}{dt} = \sum_{c=1}^{C} w_c \sum_{j=1}^{T_c} i_c^{[j]}(t) = \sum_{c=1}^{C} \sum_{j=1}^{T_c} w_c^{[j]} i_c^{[j]}(t)\,, \tag{3.70}$$

where T_c is the total number of events observed in channel c. Note that this formula explicitly highlights the fact that every event $t_c^{[j]}$ of channel c is associated with the same weight $\{j : w_c^{[j]} = w_c\}$. Therefore, a single channel with T_c events is equivalent to T_c *virtual* (or *time-flattened*) channels with a single event each. Introducing a new index $\{k : k = 1, \ldots, K\}$ over these virtual input channels such that $K = \sum_{c=1}^{C} T_c$ we obtain the following representation of (3.70)

$$\frac{dV(t)}{dt} = \sum_{k=1}^{K} w_k i_k(t)\,, \tag{3.71}$$

which is identical to (3.1). It follows that the equations for forward and backward signal propagation through the network introduced in (3.5)-(3.9) still hold for the layer observing inputs spiking over time, provided that the indices are substituted where appropriate. The proposed idea of time-flattening the input signal and projecting input layer weights is presented in Figure 3.16a.

Furthermore, allowing each neuron to spike more than once requires setting a finite non-negative refractory period τ_{ref}. In that case the causal set of input spikes becomes

$$Q_m = \begin{cases} \{k : t_k < t_{\text{out}}^{[m]}\} & \text{if } m = 1 \\ \{k : t_{\text{out}}^{[m-1]} + \tau_{\text{ref}} < t_k < t_{\text{out}}^{[m]}\} & \text{otherwise} \end{cases}, \tag{3.72}$$

where $\{m : m = 1, \ldots, M\}$ is the index over the sequence of neuron output spikes. Computing the m-th output spike time can be done iteratively, noting that M is at most equal to K (the

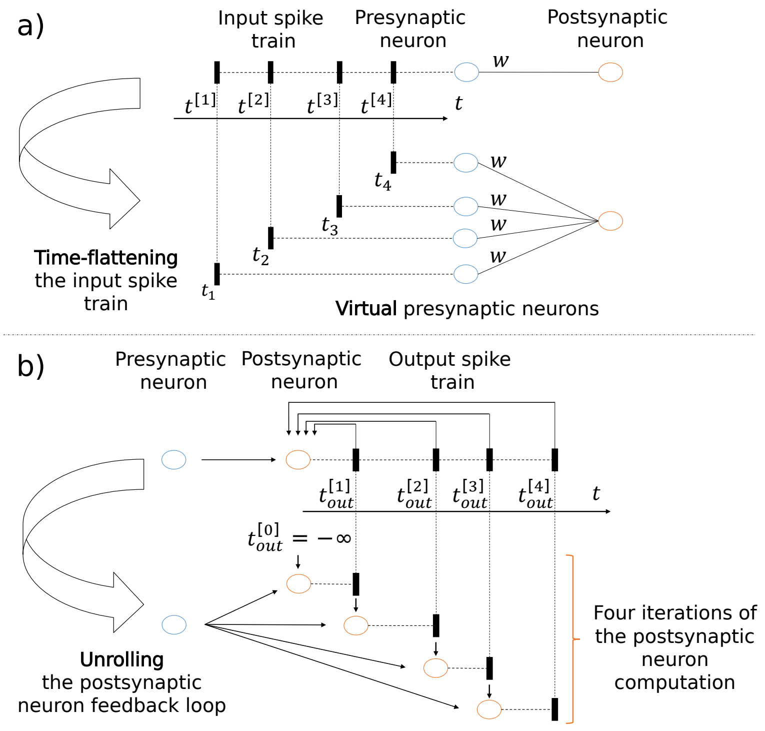

a)
Input spike train
Presynaptic neuron
Postsynaptic neuron
w
$t^{[1]}$ $t^{[2]}$ $t^{[3]}$ $t^{[4]}$ t
t_4
t_3
t_2
t_1
Time-flattening the input spike train
Virtual presynaptic neurons
b)
Presynaptic neuron
Postsynaptic neuron
Output spike train
$t_{out}^{[1]}$ $t_{out}^{[2]}$ $t_{out}^{[3]}$ $t_{out}^{[4]}$ t
$t_{out}^{[0]} = -\infty$
Unrolling the postsynaptic neuron feedback loop
Four iterations of the postsynaptic neuron computation

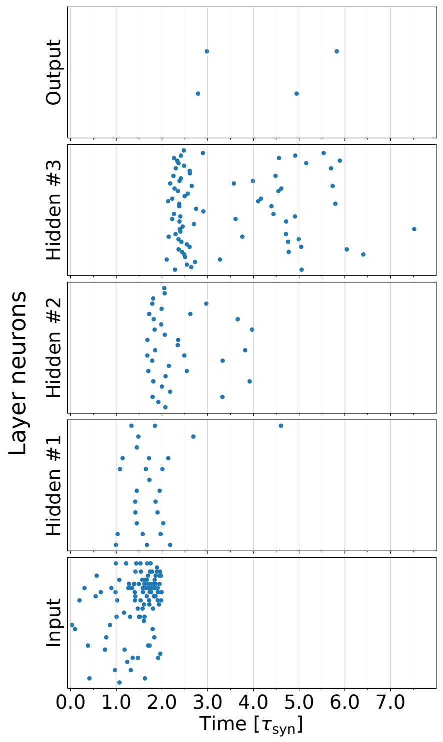
Figure 3.16: Signal propagation rules in the MIMO SNN. The flow of time is represented by an axis going from left to right, i.e., the earliest spike is at the left side of each subfigure. a) A presynaptic neuron that observes multiple input spikes can be represented as multiple virtual presynaptic neurons, each observing a single spike. All virtual presynaptic neurons have the same weight between them and the postsynaptic neuron, identical to the weight associated with the original connection before time-flattening. b) The ability of the postsynaptic neuron to produce spikes depends on all input spike trains from presynaptic neurons, as well as the previously generated output. This feedback loop imposed by the spike causality principle can be unrolled over time, where the postsynaptic neuron computation is repeated with the same input spike trains, but for different timestamps of the previously generated event. This cascade proceeds until it is impossible for the postsynaptic neuron to generate an output spike. The implicit output spike at $t_{\text{out}}^{[0]} = -\infty$ designates the initial state of the postsynaptic neuron, i.e., it has not generated a spike yet.

neuron cannot spike more often than once per input spike). The iterative computation over m can be stopped early when an empty causal set is encountered ($Q_m = \varnothing$). Taking this new definition of a causal set into consideration, the implicit formula for $t_{\text{out}}^{[m]}$ becomes

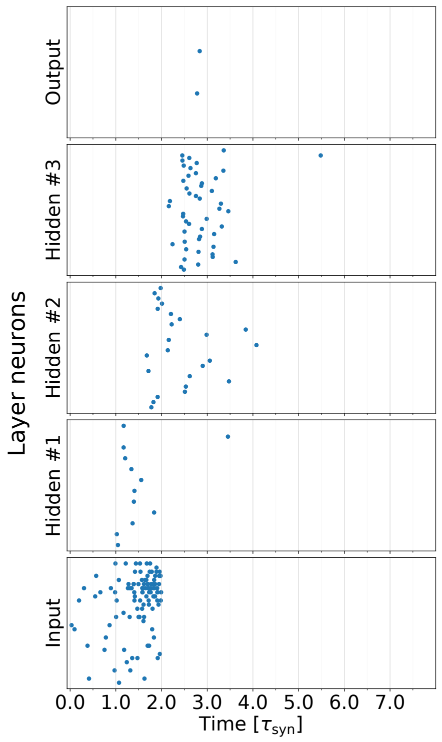
$$z_{\text{out}}^{[m]} = \frac{\sum_{k \in Q_m} w_k z_k}{\sum_{k \in Q_m} w_k - \frac{V_{\text{thr}}}{\tau_{\text{syn}}}}, \tag{3.73}$$

whereas the partial derivatives are

$$\frac{\partial z_{\text{out}}^{[m]}}{\partial z_k} = \begin{cases} \frac{w_k}{\sum_{k \in Q_m} w_k - \frac{V_{\text{thr}}}{\tau_{\text{syn}}}} & \text{if } k \in Q_m \\ 0 & \text{otherwise} \end{cases}, \tag{3.74}$$



(a) $\tau_{\text{ref}} = 0.1$



(b) $\tau_{\text{ref}} = 1$

Figure 3.17: Spike raster plot for models trained with two different values of τ_{ref} responding to the same input example.

$$\frac{\partial z_{\text{out}}^{[m]}}{\partial w_k} = \begin{cases} \frac{z_k - z_{\text{out}}^{[m]}}{\sum_{k \in Q_m} w_k - \frac{V_{\text{thr}}}{\tau_{\text{syn}}}} & \text{if } k \in Q_m \\ 0 & \text{otherwise} \end{cases}, \tag{3.75}$$

and additionally

$$\frac{\partial z_{\text{out}}^{[m]}}{\partial w_c} = \sum_{\{k:\, w_k \triangleq w_c\}} \frac{\partial z_{\text{out}}^{[m]}}{\partial w_k}, \tag{3.76}$$

where the set $\{k : w_k \triangleq w_c\}$ denotes the subset of all k for which the virtual (time-flattened) weight w_k corresponds to the original w_c. Note the absence of any explicit dependence between two consecutive output spikes $\{t_{\text{out}}^{[m]}, t_{\text{out}}^{[m+1]}\}$ in equations (3.73)-(3.76), or equivalently $\forall_m \frac{\partial z_{\text{out}}^{[m+1]}}{\partial z_{\text{out}}^{[m]}} = 0$. This is fundamentally different from signal propagation in the backpropagation through time (BPTT) algorithm [163], and is a direct result of the IF neuron's independence on its own history (i.e., it is memoryless). Instead, the gradients are simply summed over output spikes. This concept of iterating over the M output spikes is summarized in Figure 3.16b. Signal propagation through the MIMO SNN model can be visualized using a spike raster plot, such as the in Figure 3.17.

Overall, the proposed algorithm shows how to simulate and train a time-coding MIMO SNN by time-flattening the presynaptic spike trains and unrolling the postsynaptic neuron feedback loop imposed by the spike causality principle. This procedure is sufficient to encode and process information with multiple spikes. However, it must be stressed that the concepts described above and illustrated in Figure 3.16 specifically refer to operating the model on conventional hardware. Once trained, the spiking network (i.e, synaptic weights and neuron-specific hyperparameters) can be realized on existing neuromorphic hardware, in which case the virtual presynaptic neurons are no longer needed (hence the name) as reusing neurons for multiple input and output events is implied. The only requirement is that the device either implements or approximates the IF neuron computation (such as IBM TrueNorth [164] or Intel Loihi [165]).

The introduced signal propagation rules for the MIMO networks are tested on two machine learning problems studied in later sections of this thesis. Section 3.3 revisits the Twitter bot classification problem, whereas Chapter 5 is focused on vehicle identification based on magnetic profile measurements. The main difference between the two tasks is that in the former the data already exists as an event sequence with a single event type,[g] whereas the latter requires converting the underlying signal into the spiking domain.

3.2.4.1 Vectorized computation through a MIMO layer – forward pass

The proposed MIMO signal propagation rules require small adjustments to the vectorized computation proposed in Section 3.2.1. Crucially, the computation must now be described in terms of fourth-order tensors (albeit singular dimensions can also be present). Let $\mathcal{Z} \in [1, \infty]^{N\times 1\times C\times \Theta}$ be the layer input tensor with Θ denoting the largest number of events observed in a single input channel of the minibatch. Sequences included in $\mathcal{Z}$ need not have the same number of events as they can simply be padded with non-event indicator ($z_{nc\vartheta} = \infty$) across the last two modes. Then, time-flattening $\mathcal{Z}$ according to the virtual index $\{k : k = 1, \dots, K\}$ is simply the result of mode-1 matricization of $\mathcal{Z}$. The resulting matrix $\mathbf{Z}_{(1)} \in [1, \infty]^{N\times 1\times K\times 1}$ has elements

$$\mathbf{Z}_{(1)} = \begin{bmatrix} z_{111} & z_{121} & \cdots & z_{1C1} & z_{112} & \cdots & z_{1C\Theta} \\ z_{211} & z_{221} & \cdots & z_{2C1} & z_{212} & \cdots & z_{2C\Theta} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \ddots & \vdots \\ z_{N11} & z_{N21} & \cdots & z_{NC1} & z_{N12} & \cdots & z_{NC\Theta} \end{bmatrix}. \tag{3.77}$$

[g]We show in the relevant section that training a MIMO network on a single event type is detrimental to the model performance.

Conversely, time-flattening the synaptic weights matrix $\mathbf{W} \in \mathbb{R}^{1\times P\times C\times 1}$ requires mode-4 broadcasting

$$\mathcal{W} = \text{bcast}_4\left(\mathbf{W}; \Theta\right) \tag{3.78}$$

in order to associate each input neuron spiking over time with the same weight, and then mode-2 matricization of the resulting tensor afterwards. These steps produce a matrix $\mathbf{W}_{(2)} \in \mathbb{R}^{1\times P\times K\times 1}$ with elements

$$\mathbf{W}_{(2)} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1C} & w_{11} & \dots & w_{1C} \\ w_{21} & w_{22} & \dots & w_{2C} & w_{21} & \dots & w_{2C} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \ddots & \vdots \\ w_{P1} & w_{P2} & \dots & w_{PC} & w_{P1} & \dots & w_{PC} \end{bmatrix}. \tag{3.79}$$

Note that $\mathbf{Z}_{(1)}$ and $\mathbf{W}_{(2)}$ effectively have the same dimensionality as the forward pass inputs $\mathbf{Z}$ (3.38) and $\mathbf{W}$ (3.39). As the rest of the vectorized algorithm is specified in terms of $\mathbf{Z}$ and $\mathbf{W}$, it follows that $\mathbf{Z}_{(1)}$ and $\mathbf{W}_{(2)}$ can be used in their stead and no further modifications are necessary in order to process inputs spiking over time. This is illustrated in Figure 3.18. A clear limitation of the matricization-based solution is that the shape of all tensors used in the computation dynamically scales with the total number of events in the presented batch of examples. Therefore, this might be an unfeasible approach when the data is represented by many channels with many events each. As such, from a more practical perspective, one should avoid passing examples with vastly different total number of events within the same batch.

Allowing the output neurons to spike multiple times in response to the input sequence introduces a feedback loop into the signal propagation rules (3.72). This dependency makes it impossible to devise an algorithm that computes all outputs $\mathcal{Z}_{\text{out}} \in [1,\infty]^{N\times P\times 1\times M}$ (where M denotes the largest number of events observed in an output channel of the minibatch) in a single forward pass through the layer. This necessitates an iterative approach that builds $\mathcal{Z}_{\text{out}}$ (frontal) slice-by-slice. Fortunately, within a single iteration the vectorized algorithm outlined in Section 3.2.1.2.2 still holds, provided that there exists a mechanism that discards a subset of input spikes depending on the most recent output spike of the neuron. This follows from the memoryless property of the IF neuron: anything that occurred before the output spike has no impact on neuron's future operation once the refractory period subsides. Recall from (3.43) that $\mathcal{Z} \in [1,\infty]^{N\times P\times K\times 1}$ is a third-order tensor representing input spikes broadcasted across the second mode.[h] Presume that $\mathcal{Z}'_{\text{out}} \in [1,\infty]^{N\times P\times K\times M}$ is a mode-3-broadcasted output spikes tensor constructed from the intermediate results of M iterations. Similarly, presume that $\mathcal{Z}' \in [1,\infty]^{N\times P\times K\times M}$ is the masked input spikes tensor corresponding to the M iterations.

[h]Whether $\mathcal{Z}$ denotes the actual or time-flattened inputs is irrelevant for the present discussion.

Then the elements of $\mathcal{Z}'$ are

$$z'_{npkm} = \begin{cases} \infty & \text{if} \quad m > 1 \ \wedge \ (\mathcal{Z})_{npk} \leq z_{\text{ref}} \left(\mathcal{Z}'_{\text{out}}\right)_{npk(m-1)} \\ (\mathcal{Z})_{npk} & \text{otherwise} \end{cases}, \tag{3.80}$$

where $z_{\text{ref}} = \exp\left(\tau_{\text{ref}}/\tau_{\text{syn}}\right)$ represents the refractory period. Of course, according to the discussion in Section 3.2.4, the number of iterations M is not known upfront and so tensors $\mathcal{Z}'$, $\mathcal{Z}'_{\text{out}}$ must be built (frontal) slice-by-slice: passing $(\mathcal{Z}')_{:::m}$ as the forward pass input produces $\left(\mathcal{Z}'_{\text{out}}\right)_{:::m}$. No further changes are necessary in the vectorized forward pass implementation (Figure 3.18).

From a practical perspective, it is not necessary to pass the entire masked input spikes tensor through the layer at every iteration. It is possible to take progressively smaller subtensors of $(\mathcal{Z}')_{:::m}$ at every iteration by:

- removing horizontal slices corresponding to examples that do not elicit any further response in either output neuron,
- dropping lateral slices for output neurons which have not fired for the most recent iteration for any of the examples,
- removing frontal slices for input neurons which have all of their spikes masked according to the causality principle, or observe no events altogether,

whenever applicable.[i]

3.2.4.2 Vectorized computation through a MIMO layer – backward pass

Modifications in the backward pass of the algorithm stem from the fact that input $\mathcal{Z} \in \mathbb{R}^{N\times 1\times C\times \Theta}$ and output spikes $\mathcal{Z}_{\text{out}} \in \mathbb{R}^{N\times P\times 1\times M}$ are now represented by third-order tensors. Note that while the last tensor mode represents a "spike count dimension" for both $\mathcal{Z}$ and $\mathcal{Z}_{\text{out}}$, in general $\Theta \neq M$. It follows that the backward pass input signal $\frac{\partial L}{\partial \mathcal{Z}_{\text{out}}} \in \mathbb{R}^{N\times P\times 1\times M}$ is also a third-order tensor. We wish to compute the partial derivatives $\frac{\partial L}{\partial \mathcal{Z}} \in \mathbb{R}^{N\times 1\times C\times M}$ and $\frac{\partial L}{\partial \mathbf{W}} \in \mathbb{R}^{1\times P\times C\times 1}$. Let us first consider the partial derivatives with respect to the time-flattened inputs $\mathbf{Z}_{(1)}$ (3.77) and weights $\mathbf{W}_{(2)}$ (3.79). From the chain rule of derivatives we have

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{Z}_{(1)}} &= \sum_{m=1}^{M}\sum_{p=1}^{P}\left(\frac{\partial L}{\partial \mathcal{Z}'_{\text{out}}} \odot \frac{\partial \mathcal{Z}'_{\text{out}}}{\partial \mathcal{Z}'}\right)_{:p:m} \\ \frac{\partial L}{\partial \mathbf{W}_{(2)}} &= \sum_{m=1}^{M}\sum_{n=1}^{N}\left(\frac{\partial L}{\partial \mathcal{Z}'_{\text{out}}} \odot \frac{\partial \mathcal{Z}'_{\text{out}}}{\partial \mathcal{W}'}\right)_{n::m} \end{aligned}, \tag{3.81}$$

[i]This could be further optimized by expressing the computation in terms of sparse matrices, which would allow passing specific elements rather than subtensors through the layer.

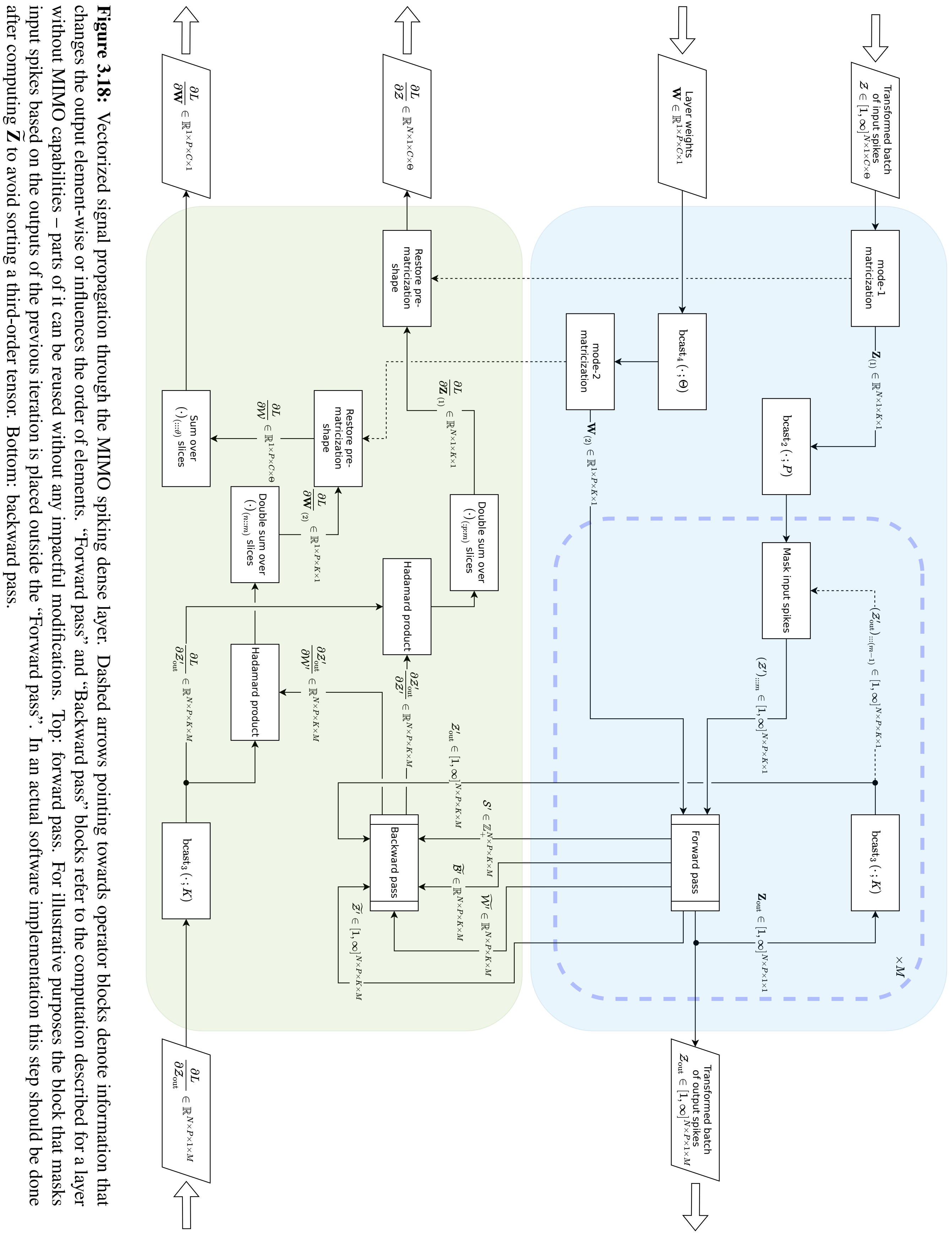

Transformed batch of input spikes $\mathcal{Z} \in [1,\infty]^{N\times 1\times C\times\Theta}$
Layer weights $\mathbf{W} \in \mathbb{R}^{1\times P\times C\times 1}$
mode-1 matricization
$\mathbf{Z}_{(1)} \in \mathbb{R}^{N\times 1\times K\times 1}$
bcast$_4$ $(\cdot;\Theta)$
mode-2 matricization
$\mathbf{W}_{(2)} \in \mathbb{R}^{1\times P\times K\times 1}$
bcast$_2$ $(\cdot;P)$
Mask input spikes
$(\mathcal{Z}'_{\text{out}})_{::::(m-1)} \in [1,\infty]^{N\times P\times K\times 1}$
$(\mathcal{Z}')_{:::m} \in [1,\infty]^{N\times P\times K\times 1}$
Forward pass
bcast$_3$ $(\cdot;K)$
$\mathbf{Z}_{\text{out}} \in [1,\infty]^{N\times P\times 1\times 1}$
$\times M$
Transformed batch of output spikes $\mathcal{Z}_{\text{out}} \in [1,\infty]^{N\times P\times 1\times M}$
$\frac{\partial L}{\partial \mathbf{W}} \in \mathbb{R}^{1\times P\times C\times 1}$
$\frac{\partial L}{\partial \mathcal{Z}} \in \mathbb{R}^{N\times 1\times C\times\Theta}$
Restore pre-matricization shape
$\frac{\partial L}{\partial \mathbf{Z}_{(1)}} \in \mathbb{R}^{N\times 1\times K\times 1}$
Restore pre-matricization shape
$\frac{\partial L}{\partial \mathcal{W}} \in \mathbb{R}^{1\times P\times C\times\Theta}$
Sum over $(\cdot)_{::::\theta}$ slices
Double sum over $(\cdot)_{:p:m}$ slices
$\frac{\partial L}{\partial \mathbf{W}_{(2)}} \in \mathbb{R}^{1\times P\times K\times 1}$
Double sum over $(\cdot)_{n:::m}$ slices
Hadamard product
Hadamard product
$\frac{\partial \mathcal{Z}'_{\text{out}}}{\partial \mathcal{Z}'} \in \mathbb{R}^{N\times P\times K\times M}$
$\frac{\partial \mathcal{Z}'_{\text{out}}}{\partial \mathcal{W}'} \in \mathbb{R}^{N\times P\times K\times M}$
$\frac{\partial L}{\partial \mathcal{Z}'_{\text{out}}} \in \mathbb{R}^{N\times P\times K\times M}$
$\mathcal{Z}'_{\text{out}} \in [1,\infty]^{N\times P\times K\times M}$
$\mathcal{S}' \in \mathbb{Z}_+^{N\times P\times K\times M}$
$\widetilde{\mathcal{B}}' \in \mathbb{R}^{N\times P\times K\times M}$
$\widetilde{\mathcal{W}}' \in \mathbb{R}^{N\times P\times K\times M}$
$\widetilde{\mathcal{Z}}' \in [1,\infty]^{N\times P\times K\times M}$
Backward pass
bcast$_3$ $(\cdot;K)$
$\frac{\partial L}{\partial \mathcal{Z}_{\text{out}}} \in \mathbb{R}^{N\times P\times 1\times M}$

Figure 3.18: Vectorized signal propagation through the MIMO spiking dense layer. Dashed arrows pointing towards operator blocks denote information that changes the output element-wise or influences the order of elements. “Forward pass” and “Backward pass” blocks refer to the computation described for a layer without MIMO capabilities – parts of it can be reused without any impactful modifications. Top: forward pass. Bottom: backward pass. For illustrative purposes the block that masks input spikes based on the outputs of the previous iteration is placed outside the “Forward pass”. In an actual software implementation this step should be done after computing $\widetilde{\mathbf{Z}}$ to avoid sorting a third-order tensor.

where

$$\frac{\partial L}{\partial \mathcal{Z}'_{\text{out}}} = \text{bcast}_3\left(\frac{\partial L}{\partial \mathcal{Z}_{\text{out}}}; K\right), \tag{3.82}$$

and $\frac{\partial \mathcal{Z}'_{\text{out}}}{\partial \mathcal{Z}'}$ and $\frac{\partial \mathcal{Z}'_{\text{out}}}{\partial \mathcal{W}'}$ are given by (3.61) and (3.62), respectively, noting that fourth-order tensors computed in the MIMO layer forward pass are used instead. The sum over mode-m in (3.81) is a direct result of the IF neuron's independence on its own history, i.e., spikes originating from the same output neuron are independent of one another.

The partial derivatives $\frac{\partial L}{\partial \mathcal{Z}}$ and $\frac{\partial L}{\partial \mathcal{W}}$ can be obtained by reshaping $\frac{\partial L}{\partial \mathbf{Z}_{(1)}}$ and $\frac{\partial L}{\partial \mathbf{W}_{(2)}}$ such that the respective n-mode matricizations are reverted. Then,

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{\theta=1}^{\Theta}\left(\frac{\partial L}{\partial \mathcal{W}}\right)_{:::\vartheta}, \tag{3.83}$$

which concludes the backward pass. The backward pass part of the algorithm is summarized in Figure 3.18.

3.2.4.3 Computational complexity of the MIMO SNN

Let us briefly comment on the computational burden of the MIMO SNN. This algorithm slightly increases the complexity of the single-spike version in [123]. While the proposed MIMO SNN approach introduces an unavoidable feedback loop resulting from the spike causality principle, it does not prevent the model from training with modern deep learning software frameworks, such as TensorFlow [158] or PyTorch [159]. During such iterative search over output spikes, the SNN is no different from a nonspiking RNN.

Note that the proposed algorithm iterates over the space of discrete output spike events rather than discretizing time at a predefined time-resolution. This leads to significantly fewer iterations when computing the full output spike train of each neuron than the alternative requiring a discretized time simulation. Additionally, this allows the MIMO SNN to compute arbitrarily late events with respect to the reference time, whereas discretized-time algorithms are limited in scope by the simulation window. Note that it is possible to impose an upper limit on the number of events generated by a single neuron in order to reduce the number of iterations, should the need arise. For now, we rely on the exhaustive search over the event space. Importantly, this event space is finite as each neuron cannot generate more events than it observes across all synapses.

Nevertheless, the MIMO SNN adversely scales with the number of events observed by presynaptic neurons, as well as by those generated by its neurons. The former has the biggest effect on the input layer, in which the number of virtual (time-flattened) channels can be extremely large when the network has numerous input neurons, each observing lengthy spike trains. This limits the applicability of our approach to multichannel data streams without too many events (although it is difficult to estimate what this upper limit actually is), unless the dataset is preprocessed to contain fewer events. Furthermore, the average processing time increases with the number of spikes generated by the layer as the feedback loop must be unrolled over time. However, note that each successive spike is less likely to be generated (because the causal set shrinks with each iteration), making the computational complexity of subsequent iterations smaller than preceding ones. Fortunately, the network spiking activity can be controlled by the τ_{ref} hyperparameter, which is evidenced in the following Sections.

3.3 Applications – Twitter bot detection

So far, the presented time-to-first-spike SNN model was assessed in terms of toy problems (such as the XOR logic), or a simple image-based dataset (MNIST). While the insights gained during this study were invaluable in identifying the limitations of the proposed model and proposing extensions that mitigate those shortcomings, it is important to consider a real-life applicability of the model. For this reason we would like to revisit the Twitter bot detection problem which was introduced in Section 2.5 in the context of evaluating a classical, point-process-based approach. Given that each sequence from the Twitter dataset is univariate (i.e., multiple spikes occurring in a single input channel) and that the events occur at timescales differing by almost five orders of magnitude, it presents an opportunity to verify the proposed MIMO signal propagation rules, as well as to evaluate ideas related to addressing the problem of training instabilities manifesting due to large absolute event-time values. Additionally, choosing to analyze the model on the same problem as before lets us compare the two approaches.

3.3.1 Preprocessing

The tweet-retweet sequences described in Section 2.5.2 are characterized by two peculiar traits that make designing an SNN-based classifier difficult. First, we note that all events occur in only one channel. If all neurons in the network have the same value of parameters τ_{syn}, τ_{ref} and V_{thr}, then this scenario imposes additional constraints on the weights during training. Note that in this scenario the synaptic weight of each connection must be positive, otherwise the postsynaptic neuron is unable to produce any spike. This also means that each postsynaptic neuron eventually produces a spike as the IF neurons are unable to lose charge if all presynaptic weights are positive. Additionally, if some weights are too large, then it is possible that the associated postsynaptic neurons will produce nearly identical spike trains, with only slight variations in spike frequency and their timing (illustrated in Figure 3.19). While it is certainly not impossible to train a network with such constraints on the input layer, we can expect this to

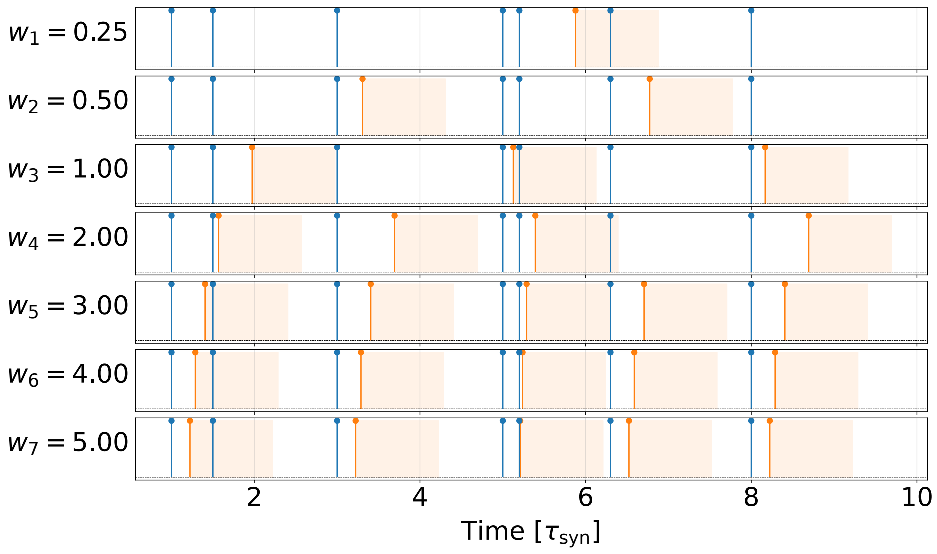


Figure 3.19: Simulated spike trains from a simple network with one input neuron and seven postsynaptic neurons. In blue: input spike train (the same in all rows), in orange: spikes generated by postsynaptic neurons. The shaded area denotes the refractory period after generating a spike. All output neurons have the same values of parameters τ_{syn}, τ_{ref} and V_{thr} with the only difference between them being the synaptic weight w. Note that if the weight it too large (in this case $w \geq 3$), the neuron elicits a spike in response to every input event, unless it occurs during the refractory period, effectively repeating the input sequence. This means that a group of postsynaptic neurons is redundant because they produce almost identical spike trains. In such scenario, output sequence variability could be improved by adjusting the values of τ_{syn}, τ_{ref} and V_{thr} for each neuron individually.

be more difficult. In fact, in Section 3.3.3.3 we empirically show that training a model with a single input neuron is indeed more challenging than the alternative.

For these reasons we explored the possibility of transforming the input spike trains into multiple sequences with fewer events. This is analogous to conducting feature engineering instead of relying on trainable feature extractors in artificial neural networks. Loosely inspired by the technique of binning used in neuroscientific studies [166], we identify a collection of *bins* that divide the spike train aggregated over all examples into sub-sequences with approximately the same number of events in each bin.[j] This concept is illustrated in Figure 3.20 by dividing the empirical density functions of the two classes. Importantly, the step that identifies binning thresholds is computed over the training examples. Each sub-sequence can then be shifted so that it starts at $t = 0$ by subtracting the corresponding binning threshold.

We note that binning is a valid strategy for this specific problem because it is assumed that events aggregated into a tweet-retweet sequence of each user are independent of one another [5]. While this is a reasonable assumption given the data, it seems plausible that graph community

[j]By contrast, in neuroscience binning produces the number (or an average number of) events that had occurred in a given time interval over multiple repetitions of the experiment.

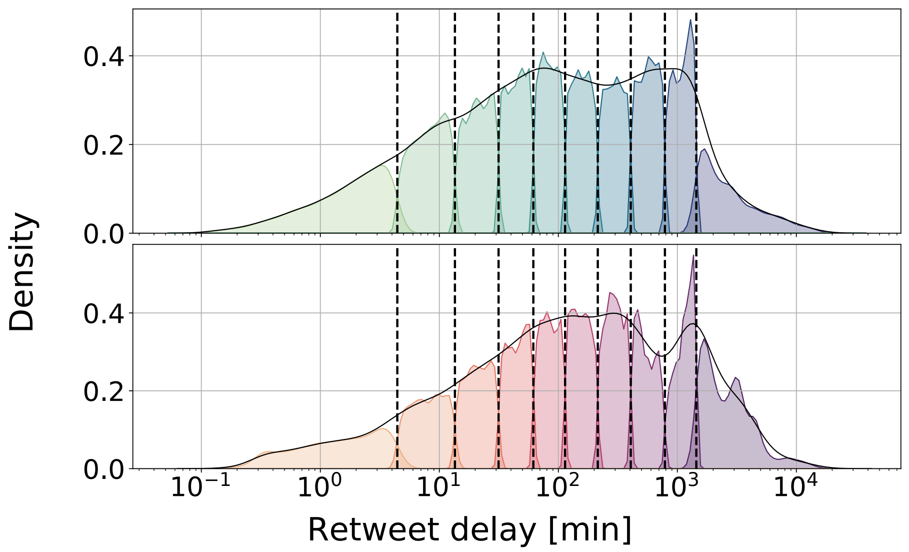


Figure 3.20: The result of binning the empirical density functions of the two classes (above – legitimate users; below – bots) into 10 bins over a given data split.

structure and programmed bot behavior play an important role in what gets retweeted and when. Additionally, the IF neuron exhibits the memoryless property where the internal state of the neuron (membrane voltage) is preserved regardless of how recent was the previous event. Finally, as a side note, taking the binning approach to the extreme would result in a multitude of thin bins such that there is at most one event per bin for each example. We did not consider this to be worth pursuing as it would greatly increase the number of neurons needed to represent the signal in the input layer of the network.

The second point that needs to be addressed is that events occur at timescales differing by almost five orders of magnitude. At such scales the transformed events in (3.6) easily surpass the limits of double-precision floating-point data format, as has been shown in Section 3.2.2. This problem still persists even after binning the sequences and shifting each sub-sequence so that it has a common starting time of $t = 0$. It is easy to see (Figure 3.20) that for bins with a higher index, the range of event times, while reduced, is still relatively large for the network. And so, we can either increase the number of bins (but as discussed previously this does not seem to be a suitable approach), or transform each sequence to reduce the range of observed values. We opt to do the latter approach. Therefore, we apply a log-transform to the binned sub-sequences with events occurring in range $t \in [T_{c-1}, T_c)$ such that

$$\forall_{t \in [T_{c-1}, T_c)} \;\; g(t; b_c, c) = \log_{b_c} (t - T_{c-1} + 1) \;, \tag{3.84}$$

for $b_c > 1$, where $c \in \{1, 2, \ldots\}$ is the index of the bin, T_c is the upper boundary of the c-th bin (i.e., the threshold), and $T_0 = 0$. Each bin corresponds to a single network input channel, hence we reuse the index c to make this explicit and avoid introducing additional notation. Note that each transformed sequence is shifted to start at $t = 0$. This transform is controlled by a single parameter b_c, the base of the logarithm. Note that $g(t; b_c, c)$ is a decreasing function of b_c.

In general, the transform in (3.84) allows setting a different b_c for each bin (channel). In our experimental scope we consider two strategies for selecting b_c:

1) Setting the same base $b > 1$ for all bins

$$\forall_{t\in[T_{c-1},T_c)}\ g_1(t; b, c) = \log_b\left(t - T_{c-1} + 1\right)\ . \tag{3.85}$$

2) Adjusting the base of the logarithm for each bin separately so that all bins have the same range of values after transform

$$g_2(t; \kappa, c) = \frac{\kappa}{g_1(T_c; b, c)} g_1(t; b, c)\,, \tag{3.86}$$

where κ is the time-instant assigned to the threshold T_c after transform, i.e., $\kappa = g_2(T_c; \kappa, c)$. Note that in this strategy the value of the parameter b does not matter as the actual logarithm base for a given bin c is controlled by the boundaries T_{c-1}, T_c and the parameter κ. For simplicity, we use the same κ for all bins.

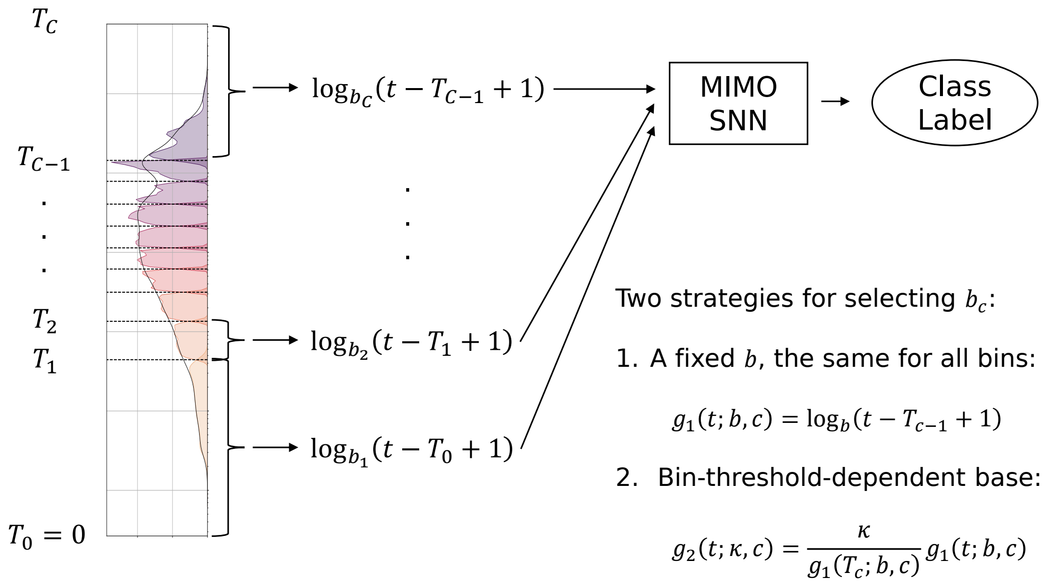
The difference between these two approaches lies in the range of values produced by the transform. Strategy $g_1(t; b, c)$ is unbounded from above and so it might be difficult to select a single b that works at both short and long timescales. Conversely, the function $g_2(t; \kappa, c)$ squeezes all values to lie in the range $[0, \kappa]$, which might make it easier for the network to learn the relationship between events at different timescales.

Figure 3.21 summarizes the outlined preprocessing steps. We found that applying the b-parameterized log-transform on binned sub-sequences sufficiently addressed our concerns for training the proposed spiking neural network. Note that while the log-transform (3.84) has the unfortunate effect of significantly increasing the latency of the model, we are only interested in the final classification prediction of the model and not its real-time performance.

3.3.2 SNN training objective

Using a MIMO SNN to propagate preprocessed tweet-retweet spike trains through the network makes the classifier training objective introduced in Section 3.1.4.2 not applicable. We can no longer operate on the assumption that only a single spike is ever elicited by any neuron in the network. Thus some adjustments are necessary, particularly to the spike regularization term.

Recall that training the network with a low spiking activity penalty term in (3.11) is crucial for ensuring event propagation through a single-spike network. This penalty term can be trivially extended to networks with inputs spiking over time by substituting the index over the input neurons c with an index over virtual input channels k. We can also apply regularization



T_C
T_{C-1}
T_2
T_1
$T_0 = 0$
$\log_{b_C}(t - T_{C-1} + 1)$
$\log_{b_2}(t - T_1 + 1)$
$\log_{b_1}(t - T_0 + 1)$
MIMO SNN
Class Label
Two strategies for selecting b_c:
1. A fixed b, the same for all bins:
$g_1(t; b, c) = \log_b(t - T_{c-1} + 1)$
2. Bin-threshold-dependent base:
$g_2(t; \kappa, c) = \frac{\kappa}{g_1(T_c; b, c)} g_1(t; b, c)$

Figure 3.21: A diagram of the preprocessing steps to obtain signals used to train a MIMO SNN on Twitter dataset.

to each neuron output spike m separately. Thus, the regularization term that considers the MIMO SNN signal propagation rules is

$$R_{\text{spiking}} = \sum_{h=1}^{H} \sum_{m=1}^{M_h} R_h \,, \tag{3.87}$$

where R_h is redefined as $R_h = \max\left(0, \frac{V_{\text{thr}}}{\tau_{\text{syn}}} - \sum_{k=1}^{K_h} w_{kh}\right)$. However, this implicitly assumes that every presynaptic weight w_{kh} is associated with an event t_{kh}. In Section 3.2.3 we showed that this need not be the case as the SNN can exhibit a sparse neuron activity. It is entirely possible that for some neuron h the inequality $\sum_{k=1}^{K_h} w_{kh} > \frac{V_{\text{thr}}}{\tau_{\text{syn}}}$ holds and yet the neurons does not fire anyway. This scenario might occur if none of the presynaptic neurons observe an event. Therefore, (3.87) can be reformulated to make this dependence on input spikes explicit

$$R_{\text{spiking}} = \sum_{h=1}^{H} \sum_{m=1}^{M_h} R_{mh} \,, \tag{3.88}$$

where $R_{mh} = \max\left(0, \frac{V_{\text{thr}}}{\tau_{\text{syn}}} - \sum_{k \in B_{mh}} w_{kh}\right)$ with $B_{mh} \subset Q_{mh}$ being the set of valid inputs for the m-th output of the h-th postsynaptic neuron

$$B_{mh} = \begin{cases} \{k : t_{kh} < \infty\} & \text{if } m = 1 \\ \{k : t_{\text{out}_h}^{[m-1]} + \tau_{\text{ref}} < t_{kh} < \infty\} & \text{otherwise} \end{cases} . \tag{3.89}$$

For completeness

$$R_{mh} = 0 \quad \text{if} \quad \{k : t_{kh} < \infty\} = \varnothing \,.$$

In our preliminary experiments we found that the penalty term is too strong for $m > 1$, skewing the training objective towards forcing neurons to output multiple spikes, rather than letting it focus on solving the actual task. As such, we train our models by setting $\forall_{m>1}\, R_{hm} = 0$, which only penalizes neurons that do not spike at all (alternatively one might consider applying a smaller weight to subsequent spikes).

Similarly to other classification tasks discussed in this Chapter, the modified cross-entropy loss (3.10) was used to ensure that the output neuron corresponding to the correct class fires first among all neurons of the last layer. Note that this training objective ignores the fact that each output layer neurons can produce multiple spikes. This means that for the last layer, either the refractory period could be set to $\tau_{\text{ref}} = \infty$, or that all $m > 1$ output spikes could simply be ignored (as they have no impact on the gradient propagation anyway). Both approaches lead to the same result.

Applying the dynamic scaling factor (3.67) to the spike regularization term computed over the set of valid inputs (3.12), and plugging the result into the loss function minimized by the original model (3.88) leads to the following loss function minimized by the MIMO SNN training objective

$$L_{\text{total}}(z, y) = \frac{1}{N}\sum_{n=1}^{N} L_n(z, y) + \gamma \frac{1}{\sum_{n=1}^{N} \mathbf{1}\left(y_n \neq \hat{y}_n\right)} \sum_{n=1}^{N} \mathbf{1}\left(y_n \neq \hat{y}_n\right) R_{\text{spiking}}^{[n]} \,. \tag{3.90}$$

The purpose of the dynamic scaling factor applied to the spike regularization term is to penalize low spike activity only when the predicted label does not match the ground truth, for reasons previously described in Section 3.2.3.

3.3.3 Training setup & results

In all of our experiments we used the same C-12-24-48-2 architecture, where C is the number of bins used to preprocess the spike train, the network has 2 output neurons, and other digits represent the number of neurons in the hidden layers. This produces a relatively small network of about $1536 + 12C$ parameters. However, a MIMO SNN model complexity is not only determined by the number of parameters, but also by the refractory period τ_{ref} (see (3.72)) which controls how often each neuron in the network is able to spike.

3.3.3.1 Impact of refractory period on the signal propagation

In order to settle on a single value of τ_{ref} to use in our experiments, we conducted a preliminary study by training the classification model on a small subset of data (stratified random 10% sample). We considered $\tau_{\text{ref}} \in [0.01, 1]$, selected on a 5-point logarithmically-spaced grid. The data was preprocessed into 30 bins, with bin-threshold-dependent logarithm base b_c

computed for $\kappa = 3$. All models were trained for 200 epochs of 10 update steps each, with a batch size of 64. This was enough to reach perfect classification score on the training set for each model (note that in this experiment we were not interested in the classifier's performance on unseen data, but rather in how the signal propagates through the MIMO network). The synaptic regularization parameter was set to $\gamma = 10^5$ in order to promote spiking activity in the network. Training was repeated 5 times in order to average-out processing time measurements.

In the preliminary study we measured the average time it took to finish the training epoch, as well as sparsity-adjusted average number of spikes produced by the network in response to the input signals. The "sparsity-adjusted" term states that only the active (i.e., not quiescent) neurons are considered when discussing the impact of the choice of τ_{ref}. To succinctly describe this measure we introduce the *network activity indicator*

$$\text{NAI} = \frac{1}{N \sum_{l=1}^{L} H_l} \sum_{n=1}^{N} \sum_{\ell=1}^{L} \sum_{h_\ell=1}^{H_\ell} M_{nh_\ell} \,, \tag{3.91}$$

where M_{nh_ℓ} is the number of output spikes generated by the h_ℓ-th neuron of the ℓ-th layer in response to the n-th example in the minibatch.

The results are summarized in Figure 3.22. We observe that both the NAI measure and the epoch training time rapidly increase in the initial stages (the first 50 epochs) because then the spike regularization term forces the model to learn how to propagate the signal through the network. Afterwards, the training time remains relatively constant throughout the rest of the training, while the NAI continues to increase, albeit at a much slower rate. Note that the growth of the NAI over training epochs is unbounded because there is no term in the loss function (3.90) that encourages the network to spike fewer times. This shows that setting a smaller τ_{ref} causes the network to produce more spikes.

However, surprisingly, the obtained processing time measurements do not support the notion that it is possible to predict which model will take the least amount of time to process the examples based solely on the value of the refractory period. Such relationship was anticipated as the number of iterations needed to compute all output spikes in (3.73) increases as τ_{ref} decreases. Perhaps significantly increasing the number of repetitions of this experiment would allow us to reach a conclusive answer (due to the uncertainty inherent to processing time measurements). Nevertheless, applying a min-max normalization to the processing time measurements separately for each experimental run shows that the relative increase in processing time over the training epochs is similar across different scenarios. Based on all these results, we settle on $\tau_{\text{ref}} = 0.1$ in our further experiments as the largest τ_{ref} that still exhibits a steep increase in the NAI measure over training steps.

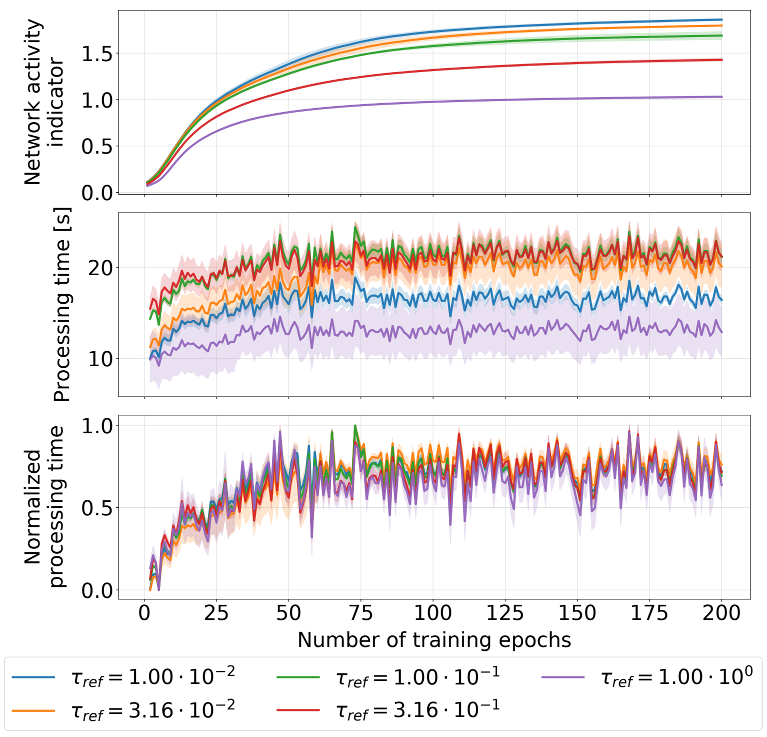


Figure 3.22: The impact of the refractory period τ_{ref} on a trained MIMO SNN properties. Top panel: sparsity-adjusted average number of spikes produced by the network. Middle panel: average epoch training time. Bottom panel: min-max normalized average epoch training time.

3.3.3.2 Bot detection – binary classifier performance

In our experiments on the classification problem, having fixed the value of τ_{ref}, we explored the impact of proposed preprocessing on model performance. We constructed a grid over preprocessing parameter space, selecting the number of bins from the set $C \in \{10, 20, 30, 40, 50\}$, whereas the log-transform parameter was set to either $b_c(\kappa)$ for $\kappa \in \{1, 2, 3\}$, or $b \in \{10, 30\}$. For every pair of parameters on this grid, several models were trained and evaluated with a 5-fold cross validation. Thus, in total 125 models were trained. All models were trained for 50 epochs, each with 100 training steps over 64 class-balanced training examples. The learning rate was set to 10^{-3} in all steps. The synaptic regularization term factor γ varied substantially depending on the current training epoch – initially set to 10^5 in order to guide the model towards a state in which it is able to propagate spikes throughout all layers. After 10 epochs, we set $\gamma = 10^{-2}$ so that the model could focus on minimizing the task-specific loss term (3.10).

Given the small dataset size (366 legitimate users and 389 bots), we opt to use data augmentation in order to increase the effective training split size. Two types of augmentation were used:

- drop events with probability 0.1,

- randomly shift each event in time by t_δ uniformly distributed in $(-0.05, 0.05)$, independently with probability 0.3.

The latter augmentation type is applied only after the sequence has been preprocessed, making sure that the shift-augmented sub-sequence is still composed of events occurring at nonnegative time.

The classification accuracy achieved by all models trained over the preprocessing grid is summarized in Table 3.3. The results obtained during the hyperparameter search on the preprocessing grid do not suggest an existence of some pattern that holds across different number of bins or the transform choice. Notably, however, the setting denoted by the parameter $b = 30$ seems to be more robust compared to other settings in that it is the only one that allowed the model to consistently reach an accuracy of about 70% or higher regardless of the number of bins.

Overall, the wide range of classification scores reported by different models suggest that it is imperative to perform hyperparameter tuning when using the proposed MIMO network. Nevertheless, it must be stressed that the hyperparameters associated with the neural model itself (V_{thr}, τ_{syn}, τ_{ref}) can be selected heuristically. Firstly, note that V_{thr} only influences the scale of trained weights and has no impact on training dynamics, making the choice of V_{thr} arbitrary. On the other hand, the time constants τ_{syn} and τ_{ref} can be selected according to the input event distribution. The post-synaptic potential constant τ_{syn} needs to be longer than the relevant temporal patterns present in the input data [130]. Lastly, setting $\tau_{\text{ref}} < \tau_{\text{syn}}$ prevents the scenario in which the network rarely generates events.

Table 3.4 compares our best model with the results obtained in the original study [5] in terms of accuracy, recall, precision, F_1-score and Matthews correlation coefficient (MCC). Our MIMO SNN model outperforms all of the presented supervised approaches (Botometer, Social fingerprinting), as well as most of the unsupervised methods (HoloScope; hand-crafted-, PCA- and TICA-based RT_{BUST}). Importantly, the latter group of methods relied on fitting the model on the entire unlabeled dataset of 63,762 accounts and evaluating on the labeled portion, whereas we focus only on the annotated subset (as outlined in Section 2.5.2), composed of about 755 labeled cases in total. We note that the variational autoencoder (VAE) variant of the RT_{BUST} model performed better than our approach. As this model also leveraged the unlabeled portion of the dataset, it stands to reason that there is a clear benefit to clustering-based methods, in which the presence of a suspicious behavior emerges only when analyzing user in groups, rather then as individuals. Crucially, the examples examined in [5] were manually labeled and thus any incorrectly labeled cases could have had a much bigger impact on the supervised model trained with significantly fewer examples. Lastly, it is important to note that the original study lacks technical details related to the LSTM-VAE network architecture, which prevents us

Table 3.3: Classification performance of the MIMO SNN models trained on Twitter tweet-retweet dataset, depending on the chosen preprocessing parameters. A star ($\star$) denotes the best result for a row, whereas the diamond ($\diamond$) denotes the best result in a column.

(a) accuracy [%]

Stratified 5-fold cross validation		**Log-transform-related parameter**				
		$b = 10$	$b = 30$	$b_c(\kappa = 1)$	$b_c(\kappa = 2)$	$b_c(\kappa = 3)$
Number of bins	$C = 10$	$^{\star\diamond}73.25 \pm 3.71$	70.33 ± 3.83	68.74 ± 3.09	68.08 ± 4.20	68.61 ± 10.46
	$C = 20$	70.20 ± 3.72	$^{\star}70.60 \pm 1.65$	67.95 ± 2.70	70.33 ± 3.80	68.21 ± 5.70
	$C = 30$	67.42 ± 2.03	$^{\star\diamond}71.52 \pm 4.74$	69.40 ± 4.76	70.86 ± 1.83	69.14 ± 3.09
	$C = 40$	69.27 ± 2.67	70.60 ± 4.40	66.75 ± 4.50	69.40 ± 4.42	$^{\star\diamond}71.13 \pm 3.49$
	$C = 50$	67.68 ± 4.20	71.26 ± 6.22	$^{\star\diamond}72.19 \pm 3.82$	$^{\diamond}70.99 \pm 4.26$	68.08 ± 3.61

(b) F_1-score [%]

Stratified 5-fold cross validation		**Log-transform-related parameter**				
		$b = 10$	$b = 30$	$b_c(\kappa = 1)$	$b_c(\kappa = 2)$	$b_c(\kappa = 3)$
Number of bins	$C = 10$	$^{\star\diamond}72.46 \pm 5.08$	67.10 ± 7.04	68.43 ± 3.90	68.71 ± 3.90	57.78 ± 29.15
	$C = 20$	69.59 ± 3.28	68.17 ± 4.59	68.16 ± 3.03	$^{\star}69.96 \pm 5.11$	65.08 ± 7.61
	$C = 30$	64.74 ± 6.75	$^{\star}70.21 \pm 5.75$	67.33 ± 7.21	$^{\diamond}70.10 \pm 2.64$	65.89 ± 2.90
	$C = 40$	66.80 ± 5.14	68.28 ± 5.40	64.43 ± 6.55	66.18 ± 6.09	$^{\star\diamond}69.07 \pm 3.64$
	$C = 50$	68.63 ± 4.31	$^{\diamond}70.25 \pm 5.60$	$^{\star\diamond}71.25 \pm 5.10$	69.89 ± 5.81	67.23 ± 4.64

Table 3.4: Comparison of model performance on the bot detection task between different techniques.

Source	**Technique**	**Accuracy [%]**	**Recall [%]**	**Precision [%]**	**F_1-score [%]**	**MCC**
[5]	Botometer	58.30	30.98	69.51	42.86	0.2051
	HoloScope	49.08	0.49	28.57	0.96	−0.0410
	Social fingerprinting	71.14	89.78	65.62	75.82	0.4536
[5]	RT_{BUST} (handcrafted features)	53.64	77.07	52.84	62.70	0.0767
	RT_{BUST} (PCA)	51.54	95.12	51.11	66.49	0.0446
	RT_{BUST} (TICA)	53.64	95.12	52.28	67.47	0.1168
	RT_{BUST} (VAE)	87.55	81.46	93.04	86.87	0.7572
our	MIMO SNN	73.25 ± 3.71	69.56 ± 8.93	76.39 ± 3.34	72.46 ± 5.08	0.47 ± 0.07

from making a comparison between the two methods that is adjusted for model complexity as expressed by the number of trainable parameters.

3.3.3.3 Model ablation study

For the best-performing model, we ran an ablation study experiment in order to determine which components of the proposed approach have a significant impact on model performance. We tested four scenarios:

1) no data augmentation,
2) setting $\tau_{\text{ref}} = \infty$, i.e., preventing the neurons in the network from spiking more than once in response to a given input signal,
3) no splitting of channels into separate bins,
4) no log transform.

Each one represents a single change to the experimental protocol outlined in Section 3.3.3.2. Clearly, with the exception of the first scenario, these changes have a major impact on how the signal is propagated through the network or on training dynamics.

As evidenced in Table 3.5, removing any of the components causes a reduction in the model performance. This shows that the proposed data augmentation scheme is effective in mitigating the problem of overfitting. Furthermore, the drop in performance for the $\tau_{\text{ref}} = \infty$ scenario suggests that some of the model's capacity to process information is tied to the ability to process it over time. Unsurprisingly, not splitting the input spike train into multiple bins makes it more difficult for the model to learn long range dependencies, which is an effect that was anticipated while designing this preprocessing step. Lastly, the steepest drop in performance is observed when the data is passed through the network without any transform that squashes the range of values at its input. Note that the obtained average accuracy of 47.68% is quite close to the ratio of the number of legitimate users to all users in the stratified evaluation data split (48.48%). This means that the model was unable to learn anything, most likely because using the raw event times as large as $2 \cdot 10^4$ minutes transformed according to (3.6) surpasses the limits of double-precision floating-point data format, making training impossible. Note that the trained models are in general biased towards higher precision, with the exception of the channel-splitting scenario, despite being trained with a nearly class-balanced data. In bot detection systems it is preferable to favor precision instead of recall as the system administrator should be reasonably certain that a user is a bot before taking any action.

For completeness, Figure 3.23 shows representative examples of confusion matrices for the five scenarios, chosen according to the MCC. It is evident in the results for the experiment without log transform that for a failed model all predictions are towards the negative class. This stems from the assumption that if the output layer produces no events, then the model should return the negative class prediction. However, this effect is negligible for properly trained models – in other scenarios less than 1% of bot accounts were misclassified as legitimate users due to this assumption.

Table 3.5: The SNN model performance in the ablation study, given the specified scenario.

Scenario	Accuracy [%]	Recall [%]	Precision [%]	F_1-score [%]	MCC
baseline	73.25 ± 3.71	69.56 ± 8.93	76.39 ± 3.34	72.46 ± 5.08	0.47 ± 0.07
no data augmentation	66.89 ± 1.45	58.09 ± 10.61	73.77 ± 6.50	63.80 ± 5.60	0.36 ± 0.03
infinite refractory period	67.28 ± 2.64	56.30 ± 4.52	74.44 ± 5.63	63.89 ± 3.06	0.36 ± 0.06
no channel-splitting	63.18 ± 4.60	69.67 ± 14.22	62.61 ± 2.74	65.42 ± 7.49	0.27 ± 0.10
no log transform	47.68 ± 1.18	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	−0.06 ± 0.07

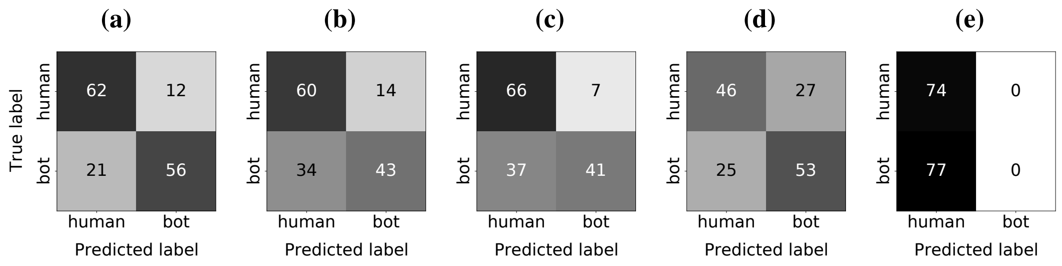


Figure 3.23: Confusion matrices corresponding to the best-performing iteration with respect to the MCC value for five ablation experiment scenarios: (a) baseline, (b) no data augmentation, (c) infinite refractory period, (d) no channel splitting, (e) no log transform.

3.4 Summary

This Chapter explored the topic of spiking neural networks which combine recent advances in deep learning with the ability to process event data. The SNN more closely model biological neurons, including their ability to process event streams similarly to biological networks by inducing a neural coding scheme using a corresponding training loss function. Among the various SNN types we focused on the time-coding single-spike time-to-first-spike SNN. They rely on a set of equations that determine when each postsynaptic neuron elicits its first spike. Such network can be trained end-to-end directly in the spiking domain.

We conducted a reproducibility study of the paper that introduced this specific SNN type. This allowed us to assess the correctness of our own implementation. Furthermore, those results could be used as a reference point when discussing the model limitations and shortcomings. Specifically, we found that the naïve algorithm is computationally expensive, making it unfeasible to train deeper networks. Furthermore, it was observed that the SNN exhibits the time-invariance property, i.e., shifting the input spikes in time causes the model to respond in the exact same way, just at a different time. This has a net negative impact on the training dynamics. Lastly, we verified that the trained model is able to elicit an output spike as soon as the signal is propagated through the network, even if some of the neurons in the hidden layers have yet to fire, making these hidden layer neurons' future spikes redundant. Our contributions to the SNN research described in this Chapter stem from this set of initial observations.

We were able to significantly reduce the time it takes to compute the layer outputs by vectorizing the computation (i.e., computing all outputs at once in a single pass over data). This requires finding all possible inputs combinations and returning the ones that produce the earliest output spike, rather than relying on the iterative search for a valid combination for each neuron separately. The proposed algorithm is three orders of magnitude faster than the baseline. This makes it feasible to train the SNN in modern deep learning frameworks.

To address the adverse effect of the absolute time reference on model training dynamics a time-aligned variant of the model was proposed. It ensures that all layers of the network observe events starting at a relative time $t = 0$. The input events are shifted in time before the layer computation and then the resulting layer output events are shifted to the original time reference. We verified that training a time-aligned SNN results in the exact same set of final network weights, regardless of the magnitude of the absolute time reference.

We noticed that the spike-firing penalty, which promotes the SNN activity during training, is extremely important during the initial training iterations, with its relative importance diminishing as training progresses. Only after the model is able to produce spikes at its output does the task-specific loss function component begin to play any role in the training process. As such, we introduced a modified regularization term that dynamically scales the penalty, applying it only when the training task is not solved correctly. We found that this change results in models that use fewer neurons to process examples, avoiding spike redundancy. This neural activity sparsity is context-based, i.e., a different subset of neurons will respond to each input signal.

The last of the proposed modifications to the baseline time-to-first-spike SNN model is to relax the requirement of an infinitely long neuron refractory period τ_{ref}. Doing so means that neurons in the model are no longer limited to eliciting at most one spike during a single input example presentation. Once trained, the model can be realized on existing neuromorphic devices that implement the IF neuron computation. The proposed MIMO SNN expresses the entire algorithm in terms of iteratively calculating successive spikes, which stands in contrast to other works simulating the state of the entire network over a finite time window with a fixed time step. The MIMO SNN is suited towards datasets with spike trains composed of relatively infrequent events occurring at different timescales. Furthermore, the MIMO SNN computation is compatible with the proposed vectorized approach for a single-spike layer, meaning that model training can be implemented in modern deep learning frameworks, making the approach scalable to large volumes of data.

The model was applied to the labeled subset of Twitter user activity data that was introduced in Section 2.5 to evaluate the kernel-based classifier. Our best model achieved an accuracy score of 73.25%, compared to 87.55% obtained by the original RT_{BUST} study. However, we note that the latter is an unsupervised learning algorithm and is therefore able to infer different non-overlapping patterns of activity of distinct groups of users not present in the class-label-aggregated data. Furthermore, our model relies on about 755 labeled example compared to the 63,762 unlabeled cases used to train $\mathrm{RT}_{\mathrm{BUST}}$. As the data was manually labeled by the authors of the study, any incorrectly labeled examples could have had a much bigger impact on the supervised model trained with significantly fewer examples. We have shown that the proposed MIMO SNN operates directly in the event-domain, and so there is no need to encode the time series in any way for it to be processed by the model. In addition to the classification model feasibility study, this work showcased novel signal preprocessing steps, exemplary spike train data augmentation techniques, and the heuristic of modifying regularization scale factor during training to tackle this challenging dataset. We found that these concepts were critical at preventing overfitting and stabilizing the training procedure, as evidenced by the results of the ablation study.

Finally, compared to the kernel-based classifier analyzed in Section 2.5.4, we observe that the SNN performed better on the Twitter bot detection task than the alternative in terms of the accuracy and F_1-score. It is entirely possible that the observed difference in performance stems from a more exhaustive analysis of the problem through the lens of the SNN framework, or that this model is simply better suited towards such a peculiar dataset. Regardless of the reason, it might be more beneficial to compare the properties and applicability scope of different models rather than their performance on specific tasks. We note that both approaches will struggle with updating their predictions as new events are observed. For the point process kernel classifier the shape function estimate must be recomputed (as the kernel bandwidth controls which part of the estimate changes) and it is unclear how to select the bandwidth when event observation window increases. On the other hand, we have demonstrated that the arrival of a new event might change the causal set, impacting the signal propagation in the SNN. This also implies that the network prediction must be recomputed on each new event. It might be beneficial to explore modifications that address this issue for both algorithms in further research. Overall, the SNN, while more difficult to tune, has the advantage of scaling better with the training data volume. Additionally, the SNN classifier can be used for multi-class problems without relying on One-vs-All meta-heuristics, and does not require that the signal is present in only one channel. And so, moving forward, we shall focus on the SNN-based solution.

An important limitation that remains unaddressed is that once excited by an input spike, the membrane voltage of an IF neuron resets to its resting potential only after eliciting an output spike. This means that such SNN cannot be used for continuous monitoring of long-running processes because the state of the entire network is not reset in the absence of events. Therefore, the model would need to incorporate a voltage decay constant, such as the one present in a

Leaky Integrate-and-Fire (LIF) neuron. Exploring this topic in the future is bound to extend the applicability of the model. Furthermore, it might be interesting to revisit the vectorized implementation of the spiking layer introduced in Section 3.2.1. Specifically, the current version of the algorithm operates on third-order tensors of fixed size. Non-event indicators present in each tensor contribute to the overall computation while having no impact on the generated spike. One possible approach would be to formulate computation rules in terms of sparse matrices so that only the informative events are considered.

Chapter 4

Siamese Spiking Neural Network

Quantifying the similarity between two objects is the basis of similarity learning – an area of supervised machine learning with applications to ranking, recommendation engines, tracking, and identity verification. In contrast to classification techniques, the supervisory information is related to the concept of "sameness" or "difference" between two objects and not their actual class label identity. In fact, a weaker form of supervision that establishes a relative degree of similarity is also acceptable. When data is abundant, and their relative similarity is known, it is possible to train a so-called Siamese network – a neural network that maps objects in the similarity space. Siamese neural networks have established themselves as a versatile class of deep learning models, achieving state-of-the-art results on tasks such as object tracking, information retrieval, and change-point detection. And yet, so far, little attention has been paid to applying these models in the context of spiking neural networks operating directly in the event-data domain. Adapting the Siamese model in the SNN context necessitates leveraging existing spike train similarity measures developed by the neuroscientific community. In neuroscientific research determining spike train similarity between pairs of event streams is commonly used to identify groups of similar neural ensembles or to find causality between the observed spike train and source stimuli. Importantly, the similarity measure chosen to analyze the spike trains must be selected in accordance with the neural coding scheme observed in the data.

The goal of this Chapter is to propose a training scheme for a Siamese SNN that is compatible with the time-to-first-spike SNN established in the previous Chapter. Section 4.1 presents a brief overview of spike train similarity measures commonly used in neuroscientific research, and introduces Siamese neural networks in a nonspiking context. Section 4.2 is focused on establishing the SNN training objective for a selected spike train similarity measure. Next, the proposed methodology is evaluated on two image datasets. In Section 4.3 the MNIST digit dataset is converted into the spiking domain with novel conversion schemes, and the quality of the Siamese embeddings of input images is evaluated by measuring the classifier performance while varying the number of output neurons. This study also evaluates the model in terms of sparse spiking activity and prediction latency. Lastly, the proposed Siamese SNN model is applied to the problem of differentiating signal from artefacts in the context of cosmic ray detection using images taken by modern smartphones. Section 4.4 introduces the CREDO experiment and briefly summarizes the image acquisition. Similarly to the MNIST study, the results are also discussed in terms of sparse spiking activity and prediction latency, additionally highlighting a simple heuristic that stabilizes the training dynamics.

4.1 Introduction

4.1.1 Spike train similarity

Spike train similarity measures are an important tool in analyzing the relationship between pairs of event streams measured in biological neural networks [105]. Computing such measures is an essential step in the spike train analysis that identifies groups of similar neural ensembles or determines causality between observed spike train and source stimuli. Commonly used similarity measures include:

- Victor-Purpura (VP) distance [37]: the cost of transforming one spike train into the other (by shifting, deleting and inserting events),
- van Rossum (vR) distance [167]: squared Euclidean distance between kernel-smoothed spike trains,
- Schreiber dissimilarity measure [168]: correlation between kernel-smoothed spike trains.

For the latter two similarity measures, the first step is to apply a kernel-smoothing transform to a discrete event stream [169] in order to obtain a continuous function representation of the underlying process. Both van Rossum and Schreiber similarity measures can be elegantly described in terms of the reproducing kernel Hilbert space (RKHS). Paiva et al. [170] derive an RKHS kernel in terms of process intensity and introduce a memoryless cross-intensity kernel that can be used to cluster spike trains. In this space they relate the van Rossum distance to the squared Euclidean norm, while Schreiber similarity is described in terms of the Cauchy-Schwarz distance (a measure introduced by the authors based on the Cauchy-Schwarz inequality).

Importantly, the choice of a spike train similarity measure is heavily dependent on the chosen neural coding scheme [171], which describes the mechanisms of information transfer between neurons. Notably, no one measure is clearly better than the others. The vR distance is better at tasks where rate-coding dominates, while the other two are better at describing precise spike timing (synchrony), although they are unstable for spike trains with a small number of spikes [172]. The result presented by Chicharro, Kreuz & Andrzejak [173] suggests that the choice of “optimal” time-scale parameter of these metrics is arbitrary and does not correspond to biologically meaningful time scales. A more recent study [171] suggests that other metrics such as the ISI- [174], SPIKE- [175] and RI-SPIKE distance [176] are better at resolving timing and synchrony coding, provided that spike trains contain a reasonably large number of events.

Experimental research strongly suggests that neuron spike trains are autocorrelated, with the preceding spike having the largest impact on the current event [107]. Short interspike intervals tend to be followed by long intervals and vice versa. This effectively reduces the spike count variability, which implies a lower noise level for a rate-coded signal, improving signal detectability in noisy conditions [106]. Moreover, this allows differentiating between spike trains that exhibit the same impulse response and have the same empirical ISI distribution [104]. This suggests that an effective spike train similarity measure should take into account the relative timing of spike events.

A recent study conducted by Sihn & Kim [177] indicates that that the Earth Mover’s Distance (EMD) is superior to other spike similarity metrics with respect to measuring the timing-sensitive coding because it depends only on the precise timing of events and is insensitive to changes of the mean rate of firing. The EMD is a metric similar to the Victor-Purpura distance in that it also minimizes the cost of transforming one spike train to another. Unlike the VP distance, however, it only considers shifts (i.e., the EMD is equivalent to the VP distance in the limit of prohibitively expensive event insertion or deletion) and operates on fractional values assigned to each spike. It is also worth noting that the EMD is a special case of the general class of the so-called Wasserstein metric.

4.1.2 Siamese neural networks

Siamese neural networks are a type of machine learning models trained to optimize a similarity measure between network outputs for different examples of data from some domain [178, 179]. More broadly it can refer to any neural network architecture that compares two or more input data points passed through identical sub-networks (such as network layers sharing parameters) [180] (Figure 4.1a). While the concept of querying inputs against a fixed database based on their similarity (i.e., given some input signal, find the most similar set of signals from the database) is not novel [181], the Siamese network has the advantage of operating on raw signals, skipping the domain-specific feature-engineering step of the model design. It has been observed that the feature space optimized by Siamese networks – or the *embeddings* space – can be used to query instances of entities that were not shown during training [182]. In the context of classification, this concept (called the one-shot learning) means training a model to recognize classes that were never present in the training set [183] (Figure 4.1b).

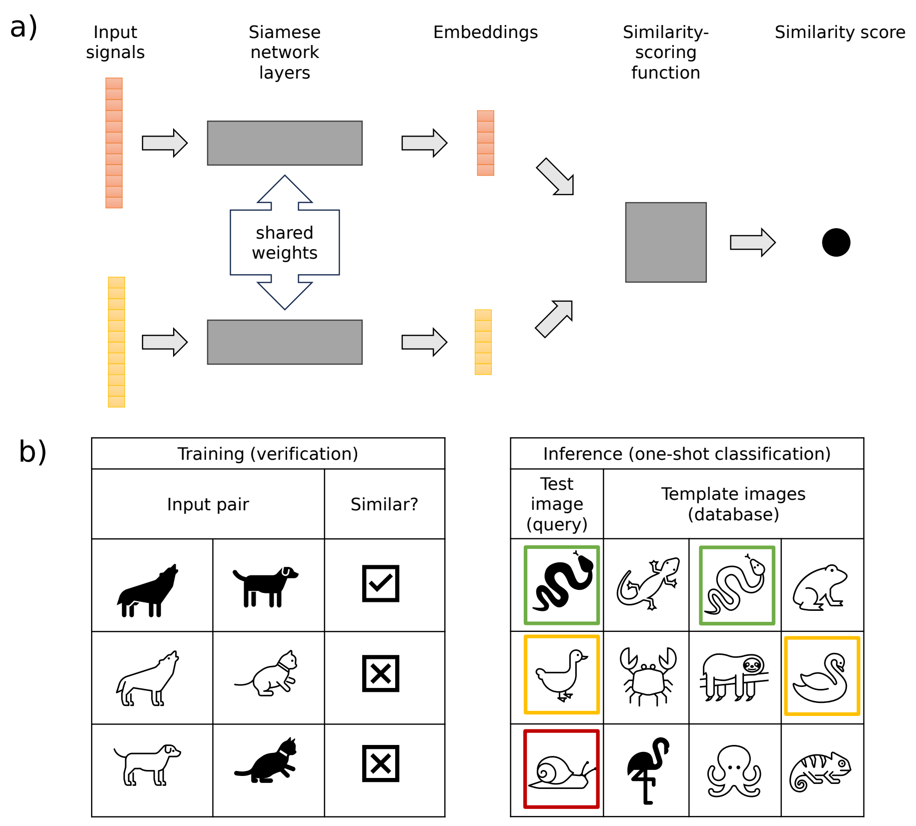


Figure 4.1: a) General structure of a Siamese neural network. The similarity-scoring function can be a separate piece of computation, or an additional set of network layers. b) One-shot classification with a Siamese neural network. During training the network learns to verify whether a given pair of images represents similar entities (based on known ground truth labels). During inference the network is shown examples of previously unseen classes. If the model is capable of generalization, it will still be able to determine whether a given test image matches an image from the set of templates. Note that it is possible to set a decision threshold such that no template image matches a given query.

Like most neural network models, the performance of Siamese neural networks depends on both the quality of data (raw signal, labels), as well as on a set of hyperparameters related to the optimized embedding space. This latter group is composed of model design decisions that influence the feature space dimensionality, distance metric, and the loss function. Even though the nonlinear data transform parameterized by the model results in data that is of lower dimensionality than the input, the embedding space might still suffer from the so-called *curse of dimensionality* (where points in high-dimensional vector space are approximately equidistant) [184]. Conversely, the choice of an optimized distance metric seems to be arbitrary (i.e., “whatever works best”) and is usually selected from a pool of simple metrics that are applicable to the given input data domain. For vector data the Manhattan distance, Euclidean distance or cosine similarity are commonly used.[a]

[a]Cosine similarity is actually a semi-metric, but in context of Siamese neural networks this distinction is a mere

In terms of the loss function definition one can differentiate two types of models, based on how many examples are needed to compute the loss. For brevity, let us denote f_x as the embedding computed by a trained network for an input example x. The contrastive loss between a pair f_a, f_b is [185]

$$L_{ab} = \frac{y}{2}\left[d\left(f_a, f_b\right)\right]^2 + \frac{1-y}{2}\left[\max\left(0, \alpha - d\left(f_a, f_b\right)\right)\right]^2 , \tag{4.1}$$

where y represents the known similarity relationship between the pair ($y = 1$ when f_a and f_b are similar, $y = 0$ otherwise), α is a margin, and $d\left(\cdot\right)$ is a distance function between the two embeddings. This optimization criterion forces the computed pair of embeddings to be close to one another in the feature space, while making the embeddings of dissimilar pairs distant by at least the margin α (Figure 4.2a). It is clear that the presence of the contrastive term in the loss function prevents a trivial solution in which the network computes the same embedding for all examples [186]. In general, models returning the same embedding for different input examples exhibit an *embedding collapse*. Note that the contrastive loss is susceptible to a partial embedding collapse as the first term is nonzero unless the embeddings are identical.

A different type of a pairwise loss function to train Siamese networks is the binary cross-entropy [183]

$$L_{ab} = -y \ln\left(p\left(f_a, f_b\right)\right) - \left(1 - y\right) \ln\left(1 - p\left(f_a, f_b\right)\right) , \tag{4.2}$$

where $p\left(\cdot\right)$ is a scoring function that returns the probability of the two arguments being similar to one another. Similarly to the contrastive loss, it forces the network to return a high similarity score for a pair of embeddings computed for similar examples, and low score otherwise. The difference between the two loss functions is subtle but important. The binary cross-entropy loss trains the network to predict the relationship between a pair of inputs, whereas the contrastive loss makes the model differentiate between its inputs. Note that either of these two functions can be realized by the model presented in Figure 4.1a by properly choosing the activation function for the final layer with a single neuron. Furthermore, the function $p\left(\cdot\right)$ either takes into account a predefined distance measure between the two embeddings, or computes a learnable metric using its layers.

The triplet loss differs from the previously mentioned loss functions in that it compares three examples at once [182]. Let f_a, f_p, f_n be the embeddings computed by the model for examples a, p, n called the *anchor*, *positive*, and *negative*, respectively. Then, the formula

$$L_{apn} = \max\left(0, \alpha + d(f_a, f_p) - d(f_a, f_n)\right) , \tag{4.3}$$

describes the contribution of a given triplet to the total loss. Minimizing L_{apn} ensures that the distance computed by the function $d(\cdot)$ between f_a and f_p (or the anchor-positive pair of

technicality.

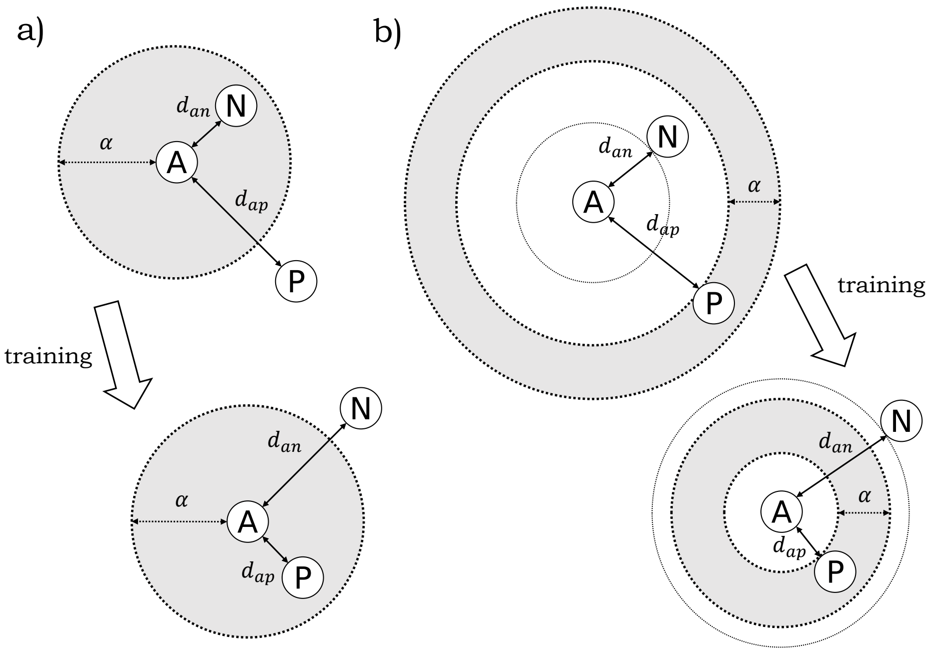


Figure 4.2: The impact of the loss function on the embedding space produced by the Siamese network. d_{ap} denotes the distance between embeddings of the anchor-positive pair, d_{an} is the distance between embeddings for the anchor-negative pair, and α is the margin. a) Contrastive loss (margin forms a hypersphere around the anchor embedding). b) Triplet loss (margin determines the thickness of the shell around the anchor-positive pair in the hyperdimensional space). In both scenarios the model is trained to push the embedding of the negative example outside the volume defined by the margin.

examples) is smaller than the distance between f_a and f_n (the anchor-negative pair) by at least some margin α. This means that the loss can be minimized by either pushing the negative embeddings away from the anchor, or by pulling the positive embedding closer (Figure 4.2b). This push-pull mechanism makes the triplet loss less susceptible to an embedding collapse than the contrastive loss, although it is more computationally expensive. Note that the binary cross-entropy loss can also be adapted to the triplet scenario [187].

Perhaps uniquely, Siamese networks are much more vulnerable to training data sampling than other deep learning models, due to interactions between same- and different-class pairs (triplets) [188]. As neural networks are trained with minibatches, it is important for batches to contain examples of different classes. Moreover, after some number of training iterations, most pairs (triplets) of examples will contribute close to zero loss, and computing loss for these examples will simply waste computational resources [189]. It is therefore important to select difficult examples in order to achieve model convergence in a reasonable time frame.

Siamese networks have achieved state-of-the-art results on forgery detection [190], person re-identification [182], as well as object- [191, 192] and person-tracking tasks [189]. Furthermore, these networks reach competitive performance in change point detection [193, 194] and can even be used to design ranking engines [195, 196, 187]. These examples support the notion that the Siamese neural network is a versatile class of deep learning models, capable of solving a wide variety of tasks.

4.2 Siamese SNN training objective

Training a spiking neural network such that the output spike train matches a predetermined temporal pattern based on some similarity measure has been studied extensively. Several early studies, such as the ReSuMe [128] or the Tempotron [116], focused on single-layered networks. More recent works, however, emphasize the ability to train a multilayer SNN. For example, in [197] a general supervised learning rule that minimizes an L_2-distance between kernel-smoothed spike trains is proposed; Zenke & Ganguli [141] optimize the van Rossum distance between spike trains; whereas Xing et al. [198] train a network by minimizing the spike count differences in predetermined time windows over all neurons in the output layer.

However, to train a Siamese model we require an objective that scores the relative similarity of multiple output spike trains and not their absolute similarity to some predetermined pattern. To the best of our knowledge, the only work directly related to adapting the Siamese model to the SNN is by Luo et al. [199] which describes a Siamese spiking neural network obtained by converting an existing convolutional neural network model to the spiking domain. They measure the spike train similarity using a variant of the SPIKE-distance [200] and SPIKE-synchronization [201] measures. Their model achieved competitive performance on several visual object tracking benchmarks with low precision loss with respect to the original nonspiking network.

In contrast to the work of Luo et al. [199], we propose a Siamese SNN model that is optimized in the spiking domain, rather than be a product of converting an existing neural network to the spiking domain. Additionally, we opt to encode information using single events instead of bursts of spiking activity, making this model suitable for time-coding imposed by the SNN type introduced in Chapter 3.

Generally speaking, the loss function to train the proposed time-coded Siamese SNN is a composite of the triplet loss (4.3) averaged over a set of valid triplets in the minibatch Q (the “batch-all” strategy described by Hermans et al. [189]), and the modified spike regularization term R^*_{spiking} (3.67) which dynamically scales the spike-firing penalty depending on how well the network solves the task in the current iteration

$$\begin{gathered} L^d_{\text{total}} = \frac{1}{|Q|} \sum_{\{a,p,n\} \subset Q} L^d_{apn} + \gamma R^*_{\text{spiking}} \\ Q = \{a, p, n : a \neq p \neq n,\ y_a = y_p \neq y_n\} \end{gathered} , \tag{4.4}$$

where y_a, y_p, y_n are the class labels of examples a, p, n; γ is a hyperparameter; and d indicates the spike train distance function. In order to define the modified spike regularization term R^*_{spiking} for the Siamese SNN, let us first introduce $Q_a \subset Q$ as the set of all triplets associated with the anchor a

$$Q_a = \{p, n : a \neq p \neq n,\ y_a = y_p \neq y_n\}\,. \tag{4.5}$$

Additionally, let

$$Q_{\mathrm{AT}a} = \{p, n \subseteq Q_a : L^d_{apn} \neq 0\} \tag{4.6}$$

be the set of active triplets (i.e., triplets which contribute to a nonzero loss) for the anchor a. Lastly, define

$$\mathrm{AT}_a = \frac{|Q_{\mathrm{AT}a}|}{|Q_a|} \tag{4.7}$$

as the ratio of active triplets for anchor a. Then,

$$\mathrm{AT} = \frac{1}{|U|}\sum_{a\in U} \mathrm{AT}_a \tag{4.8}$$

is the average ratio of active triplets computed for the current batch of examples composed of anchors U. The empirical measure AT is used as an early stopping criterion for the training procedure. Finally, the relaxed spike regularization term for the proposed Siamese SNN is

$$R^*_{\text{spiking}} = \begin{cases} 0 & \text{if} \quad Q = \varnothing \\ \frac{1}{|U|\cdot \mathrm{AT}} \sum_{a\in U} \mathrm{AT}_a \cdot R^{[a]}_{\text{spiking}} & \text{otherwise} \end{cases}, \tag{4.9}$$

where $R^{[a]}_{\text{spiking}}$ is the spike-firing penalty term in (3.11) computed only for the given anchor a. Note that the spike-firing penalty is not applied to the anchor a if all of its triplets are correctly distributed in the embedding space. This is analogous to conditioning R^*_{spiking} based on the correct classifier prediction for a classifier SNN (3.67).

The proposed training objective is applicable to any bounded pairwise distance function d. Throughout this Chapter we use the Earth Mover's Distance as the distance function to measure the similarity of spike trains produced by the proposed Siamese SNN. In doing so we rely on the results in [177] stating that the EMD is superior to other spike similarity metrics when biological neurons rely on precise timing to process information. Additionally, it has a low computational complexity and is parameter-free. Conversely, in Chapter 5 we use the van Rossum distance to select the parameters of signal-to-spike encoding schemes for multivariate time series data. In contrast to the EMD, this measure is able to compute similarity between populations of neurons rather than only between pairs of spike trains. Therefore, substituting the EMD with the van Rossum distance in the Siamese SNN training objective would be a fairly straightforward modification of the model.

4.2.1 Earth Mover's Distance

A *spike train embedding* $f(t)$ of a spike train composed of events $\{t_i\}$ is

$$f(t) = \frac{1}{P}\sum_{i=1}^{P}\delta(t - t_i)\,, \tag{4.10}$$

where $\delta(t)$ is a Dirac delta function and P denotes the total number of spikes in the spike train. Given another spike train embedding $g(t) = \frac{1}{R}\sum_{i=1}^{R}\delta(t - \tau_i)$, the EMD between $f(t)$ and $g(t)$ takes the following form

$$\mathrm{EMD}(f, g) = \int_{-\infty}^{\infty} |F(t) - G(t)|\, dt\,, \tag{4.11}$$

where F, G are cumulative distribution functions of embeddings f, g, respectively. The function F (G is defined analogously) is the normalized counting function (2.1) of the corresponding spike train and takes the form $F(t) = \frac{1}{P}\sum_{i=1}^{P} u(t - t_i)$, where $u(t)$ is the step function. It is worth noting that the $\mathrm{EMD}(f, g)$ is a special case of the Wasserstein metric corresponding to the L_1 norm. In this case it can be shown [202] that the Wasserstein metric can be considerably simplified and represented by the formula in (4.11). The Wasserstein metric has very deep geometrical properties as it is connected to the theory of optimal transport [202]. Lastly, the distributions F, G are piecewise constant nondecreasing functions, therefore the numerical evaluation of (4.11) is straightforward [177]. The overall computational complexity of the EMD for a pair of spike trains $f(t)$ and $g(t)$, with P and R events respectively, is $\mathcal{O}\,(P + R)$, assuming that events $\{t_i, i = 1, \ldots, P\}$ and $\{\tau_i, i = 1, \ldots, R\}$ are sorted [203]. Figure 4.3 summarizes the computation of the EMD for a pair of spike trains.

In general, the EMD is set to zero if both sequences under comparison have no events, and is set to infinity if one of the sequences is empty while the other is not. In any other case the EMD is finite. This property violates the requirement stated in Section 4.2 that the distance function used to train the Siamese SNN must be bounded. However, it can be utilized in the training objective if all neurons in the output layer of the network generate at most one spike (note that the SNN can be composed of the MIMO hidden layers). In this scenario, the output spike train is constructed from concatenating events of all neurons and sorting them in an ascending order. It can be argued that for an empty output spike train the optimization algorithm should focus on generating at least one output spike by minimizing the spike-firing penalty R_{spiking} instead of the ill-defined task-specific loss, because the latter may be infinitely large. This leads to the following definition of the EMD used in our Siamese SNN training objective

$$\mathrm{EMD}^*(f, g; P, R) = \begin{cases} \int_{-\infty}^{\infty} |F(t) - G(t)|\, dt & \text{if } P > 0 \land R > 0 \\ 0 & \text{otherwise} \end{cases}, \tag{4.12}$$

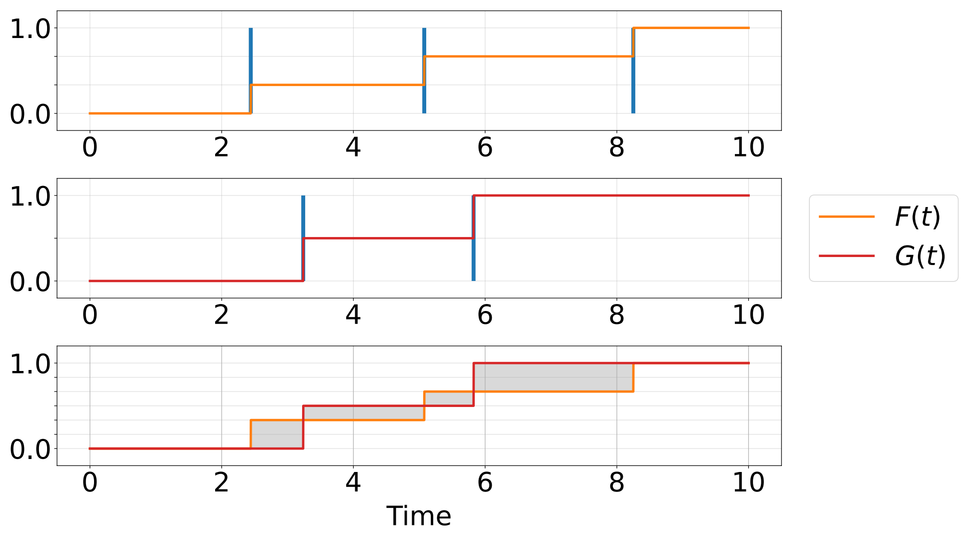


Figure 4.3: Top and middle panels: spike train embeddings $f(t)$, $g(t)$ and their respective cumulative distribution functions $F(t)$, $G(t)$. Bottom panel: the shaded area is the EMD between spike train embeddings $f(t)$ and $g(t)$.

which obviously requires that the R_{spiking} component is also a part of the overall loss function. During inference we rely on the solution to the unbounded distance problem proposed in [177]. Let f_0 denote an embedding of an empty spike train and f_n correspond to an embedding for a spike train composed of n uniformly distributed events on some bounded interval. Then

$$\text{EMD}(f_0, g) = \lim_{n\to\infty} \mathbb{E}\left[\text{EMD}(f_n, g)\right] , \tag{4.13}$$

where the events in f_n are distributed in the interval imposed by the spike train embedding g. This definition stems from the intuition that a spike train composed of uniformly distributed events carries little information about spike timing, and so does an empty spike train.

4.3 Exploring the properties of the Siamese SNN

To verify the feasibility of training a Siamese spiking neural network using the proposed objective, the model is trained on the MNIST dataset. This follows a well established practice of evaluating novel image processing algorithms on this specific dataset, which has several benefits. First of all, having a common benchmark dataset encourages comparison with other methods, and promotes conducting reproducibility studies. Secondly, the MNIST dataset is considered relatively simple, therefore it is prudent to validate a novel algorithm on a simple benchmark before applying it to more challenging problems. Lastly, the experimentation sheds light on reasonable initial settings for hyperparameter tuning. The presented summary of the MNIST-based experiments is focused on analyzing the impact of the data-to-spike conversion scheme on the properties of the Siamese SNN, particularly on spiking activity sparsity and model response latency.

4.3.1 Data preprocessing

In order to convert the static MNIST images into the spiking domain, we devise three separate coding schemes which differ primarily in the number of events passed down to the network input neurons. This allows us to explore how the resulting model properties are influenced by different input data pixel-to-spike conversion methods. Each 28×28 image is first flattened into a 784-element vector. The resulting vectors are processed differently, depending on the experimental setting:

- *Black&white* (adapted from [123]): the vector representation of each image is binarized with a threshold of 50% of global maximum pixel intensity, and one of two time instants t_0 or t_1 is assigned to the value of each bit. We set $t_0 = 0$ and $t_1 = 1.79\ \tau_{\text{syn}}$, which correspond to the values chosen by [123] for $\tau_{syn} = 1$. This is the same encoding scheme introduced in Section 3.1.4.2 and used throughout the previous Chapter.
- *Binary*: each vector representation is binarized as in the *black&white* setting; however, only a single time-instant t_0, associated with white pixels, is used to describe the signal. This slight modification of the previous setting stems from the observation that defining two event-types, spiking at two different points in time, is redundant. Events are mutually exclusive and thus the presence of one event implies that another could not have occurred. As an implementation detail, we associate an auxiliary event-time t_1 with black pixels and set $t_1 = \infty$.
- *Grayscale*: the original grayscale images are converted to the spiking domain by modeling each pixel as an artificial neuron responding to a driving signal (synaptic current) of a constant intensity I proportional to the image pixel intensity (in range 0-1). For $V_0 = 0$ the formula for the membrane voltage of the IF-based converter neurons is $V(t) = \frac{t}{\tau_{syn}} I$, and the spike time corresponding to a pixel of a given intensity is

$$t_{out} = \frac{V_{thr}\tau_{syn}}{I} . \tag{4.14}$$

  This model retains the desirable property that $t_{out} \longrightarrow \infty$ as $I \longrightarrow 0$.

Figure 4.4 summarizes the three image conversion schemes. Note that in the *binary* and *grayscale* settings some channels might not have any events associated with them. In this context a lack of an event occurrence carries implicit information [204] that can be exploited by the network.

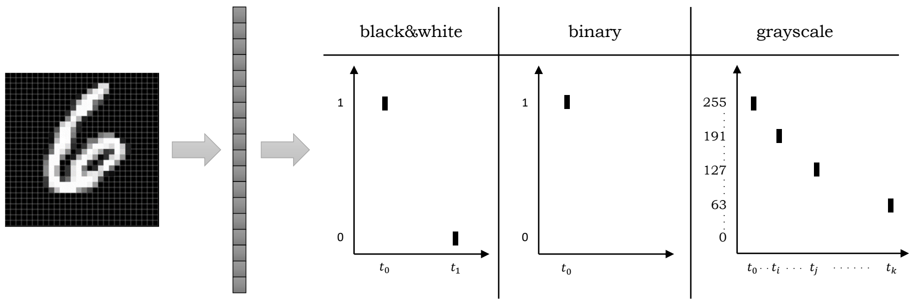


Figure 4.4: Image-to-events conversion schemes. Each pixel of the flattened image is associated with a single spike event.

For each of the three settings several 784-400-400-X networks were trained (denoting the number of neurons in input, hidden and output layers, respectively). The dimensionality of the last layer X was varied between 1 and 100. For brevity we use a *setting-X* notation which can be understood as "a model trained under the experimental *setting* with *X* output neurons". During training we monitor the ratio of active triplets in the batch (4.8)) and stop training when it does not decrease for 5 epochs. Each model was optimized using the RMSprop algorithm [161] with a learning rate of 10^{-3}, synapse regularization parameter $\gamma = 400$, synaptic time constant $\tau_{syn} = 1$, voltage threshold $V_{thr} = 1$, and the triplet loss margin $\alpha = 0.1$. Additionally, we apply the L_2 regularization with $\lambda = 10^{-3}$ to the loss function (4.4). Finally, we find that models trained in the *binary* setting require a much larger batch size of $n = 256$ to effectively train, whereas both *black&white* and *grayscale* models can be trained with a batch size as low as $n = 64$ examples. Therefore, a batch size of $n = 256$ was used across all experiments. Each model was trained 5 times, starting from different weights. Unless noted otherwise, the presented results were obtained by averaging over different versions of each model (discarding any outliers).

4.3.2 MNIST digit classification

In order to evaluate our approach, we measure the k-Nearest Neighbor (k-NN) classifier performance as a proxy for the embedding space example proximity. A spike train embedding was computed for each example using the trained Siamese SNN, then k training set embeddings closest to a given test set embedding were selected, which then could be used to determine the test example label prediction by majority voting. We found that the classifier performance is not influenced by changing k for $k \geq 7$. Therefore, we set $k = 7$ for all experiments. Compared to the other time-coding SNN (Table 4.1; refer to Table 3.1 for comparison with other training

Table 4.1: Classifier performance of different time-to-first-spike SNN on MNIST.

Model type	**Network architecture**	**Performance**
classifier	784-800-10 [123]	0.975
	784-400-400-10 [123]	0.971
	784-400-400-10 (our)	0.971
Siamese network	784-400-400-10 (our)	0.948

approaches), our best-performing model achieved a similar level of performance using a novel approach to dataset encoding and signal transformation defined by the trained spiking neural network. Importantly, the focus of this study was to show that the proposed methodology can be used to train multilayer Siamese spiking neural networks with timing-sensitive neural coding. Obtaining high accuracy was a secondary objective, mainly as a proxy for determining whether the training procedure was successful or not.

The results obtained for each experiment for different output layer dimensionality are summarized in Figure 4.5a. Models exhibit a steady increase in the classifier performance (and as a result: the spike embedding quality) as the number of output neurons increases, up to some experiment-dependent number of neurons, which then slowly deteriorates with a further increase in layer size. We presume that a gradual drop in accuracy for larger output layers might be a result of either a poor training hyperparameter choice, or due to the reduction of class embedding separation in a larger embedding space. We test this claim by computing the empirical relative contrast [184]

$$\mathrm{RC} = \frac{D_{max} - D_{min}}{D_{min}}, \tag{4.15}$$

which compares the distance to the closest- (D_{min}) and the furthest-neighbor (D_{max}) for each example. We compute the average relative contrast over all test set examples for each trained model and summarize the results in Figure 4.5b. The average relative contrast diminishes as the number of events increases. However, we also observe a plateau similar to the one in the classifier performance. These results show that the effect of the number of events in the spike train on the empirical relative contrast of the EMD is similar to that of a vector space dimensionality on the L_k norm distance measures for non-fractional k [184]. This suggests that the observed model performance decrease was caused by an inherent difficulty of optimizing distance between points in a high-dimensional space.

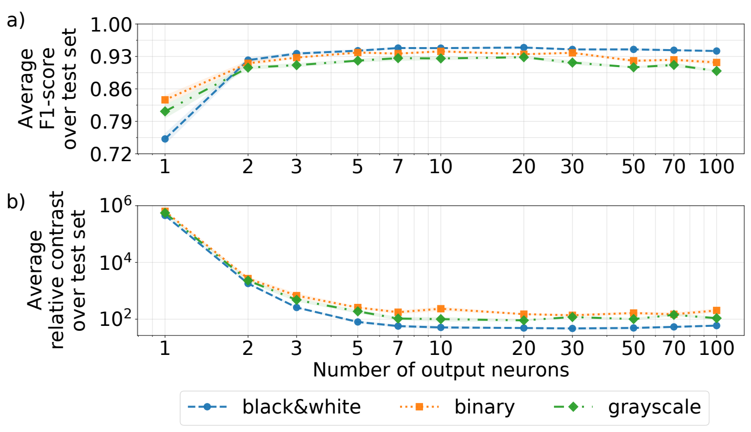


Figure 4.5: a) The impact of spike embedding length on the average k-NN classifier performance over test set examples for three different experimental settings. Note the logarithmic scale on x-axis. b) The average relative contrast vs. spike embedding length over test set examples for three different experimental settings. Note the logarithmic scale on x- and y-axis.

4.3.3 Spike train embedding visualization

In order to visualize the spike train embeddings we compute empirical distributions of output layer event-times, separately for each class, over all examples of the training set. An example of such set of distributions for a pair of models trained under the *black&white* setting is shown in Figure 4.6. Embeddings for the model with one output neuron are projected into a single point in the embedding space, whereas the model with 10 output neurons exhibits larger variability for all classes. These properties were observed across all experimental settings, implying that they depend on the output layer dimensionality and not on the chosen input encoding.

4.3.4 Hidden layer activation sparsity

Interestingly, we observed that models trained under the *binary* and *grayscale* settings exhibit a sparse internal representation of the input signal. Recall that the the fraction of quiescent neurons to all neurons in hidden layers is the network sparsity index QN_x (3.69). We compute the ratio QN_x for each example in the test set, which we denote QN for brevity. The resulting hidden layer activation sparsity empirical distribution is presented in Figure 4.7a. The results suggest that this neural activity sparsity is context-based, meaning that a different subset of neurons will respond to each input signal. Only a negligible number of neurons never fire in response to any image (corresponding to $\mathrm{QN}_x = 1$ for those images), which implies that the observed sparsity is a result of the causal set neuron selection and is fundamentally different from permanently inactive neurons which can be pruned from the network. Lastly, while there

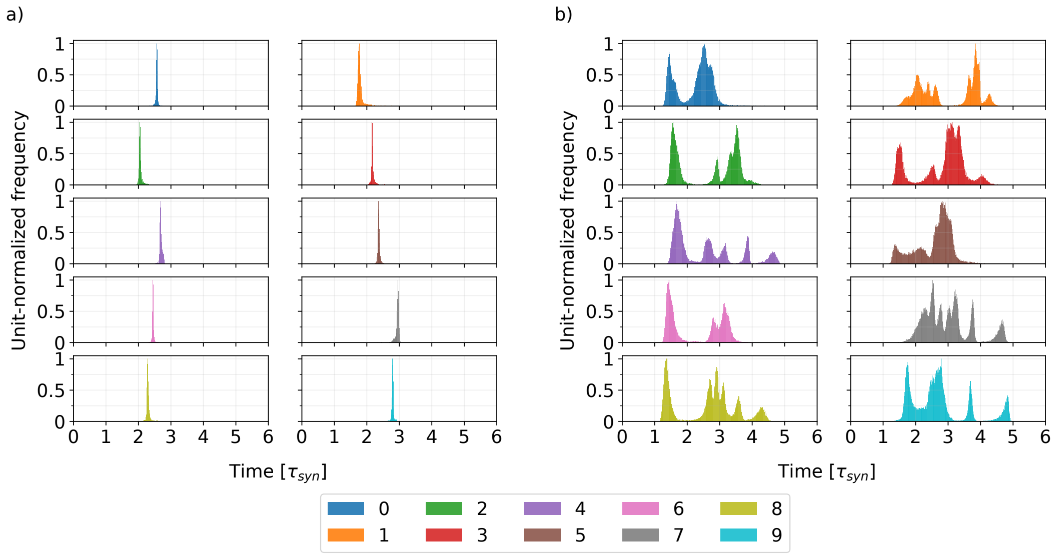


Figure 4.6: Empirical distributions of spike train embeddings computed over all training set examples for *black&white* model with: a) one output neuron, b) 10 output neurons.

seems to be a slight trade-off between the classifier performance and the network sparsity (as evidenced in Table 4.2), it can be considered small given that only about 15% of neurons are used to process each example.

Furthermore, Figure 4.7b presents the effect of network output size on the observed sparsity index. It seems that as the number of neurons increases, it becomes more difficult for the model

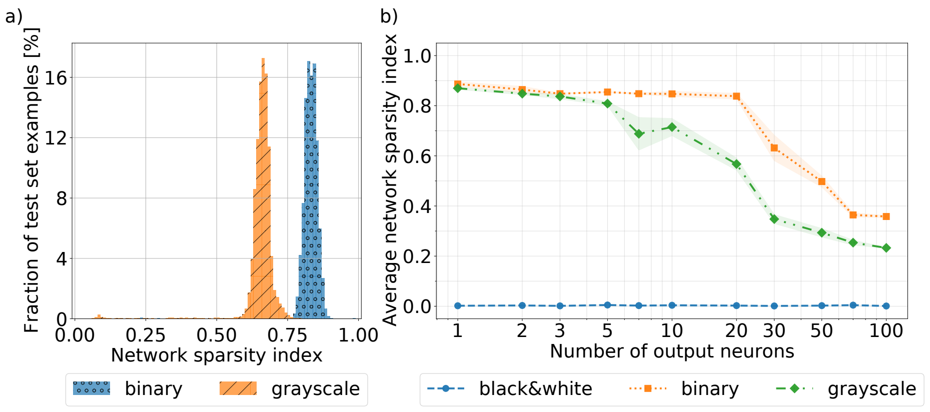


Figure 4.7: a) Network sparsity index empirical distributions for the *binary-10* and *grayscale-10* models. b) Test-set-averaged network sparsity index as a function of the number of output neurons for each experimental setting.

Table 4.2: Summary of the test-set-averaged classifier performance and the observed network sparsity indices QN for models trained with 10 output neurons.

Model name	**F_1-score**	**QN**
black&white-10	**0.9480 ± 0.0009**	0.0038 ± 0.0026
binary-10	0.9410 ± 0.0021	**0.8473 ± 0.0100**
grayscale-10	0.9257 ± 0.0030	0.7148 ± 0.0344

to process information using only a small subset of hidden layer neurons. This drop in the sparsity index value is gradual for the *grayscale* model, but much steeper for the *binary* setting. Note that the *black&white* model almost always uses all neurons in the network, regardless of the last layer shape (the largest observed value of QN for this model type was 0.0049 ± 0.0009 for 5 output neurons).

4.3.5 Classifier time-performance

In order to investigate how the classifier performance changes as output events are observed over time, we simulate the use case where classifier is asked to update its prediction whenever a new output event occurs by comparing with the embeddings of the training set. This procedure provides a valuable insight into the prediction latency. Figure 4.8 shows the class-averaged classifier accuracy for models trained with different encoding schemes. Interestingly, while the classifier performance of the *binary* and *grayscale* models roughly correlates with the overall number of observed output spikes, the performance for the *black&white* model stops improving quite early and reaches its maximum only after observing a small number of late events. If we consider the maximum-accuracy performance of the model as its steady-state, then we find that the *black&white* model achieves steady-state about 45% later than the other models. More broadly, the results obtained for the three experimental settings vs. the number of output neurons are summarized in Figure 4.9. Interestingly, the *black&white* models were consistently slower than their *binary* and *grayscale* counterparts, up to about 37% for 50 output neurons. This analysis was restricted to models with up to 50 output neurons as conducting it for larger models becomes prohibitively time-consuming.

Overall, the model time-performance curves show that the quality of class label prediction increases over time as more output events are observed. A similar classifier accuracy vs. time study was conducted by Diehl et al. [157] by describing a rate-coding, artificial-to-spiking neural network conversion scheme. They report that the steepness of the time-accuracy curve depends on the network structure and the input signal properties. Our results seem to give

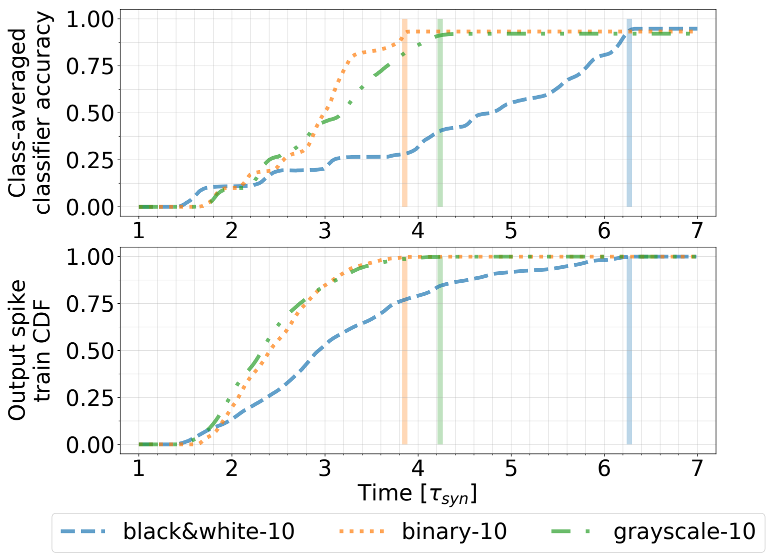


Figure 4.8: Top row: the change in class-averaged classifier accuracy over time for three models trained with 10 output neurons. Bottom row: the cumulative distribution functions (CDF) of output spike-times for all test set examples, regardless of class labels. Thick vertical lines denote the time when each model reaches its steady-state performance.

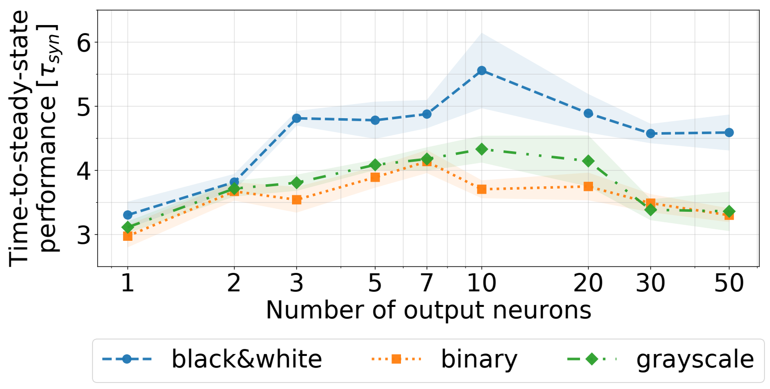


Figure 4.9: The observed relationship between the number of output neurons and the time-to-steady-state for models trained under the three different experimental settings.

further proof to their conclusions, although we did not vary the network structure.

4.4 Applications – CREDO artefacts rejection

The preliminary experiments on the benchmark MNIST dataset gave us an insight into the properties of the proposed model, as well as reasonable initial settings for hyperparameter tuning. In order to evaluate the algorithm on more challenging, real data, we apply it to images acquired by the CREDO experiment. The Cosmic Ray Extremely Distributed Observatory (CREDO) is an international research initiative aimed at observing high energy cosmic ray particles [205]. The associated Android/iOS application enables the registration of muons with smartphone devices. The ubiquity of the CREDO infrastructure entails virtually no control over the detectors' working conditions. As such, it is imperative to design a mechanism to automatically differentiate images of signal from artefacts. This is a subtype of the same/different discrimination-type machine learning problem to which the Siamese neural network model is suited for. Furthermore, the region of interest (ROI) of each CREDO image, or the part that contains the actual information about the detected object, is quite small relative to the size of the image. This means that models processing this type of data should avoid spending computational resources on processing the parts of the image which effectively convey no information. We have shown in the previous Section that the Siamese SNN is capable of processing sparse data. Overall, the type of the research problem as well as the properties of the CREDO dataset present an opportunity to evaluate the proposed Siamese SNN approach on nontrivial data.

4.4.1 CREDO experiment description

The Cosmic Ray Extremely Distributed Observatory (CREDO) is a global research consortium running several projects related to the observation of cosmic rays (CR) with a particular stress put on the Extensive Air Showers, i.e., jets of secondary particles initiated in the atmosphere by the high energy primary particles. This research initiative operates according to the citizen science paradigm, with most of the data collected from smartphones running a dedicated mobile application. The CREDO infrastructure registers large amounts of potential hits but only a fraction of them can be attributed to the particles of interest (mostly muons). To single out the genuine particle hits, effective on-line or off-line triggers are a must [206]. The CREDO data analysis is mainly concerned with muon hits since the characteristic features make them easily distinguishable from other types of radiation. In most cases they are observed as dots (when they hit the CMOS array almost vertically) or straight lines (when they impinge at a certain angle). Less frequently than muons one can register either curvy or forking particle tracks. They are of special interest as they can be attributed to either deflection or decay of the original particle within the CMOS array. The seemingly straightforward classification of hits and their distinguishing from artefacts is hindered by both smartphones' hardware (varying the CMOS array densities) and software (denoising algorithms implemented in modern smartphone cameras). These factors make the artefact rejection and signal classification a difficult and computationally challenging task where machine learning methods are the natural choice [207]. Leveraging the large number of smartphones running the CREDO Detector mobile application one can study the correlations among observations performed in areas corresponding to the typical sizes of atmospheric showers, i.e., about 1 km.

Application of the CMOS sensors for the cosmic ray detection with the CREDO Detector mobile application is based on the same working principle that governs the operation of silicon detectors in high energy experiments [208], i.e., the collection of charge induced in the depleted zone by a passing particle. The crucial difference between sensors used in the laboratory environment and those mounted in smartphones is that the latter are primarily designed to register the visible light. And so, to use them as detectors of the corpuscular radiation, the access of light has to be cut off by tightly covering the camera lens. But even then, the sensor is subject to some level of thermal and electric noise. The detection of cosmic ray signal is manifested by bright flashes occurring on average about once an hour in the image frame. The luminosity of the array region affected by a particle impact is higher than the noise level. The particle-related flashes are either elliptical in shape (up to about 10 pixels in diameter) or longitudinal with a width of several pixels and up to 30 pixels in length. The frequency of these flashes is compatible with the average secondary muon flux measured by other experiments.

The CREDO Detector mobile application has two operational modes, the initial calibration mode and the ongoing detection mode [205]. In the calibration mode the application collects statistics from 500 frames and calculates the thresholds of the detection. In the detection mode the application is acquiring, collecting, and analyzing data from detectors. In this mode two steps are executed. Firstly, the algorithm verifies the proper camera covering by integrating the frame luminosity. The frame that passes this test is considered as signal detection. In the second step the information about the detection is sent to the server. The relevant part of the frame is cut out and transformed into a 60×60 pixel image around the center of the mass of the hit. The image is sent to the server along with other data collected by the application. A single bright cosmic ray flash recorded by the typical HD camera (1280×720) usually consists of no more than 100 pixels, whereas an image containing multiple flashes consists of about 300 pixels that are brighter than the threshold, which is equivalent to 0.01% and 0.03% of such a frame, respectively.

A subset of the CREDO dataset is available publicly.[b] This dataset – the focus of our

[b]The dataset is available at https://github.com/credo-ml/cnn-offline-trigger/blob/main/data-set.zip

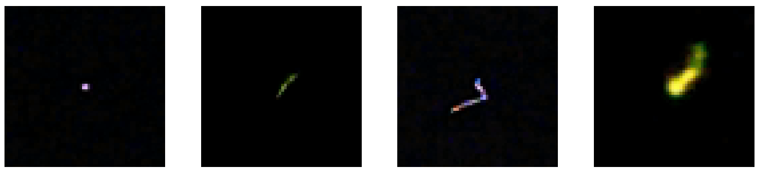

(a) spot **(b)** track **(c)** worm **(d)** artefact

Figure 4.10: Selected examples from the CREDO dataset. Images (a)-(c) represent a useful signal.

research – consists of 1232 images of a useful signal: 535 dot patterns (“spots”), 393 straight line patterns (“tracks”), and 304 forking patterns (“worms”). Additionally, it contains 1122 images of artefacts. Figure 4.10 presents an example of each signal type and an artefact. It is clear that the effective ROI of the image is a fraction of the total image area.

4.4.2 Data preprocessing

We preprocess the CREDO data by following the scheme outlined in [206]. The values of all $60 \times 60 \times 3$ pixels images are summed across the three color channels, which removes redundant information carrying no physical interpretation. Next, for every image a separate threshold is computed as

$$\gamma_i = \min\left(100, \overline{b_i} + 5\sigma_i\right) , \tag{4.16}$$

where $\overline{b_i}$ is the average brightness of the i-th grayscale image, and σ_i is its standard deviation. All pixels below the threshold are set to zero. As an additional, final preprocessing step that was not present in the reference paper we perform min-max scaling on the images to remove the bias that may be associated with an absolute brightness level (i.e., one class having images that are on average brighter than others). The resulting set of images has a very small effective ROI compared to the image size, with on average almost 90% of pixels being equal to zero.

In order to train a spiking neural network, we considered only the *binary* and *grayscale* encoding types as valid strategies, as only these two coding schemes resulted in models with sparse activity in the preliminary MNIST experiment. In addition to the Siamese SNN, a nonspiking network (denoted ANN) was trained as a baseline reference. Both the SNN and ANN models had the same 3600-256-256-10 architecture which corresponds to about $9.9 \cdot 10^5$ parameters. Setting the dimensionality of the last layer to 10 is once again a direct follow-up of the MNIST study which suggested a good compromise between classification performance and desirable properties (sparsity, time-to-steady-state). The set of hyperparameters used to train the models is presented in Table 4.3.

The class labels were assigned according to the original study [206]. Merging the labels for spots, tracks and worms into a joint “signal” class, and leaving the “artefact” set of images

Table 4.3: The set of hyperparameters used to train the models on the CREDO dataset. Both models were presented with the same number of training examples over the course of training. The value of the synaptic regularization strength varies depending on the current training epoch.

	ANN	**Siamese SNN**
learning rate	10^{-3}	10^{-3}
batch size	32	64
num epochs	40	40
steps per epoch	100	50
L_2 regularization λ	10^{-3}	10^{-3}
synaptic regularization	n/a	epochs 1-12: 10^5 epoch 13-40: 10^{-2}
synaptic time constant [a.u]	n/a	1
triplet loss margin	n/a	10^{-1}

intact results in 1232 examples of the "signal" class and 1122 examples of "artefacts". Given the small training set size relative to the number of parameters in the network, we opt to use data augmentation:

- horizontal flip,
- vertical flip,
- rotation by a random angle sampled from $[-45°, 45°]$ such that pixels of the rotated image which do not correspond to any pixel of the image prior to the affine transform are set to the value of the nearest border pixel.

Augmentations are applied in a random order, each one with probability 0.5. Note that for the SNN the data augmentation is applied before the conversion to the spiking domain. Finally, the training procedure of the two models is repeated 25 times by sampling the training-test split according to a repeated stratified k-fold validation protocol (5 rounds with 5 folds each). Each training minibatch was balanced in terms of class labels.

4.4.3 Results & discussions

The sparsity of the input data and the rich information carried by the pixel intensity levels turned out to be prohibitive in training the SNN with binary encoded inputs. Therefore, any further mentions of the Siamese SNN model properties and performance refer only to the *grayscale* model. We found that in order to successfully train the SNN models on the CREDO data it was important to substantially change the synaptic regularization parameter γ (4.4) during training (Figure 4.11). Initially γ is set to a large value of 10^5 in order to guide the model towards a

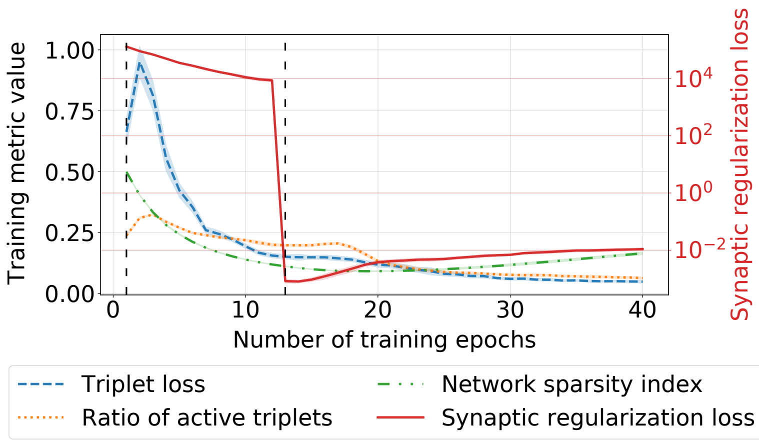


Figure 4.11: Training measures monitored during training of the Siamese SNN. Dashed vertical lines denote the epochs in which the synaptic regularization parameter changed.

solution that allows the input spikes to propagate through all layers. Once the ratio of active triplets (4.8) plateaued – which implies that the network learned how to solve the "easy" triplets – the value of the parameter γ is decreased to 10^{-2}. This stage of training emphasized solving the actual task without focusing too much on ensuring that all neurons produce spikes. As a result, the network sparsity index (3.69) (averaged over training minibatch examples) steadily increased from the achieved local minimum. The devised strategy shows the importance of correctly choosing the relative impact of the task-specific loss component (triplet loss) and the synaptic regularization on the minimized total loss function.

A summary of the classification performance of the SNN and ANN models compared to prior results is presented in Table 4.4. For the Siamese model the labels were predicted with a k-NN classifier for $k = 7$. Both of our models achieved similar accuracy scores for signals and artefacts separately, which may be a result of ensuring class balance within each training batch. We note that both models were outperformed by the CNN-based architecture, although the difference is relatively small. Nevertheless, extending this work to convolutional spiking neural networks might be worth considering as part of further research.

Across all trained Siamese SNN models we noticed that what the network considers "most similar" does not necessarily align with the perceptual similarity between a pair of images. This can be observed by comparing sets of images using a trained Siamese network, which makes the model return images that are most similar to a given query image. Figure 4.12 presents an example of querying the model using images from a test set against all images that – for this specific model – were a part of the training set. During training images of spots, tracks, and worms were all treated as a catch-all "signal" class; therefore, unsurprisingly, the model is

Table 4.4: Classifier performance on the CREDO dataset for the binary signal/artefact classification problem (best reported results from each paper).

Model	**Overall Accuracy ± Std Dev**	**Signal Accuracy ± Std Dev**	**Artefact Accuracy ± Std Dev**
Random Forest [206]	97.07 ± 0.77	99.17 ± 0.73	94.76 ± 1.39
CNN+DWT [206]	98.93 ± 0.39	98.99 ± 0.63	98.86 ± 0.67
ANN (our)	97.70 ± 0.69	97.56 ± 1.05	97.86 ± 1.00
Siamese SNN (our)	96.35 ± 0.74	96.20 ± 1.35	96.52 ± 1.36

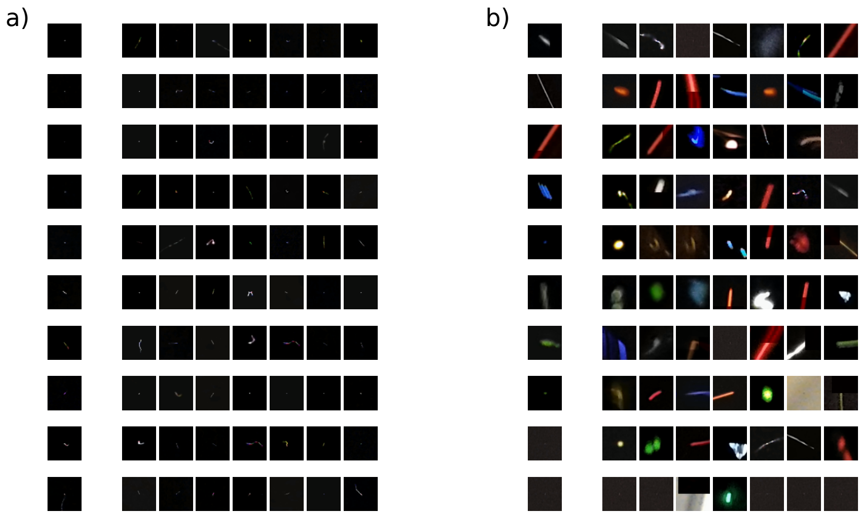


Figure 4.12: Examples of similar images of the CREDO data identified by the Siamese SNN for a) signals and b) artefacts. Each row corresponds to a different comparison, and is comprised of a test set image (left-most column), followed by top-7 most similar images from the training set. Best viewed in color.

unable to distinguish between them. Similarly, it might be difficult to understand what are the common features of the queried artefact images. This suggests that the proposed Siamese SNN suffers from the same limitations with respect to prediction interpretability as a regular ANN.

In order to demonstrate how the SNN is able to adapt to the properties of the input data, we compute the network sparsity index (3.69) separately for both class labels, and depending on whether the label predicted by the model matches the ground truth. The results are presented in Figure 4.13. For images of useful signal the network requires only a fraction of its neurons to produce a correct prediction. This stems from the fact that those images have a very small ROI, which when converted to the spiking domain results in a sparse vector. Conversely, artefacts

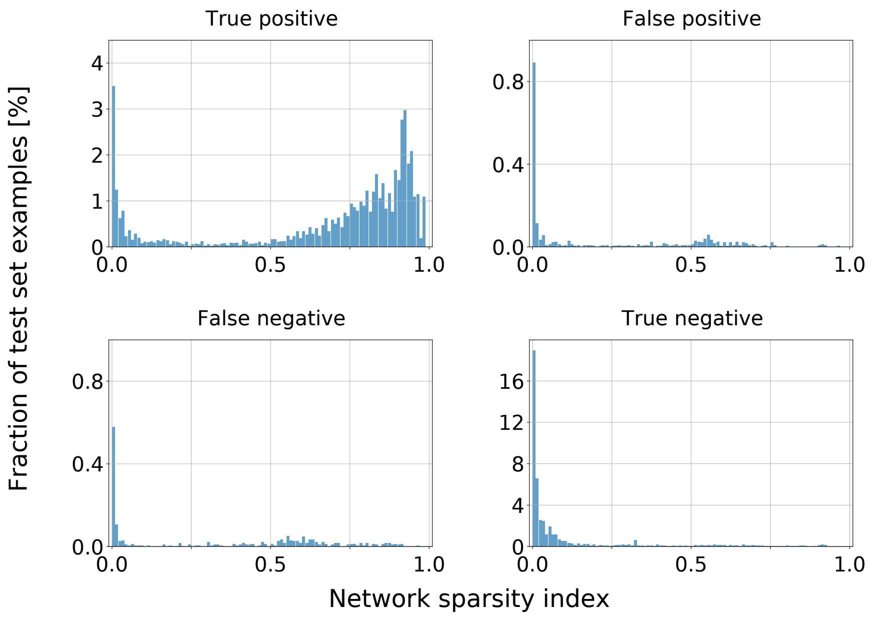


Figure 4.13: Siamese SNN network sparsity index empirical distributions estimated for all 25 experimental training runs. Examples with a ground truth label of a signal are denoted as positive.

have many more active pixels, and so the model uses more neurons to process each example. Interestingly, in all four cases the network sparsity index has a clear peak at zero, regardless of the ground truth class label. This observation, coupled with the fact that in both the false positive and false negative scenarios no peaks are evident for higher values of the network sparsity index, suggests that the examples from the set of difficult training triplets (i.e., the ones that actually contribute to the training loss), are processed with almost all neurons. Unsurprisingly, given that the model did not reach zero training loss, this results in the region of high test set error.

Finally, we analyzed the time-performance of the spiking neural network model separately for each class label (Figure 4.14). The model seems to reach the steady-state performance for artefacts significantly earlier than for the signal class. In fact, according the the output spike train CDF, the steady state is reached before even a fraction of spikes that form the class embedding for signals is generated. This suggests that the classes are well-separated in the time domain, which unfortunately means that the best overall accuracy is achieved quite late, after the steady state is reached for the other class. Note that the change in the classification accuracy over time closely follows the spike train CDF for that class. This matches the pattern observed previously for the MNIST dataset (Figure 4.8).

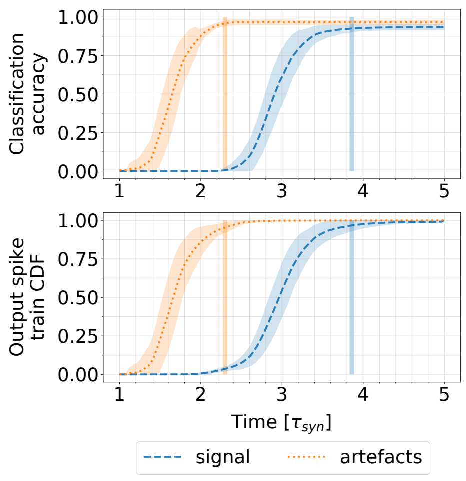


Figure 4.14: Top row: the change in classification accuracy over time per class label for the Siamese SNN models. Bottom row: the cumulative distribution functions (CDF) of output spike-times for all test set examples. Thick vertical lines denote the time when each model reaches the steady-state performance.

4.5 Summary

In this Chapter we presented a novel supervised training scheme for multilayer Siamese spiking neural networks which optimizes the Earth Mover's Distance between output spike trains. We built the network using Integrate-and-Fire neurons that are tuned to respond to precise timing of input events. In contrast to existing works that adapt the Siamese networks to the spiking context, our Siamese SNN is optimized directly in the spiking domain, rather than be a product of converting an existing neural network to the spiking domain. Additionally, we opt to encode information using single events instead of bursts of spiking activity.

The model was evaluated on image data converted into the spiking domain using a novel input coding scheme based on the concept of implicit information carried by events in the spiking domain. This means that some data inputs are represented by a lack of any event rather than a new event category. We find that for the MNIST data the proposed methodology is robust to the change in output layer dimensionality, which can be tuned to the task at hand. This training procedure results in models which take less time to make high-accuracy predictions and process signals using only a small subset of hidden layer neurons firing in response to the input event stream (*binary*, *grayscale*), compared to models trained with an explicit event encoding (*black&white*). Importantly, these model properties held true across all tested output layer dimensionalities. As a concrete example, the *black&white-10* model reached an F_1-score of 0.9480 using almost all neurons to make predictions, whereas the *binary-10* model achieved a slightly lower F_1-score of 0.9410; however, it encodes information using only 15% of all hidden layer neurons, and is significantly faster to reach its steady-state performance. Further investigation is required in order to determine whether the observed accuracy-sparsity trade-off is a result of our training procedure, or whether it is an inherent property of spiking neural networks.

Training the Siamese SNN model on a more challenging dataset of the CREDO experiment images required carefully selecting the value of the synaptic regularization parameter and changing it as training progressed. Doing so has forced the model to focus on ensuring spike propagation through the model during the initial training stage, and minimizing the task-related loss only after the regularization component has been relaxed. We hope that this insight will be valuable in further research studying such networks. Contrary to the results obtained on MNIST, we were unable to successfully train the *binary* model, which might be a corollary of almost 90% of input pixels (on average) being zero, as well as due to discarding information present in different pixel intensity levels when transforming the image to the spiking domain. The *grayscale* model achieved a performance level similar to the nonspiking ANN baseline, although they were both outperformed by a CNN-based solution. Nevertheless, the trained Siamese SNN model has shown the ability to adapt to the properties of the input data by using a fraction of all neurons to process examples of the "signal" class, whereas using almost all of them to process "artefacts".

Note that while the proposed approach is cast in the context of image classification, it can be readily applied to problems that are solvable with regular (nonspiking) Siamese neural networks, such as object tracking or change point detection. Naturally, the model requires that the data either exists in or can be converted to the spiking domain. This spike-domain-centric design makes the model suited towards analyzing sparse and irregular patterns present in the spiking data. Conversely, it is difficult to predict how the model will perform compared to alternative methods when the underlying data exists in the nonspiking domain as the conversion procedure creates a spike sequence that is much more parsimonious than the original data (in this case: static images). Investigating whether this procedure is of any benefit (by removing redundant information) warrants further research.

It must be noted that both the MNIST and CREDO datasets contain images of a relatively small size, which allowed us to train models with the spiking analogue of the dense layer introduced in Chapter 3. However, this might become impractical for images of larger size. Adapting the proposed spiking neuron to construct a convolutional layer is surely worth further investigation. As an added benefit, the use of CNN-based architectures increases the interpretability of model predictions for image data. Lastly, combining such developments with the MIMO framework presented in Section 3.2.4 would allow applying the model to image streams instead of static images.

Chapter 5

Event Sequence Classification for Multivariate Time Series

Event sequence data is a natural representation for signals in which the information is carried only by a set of predefined events, such as banking or social media activity. There is no information in between the two events. However, in some cases the event sequence is created by monitoring some complex phenomenon in which the underlying state of the system may have changed between two different events. To illustrate this, take for example an electronic health record in a hospital admission [8]. The efficiency of patient's state monitoring and treatment is conditioned on the frequency of events registered in the record. However, given the budget constraints, continuously monitoring the state of all patients in a facility at an arbitrary frequency is unfeasible. Therefore, it is crucial to make informed decisions on *when* to create a new event. In fact, the monitored phenomena need not be complex for this scenario to occur. Event-sensors aim to improve upon typical sensors sampling an analog signal at a uniform rate by focusing only on the informative data. Such event-driven sampling schemes additionally offer wide dynamic range and low latency in asynchronous communication.

In this Chapter we focus on analyzing the performance of the SNN models trained on event data obtained by different signal-to-spike conversion schemes. In contrast to Chapter 4 which described conversion of static images to events, here we operate on multivariate time series data. The presented approach is specifically applied to data originating from an Intelligent Transportation System (ITS) sensor for vehicle monitoring. Notably, this methodology makes no assumptions about the underlying signal, ensuring that it can be readily adapted to other data domains. In Section 5.1.1-5.1.2 we introduce various event-triggered sampling schemes and provide details about the analyzed signal, respectively. Then, Section 5.2 describes the proposed approach. The solution is split into two parts. Section 5.2.1 shows how to choose the signal encoding such that information important for the downstream classification task is preserved. These insights are followed-up upon in Section 5.2.2, which summarizes the results of training the SNN models on this dataset converted into the spiking domain with the chosen encoding schemes. In our analysis, in addition to the absolute classification performance, we also focus on the efficiency of the solution by relating the model performance with the number of events produced by an event-triggered sampling scheme.

5.1 Introduction

5.1.1 Event-triggered sampling

Conventional data acquisition of bandlimited signals applies uniform sampling, with the data sampling frequency selected according to the Nyquist–Shannon theorem. This means that in order to perfectly reconstruct the signal from its samples, the sampling rate must be twice the highest expected spectral frequency component of the signal (unless some additional information about the signal is available, as in e.g., compressed sensing [209]). However, this is far from optimal when signal's spectral properties significantly change over time, e.g., by alternating low- and high-frequency signal content, or even lacking any amplitude changes altogether. This is undesirable in applications that rely on efficient use of available resources. In such scenarios non-uniform sampling techniques can be employed [210].

Event-triggered sampling is one such strategy that attempts to address these limitations by making the sampling frequency adapt to the signal itself (Figure 5.1). It does so by letting events dictate the sampling instances, i.e., samples are generated only when the signal satisfies some pre-defined event [211]. Fortunately, the triggering condition definition is flexible, allowing event-based sampling to comply with a wide range of design objectives. Employing this strategy in signal acquisition systems removes the need for a sampling clock, which is a significant energy consumer in uniform sampling strategies. Additionally, moving the quantization process from the amplitude to the time domain is in line with modern electronics manufacturing technology that promotes fast circuit operation but makes the fine quantization of the amplitude difficult at low supply voltages [211].

An important class of event-based criteria is the reference-crossing sampling. The signal is sampled whenever it intersects with a predefined reference function. The most popular sampling scheme of this type is the level-crossing sampling which defines crossing some amplitude level as the event trigger [212, 213]. In practical applications multiple levels are used. They can be positioned arbitrarily, although the typical solution is to set them at uniformly distributed levels along the amplitude range of the signal (Figure 5.2a). The rate of events is directly proportional

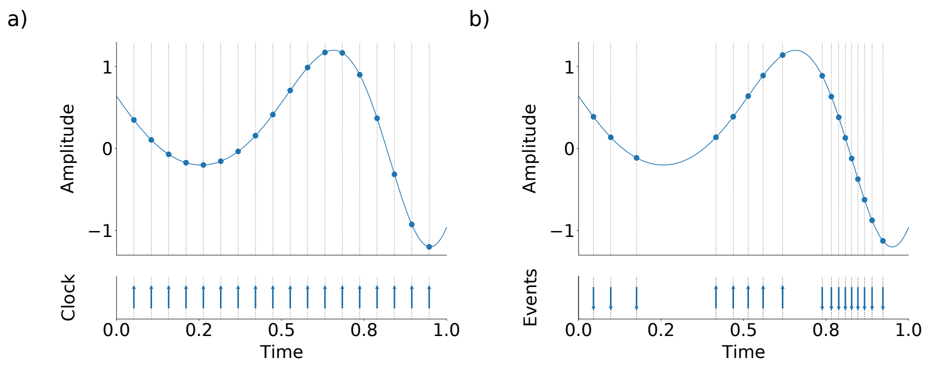


Figure 5.1: Comparison between periodic (left) and event-triggered (right) sampling schemes. The latter sampling scheme produces two event sequences, separately for the rising and falling slopes of the signal.

to the variability of the signal being sampled, with higher rates corresponding to periods when the signal varies quickly. It is therefore possible to estimate the local signal bandwidth based on the event counts [214].

Reference-crossing schemes also encompass level-crossing sampling of the transformed input signal, such as its derivative [215]. A special case of this is the extremum sampling, which is a one-level, zero-crossing sampling scheme of the signal derivative [216]. Moreover, the reference signal need not be constant over time [217]. Figure 5.2b-c shows examples of these reference-crossing sampling criteria.

A different category of event-triggering criteria that removes the need for a reference signal relies instead on temporal variations of the signal amplitude. Send-on-delta scheme [218] samples the signal whenever its value changes by a threshold Δ (Figure 5.2d). Intuitively, the magnitude of Δ should be selected so that the signal's variability is adequately captured while keeping the number of samples low. Several extensions and modifications of this algorithm have been proposed. Send-on-delta with linear prediction [219] samples the signal only when its actual value deviates from the value predicted based on its history by at least Δ. Doing so reduces the number of samples that need to be transmitted. Furthermore, send-on-area [220] and send-on-energy [221] schemes trigger sampling if the signal's integral (or energy) changes by some threshold value, relative to the most recent sample. These criteria alleviate the issue present in the send-on-delta scheme when samples are rarely generated because all amplitude variability is between the threshold $\pm\Delta$ (Figure 5.2e)

In all methods discussed so far the produced samples are implicitly associated with the signal amplitude at the time of the measurement. An interesting concept that uses time as the sole source of information about the signal is the time encoding machine (TEM) [222]. These are real-time asynchronous circuits capable of encoding the signal in such a way that

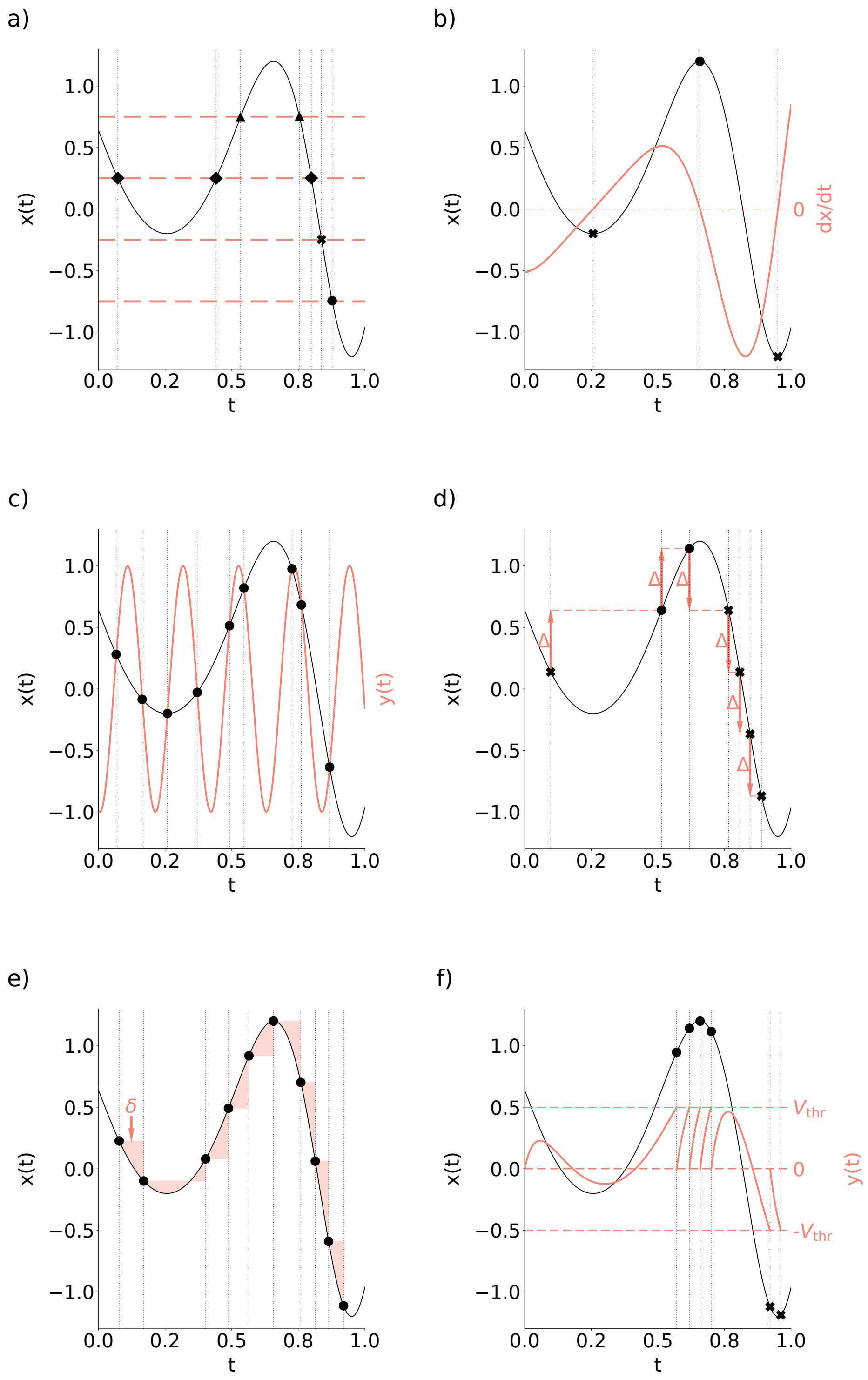


Figure 5.2: Examples of event-triggered sampling criteria. Black marker symbols denote different event types produced by the encoding. a) Level-crossing. b) Extremum sampling. c) Sine-wave-crossing. d) Send-on-delta. e) Send-on-area. f) LIF time encoding machine.

the amplitude information can be recovered. It has been shown that neural models such as the (leaky) Integrate-and-Fire neuron [223, 224] or the Hodgkin-Huxley neuron [225] can be used in the TEM circuit to encode the signal. For instance, the events produced by the LIF-TEM are recursively defined as

$$t_n \coloneqq \min\left\{t > t_{n-1} : \frac{1}{\tau_{\text{int}}} \int_{t_{n-1}}^{t} \exp\left(-\frac{t-s}{\tau_{\text{leak}}}\right) [x(s) + c]\, ds = \epsilon V_{\text{thr}}\right\} \tag{5.1}$$

starting from $t_0 = 0$, where $\tau_{\text{int}} > 0$, $\tau_{\text{leak}} > 0$, $c \geq 0$, $V_{\text{thr}} > 0$ are known constants, and $\epsilon = \{-1, 1\}$. τ_{int} is the integration constant, τ_{leak} is the leak constant (controlling how quickly the TEM is discharged in the absence of new information), and c biases the signal away from zero. Figure 5.2f presents an example of this encoding. Note that the LIF-TEM produces two types of events, depending on the sign of the definite integral in (5.1).

Overall, the main focus of research in the field of event-triggered sampling is on recovering the original signal from its samples. While the general theory of signal reconstruction from non-uniform samples can be applied to any event-based sampling method [226, 227, 228], the development of techniques dedicated to specific encoding types aims to further reduce the number of generated samples by exploiting known signal properties [212, 214, 229]. Analyzing different event-sampling schemes through the lens of signal reconstruction is one possible way of assessing their usefulness and robustness in solving real tasks. However, not all applications need to reconstruct the signal. We argue that it is also important to analyze these methods in terms of how the produced signal representation impacts the performance and properties of a machine learning model operating on event data. We find that this topic is insufficiently explored in the literature.

5.1.2 Inductive loop vehicle magnetic profiles (VMP)

The goal of Intelligent Transportation Systems (ITS) is to promote efficient utilization of existing transportation facilities by applying accurate traffic data acquisition technologies in order to monitor and route traffic flows. Despite recent advances in the development of different sensor technologies, the inductive loop (IL) sensors are by far the most commonly used sensor in modern traffic control systems [230]. This technology is characterized by low installation costs, robustness with respect to weather conditions such as rain, fog or snow, and flexible design that is able to accommodate different applications. Systems based on IL sensors are capable of vehicle classification [231, 232, 233, 234, 235], vehicle re-identification and tracking [236, 237, 238, 239], speed estimation [240, 241, 242], as well as wheel and axle detection [243, 244, 245].

The scientific principles underlying the inductive loop sensor operation are well understood. At minimum, the IL detector consists of two components: a wire loop with one or more turns mounted on or embedded in the roadway pavement, and a controller cabinet that houses an electronics unit [230]. When a conducting metallic mass passes over the IL connected to an oscillating circuit, the currents induced in the object change the magnetic field distribution and the loop inductance [246]. These dynamic interactions between the loop and the vehicle can be monitored and observed as impedance changes. This produces a waveform signal known as the vehicle magnetic profile (VMP).

Depending on the circuit design, different components of the VMP waveform can be extracted. This impacts their applicability to the various traffic measurement problems. The LC-generator-based circuits allow measuring only the change in the inductive reactance of the loop [244]. Conversely, systems using the AC-bridge oscillators measure impedance changes, however, obtaining precise measurements of the real (R-VMP) and imaginary (X-VMP) impedance components of the VMP waveform is time consuming [243]. A microcontroller-supported, Maxwell-Wien-bridge-based solution that uses vector voltmeter measurements is capable of preserving both the R-VMP and X-VMP components [247].

Figure 5.3a presents a quad loop, four-channel VMP measurement system proposed in [248]. It consists of two standard loops (IL1, IL3) and two slim loops (IL2, IL4) arranged in a series. The two inductive loop types differ in the produced magnetic field distribution. The standard loops generate a spread field that encompasses many vehicle chassis components. Conversely, the magnetic field of the slim IL sensors is spatially less spread. This implies that the standard loop design is better suited towards generic tasks, whereas the slim loops are preferable for axles identification and wheel rim detection [243]. Using two loops of each type instead of a single one introduces redundancy that makes the system more robust against signal interference, and makes estimating some traffic parameters (such as vehicle speed) simpler [249]. Finally, three different excitation frequencies are applied to each channel (Figure 5.3b-e). Simultaneous multi-frequency measurement improves the system robustness against noise, reducing the likelihood that the poor signal quality prevents processing in downstream tasks. We use data obtained from the quad loop system in our analysis.

5.2 Vehicle type identification based on the VMP signal

Marszałek et al. [250] proposed a system capable of inferring the carry load of a vehicle based on the magnitude of the VMP signal. Given that these signals are very similar to one another when recorded for the same vehicle model, but are quite distinct for different models, this load estimation is a two-step procedure. First, the measurement system is calibrated for a given vehicle model to determine a load-dependent parameter by observing the VMP signal obtained at different loads. This is repeated for every vehicle expected to be measured which results in a

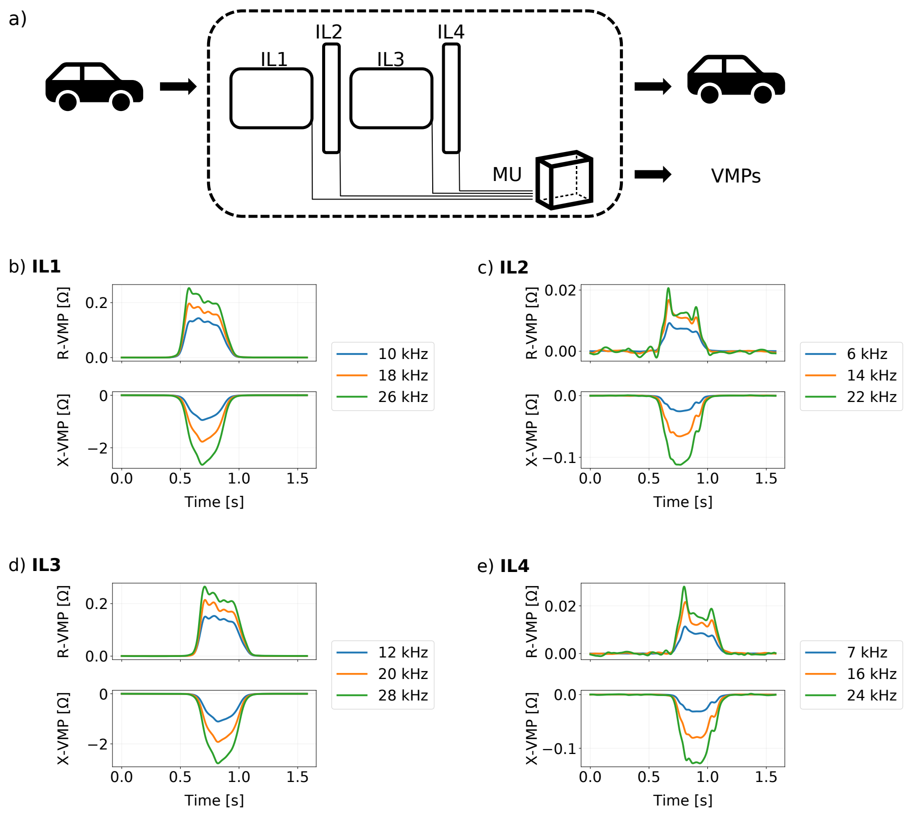


Figure 5.3: a) Diagram of the quad loop VMP measurement system. The signal is registered by the measurement unit (MU) when a vehicle passes over the four inductive loop sensors. b-e) Exemplary VMP profiles acquired by the quad loop VMP measurement system, divided into separate IL sensors, VMP components and loop excitation frequencies. Note that signals originating from the IL sensors of the same type (IL1-IL3; IL2-IL4) are slightly shifted in time. Furthermore, the loop excitation frequency impacts the signal magnitude, but does not significantly change its characteristic features.

creation of a vehicle model reference database. Then, during inference, a newly-acquired VMP signal is compared to all records in the reference database in order to determine the identity of the vehicle. Having found a match, the carry load can be estimated. This process exemplifies the importance of vehicle model detection in the VMP-based signal processing pipeline.

It is unlikely that the reference database contains exemplary measurements for all vehicles. In fact, it can be argued that some vehicle types (such as bicycles) should be explicitly excluded from the database to avoid unnecessarily comparing them with the records in the reference set. In an event that the system does support multiple vehicle types, a presence of a filtering model upstream would have the benefit of allowing specialized models for vehicle model detection, separately for each type. For these reasons we focus our efforts on the vehicle type identification problem. Note that the approach described hereinafter can also be used to train a neural-network-based vehicle model detector, provided that enough training data per vehicle model is available.

To conduct our study, we used the data provided by [250]. The entire dataset contains 3328 records, split into five classes as follows: motorcycle (24.32%), bicycle (20.42%), car (15.02%), delivery van (14.11%) and truck (6.61%). Each record stores simultaneous multi-frequency measurements of the real and imaginary components of the VMP signal (described earlier in Section 5.1.2). We take a subset of channels, leaving only four of them for further processing (the R-VMP signals registered by sensors at their lowest excitation frequency). This reduces the scope of our analysis and encourages further research that considers the full extent of the available data.

In order to process the VMP data with a spiking neural network, it is necessary to first encode the multivariate time series into event streams using one of the methods introduced in Section 5.1.1. The experiments conducted on Twitter, MNIST and CREDO data have shown that the choice of an encoding scheme impacts the number of events observed by the network, as well as the effective number of input event types. Those factors, in turn, influence the properties of the trained model. We can expect similar findings on the VMP data. Note that assessing the SNN properties with respect to the chosen encoding type and its parameters similarly to our prior analyses would require training the model from scratch many times over. Given the comparatively larger size of the VMP dataset, it is clear that doing so would be time- and compute-intensive, especially when the SNN-specific hyperparameters are also considered in the evaluation process.

For these reasons, we explore a more pragmatic approach. Rather than jointly analyzing the impact of spike encoding on the model performance, we split the process into two stages. In the first stage we focus solely on finding a set of encoding-specific parameters that preserves information that is important for time series classification. Afterwards, the VMP signals are preprocessed according to the found parameters, which can then be used to train the models. By dividing the hyperparameter optimization process into two parts we reduce the overall time it takes to arrive at a good-enough solution. Of course, it can be argued that training the SNN end-to-end with encoding optimization might ultimately lead to a better model. Nevertheless, the proposed two-step procedure is more general by being independent of the SNN model complexity.

5.2.1 Choosing the signal-to-spike encoding parameters

The main purpose of the research summarized in this Section was to find a set of encoding parameters that preserves information about the underlying signal that is important for the downstream classification task. We assume that the performance of one machine learning model on some data might be indicative of the classification measure scores of another model on the same data. Such relationship implies that changing the data source (e.g., due to preprocessing) leads to the same outcome in both models (an increase, reduction or no change of the model performance relative to the baseline), but says nothing about the model performance in absolute terms. Note that this need not be true for an arbitrarily chosen pair of models due to factors such as model complexity, or whether they are general-purpose or specifically designed for a given machine learning problem.

Let us consider the k-NN classifier approach. Its classification score provides an insight into which encoding (i.e., preprocessing) parameters produce signal spaces with relatively higher intra-class similarities with respect to the chosen (potentially parameterized) distance function. Marginalizing-out the impact of the distance function parameters it is possible to infer which encoding parameters lead to more robust signal spaces for classification problems. Therefore, we can expect the SNN classifier to perform better on such event spaces in comparison to randomly-selected ones. Naturally, this strategy is valid only when the choice of the spike encoding parameters has a measurable impact on the k-NN classifier performance and – by proxy – signals' pairwise similarity. If that is not the case, and the k-NN classifier performance is generally poor, then a different approach is needed.

A crucial part of this strategy is the choice of the distance function to assess event sequence similarity. Given that the VMP signals are relatively short (on the order of a couple of seconds), the metrics used in computational neuroscience research are appropriate. In Section 4.1.1 we introduced some commonly used similarity measures, with the Earth Mover's Distance described in greater detail in Section 4.2.1. The latter was used in the Siamese SNN training objective due to its low computational complexity and being parameter-free. Unfortunately, the EMD requires that the event sequences under comparison contain at most one event type, the same in both sequences. There is no explicit formula for the EMD between event streams composed of events of different types. While there exist some iterative approaches (most notably those based on the Sinkhorn's algorithm for entropically-regularized optimal transport problem [251, 252]), we found them to be too computationally expensive in our experimental setup and therefore a different distance measure was used.

5.2.1.1 Preliminaries

Due to its prominence in neuroscience, we opt to use the multi-neuron van Rossum distance [253] to assess the similarity of the VMP event sequences produced by spike-encoding functions.[a] Let $\mathcal{U} = \left\{\mathbf{u}^1, \mathbf{u}^2, \dots, \mathbf{u}^P\right\}$ and $\mathcal{V} = \left\{\mathbf{v}^1, \mathbf{v}^2, \dots, \mathbf{v}^P\right\}$ be two populations of P neur-

[a]The van Rossum distance was originally defined for neural spike responses, therefore its formulation refers to various (populations of) neurons. We adopt this terminology for consistency with previous works in the field, noting that different "neurons" relate to "event types", whereas "populations of neurons" intuitively correspond to "sets of event types".

ons. Without a loss of generality, assume that each neuron produces n spikes such that

$$f\left(t,\mathbf{u}^p\right)=\sum_{i=1}^{n} h\left(t-u_i^p\right) \tag{5.2}$$

is a spike train with events $\mathbf{u}^p=\left\{u_1^p,u_2^p,\ldots,u_n^p\right\}$ smoothed by a causal exponential kernel

$$h(t;\tau)=\begin{cases}0 & t<0\\ e^{-t/\tau} & t\geq 0\end{cases} \tag{5.3}$$

(the spike train $\mathbf{v}^p$ is defined analogously). Then, the multi-neuron van Rossum distance between $\mathcal{U}$ and $\mathcal{V}$ is [254]

$$d\left(\mathcal{U},\mathcal{V};\tau,c\right)=\sqrt{\frac{2}{\tau}\sum_{p=1}^{P}\left(\int_0^{\infty}\left|\delta_p\right|^2 dt+c\sum_{q\neq p}\int_0^{\infty}\delta_p\delta_q dt\right)}, \tag{5.4}$$

where $\delta_p=f\left(t,\mathbf{u}^p\right)-f\left(t,\mathbf{v}^p\right)$ and δ_q is defined analogously. For a causal exponential smoothing function this integral sum can be computed explicitly as

$$d\left(\mathcal{U},\mathcal{V};\tau,c\right)=\sqrt{\sum_{p=1}^{P}\left(R_p+c\sum_{q\neq p}R_{pq}\right)}, \tag{5.5}$$

with

$$R_p=\sum_{i,j}e^{-\left|u_i^p-u_j^p\right|/\tau}+\sum_{i,j}e^{-\left|v_i^p-v_j^p\right|/\tau}-2\sum_{i,j}e^{-\left|u_i^p-v_j^p\right|/\tau} \tag{5.6}$$

being the *labeled line* term representing the single-neuron van Rossum distance (i.e., a distance between neurons that directly correspond to one another), and

$$R_{pq}=\sum_{i,j}e^{-\left|u_i^p-u_j^q\right|/\tau}+\sum_{i,j}e^{-\left|v_i^p-v_j^q\right|/\tau}-\sum_{i,j}e^{-\left|u_i^p-v_j^q\right|/\tau}-\sum_{i,j}e^{-\left|v_i^p-u_j^q\right|/\tau} \tag{5.7}$$

is the *summed population* term representing the cross-neuron distance. Assuming that the sequences $\mathcal{U}$, $\mathcal{V}$ are sorted, the computational complexity of (5.5) is on the order of P^2n^2 and can be reduced to P^2n using a so-called markage trick [254]. The multi-neuron van Rossum distance is visualized in Figure 5.4.

The double-sums in (5.6)-(5.7) denote the computation over all pairs of events between the two spike trains. This means that these event sequences can have a different number of events each. In fact, the measure is bounded even when one spike train is empty while the other is not, and also when both event sequences are empty. This is important in our application because we cannot make any assumptions about the number of events of each type produced by the spike encoding functions.

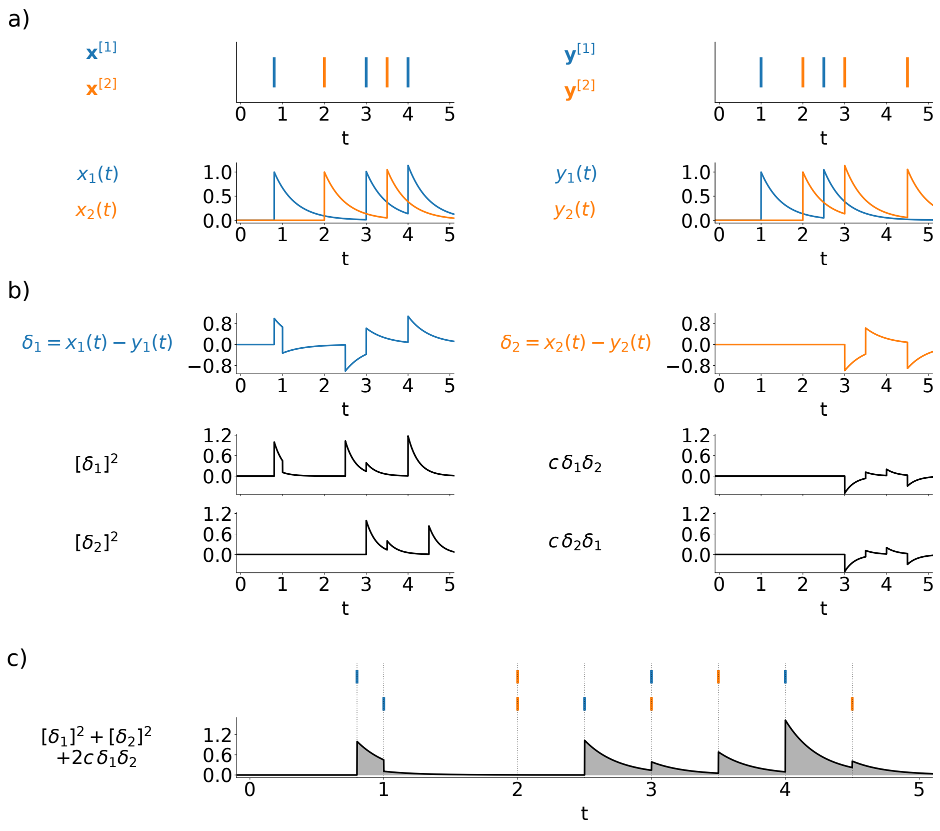


Figure 5.4: Computing the multi-neuron van Rossum distance between two neuron populations $\mathcal{X}$, $\mathcal{Y}$ with two neurons each. a) The original spike trains of the two neuron populations, as well as their causal exponential kernel-smoothed representations. b) Finding the labeled line ($[\delta_1]^2$, $[\delta_2]^2$) and summed population ($\delta_1\delta_2$, $\delta_2\delta_1$) terms. The van Rossum distance is symmetric, therefore in this scenario all summed population terms are identical. Note the presence of a scaling factor c. c) The squared multi-neuron van Rossum distance is proportional to the shaded area. The original spike trains are presented for reference.

The multi-neuron van Rossum distance defined in (5.5) is parameterized by c and τ, allowing it to model various phenomena observed in neuroscience. The mixing parameter $0 \leq c \leq 1$ weighs the importance of treating each neuron separately ($c = 0$) versus viewing the entire population as a single-unit ($c = 1$). The decay constant τ influences the range of interspike dependencies. For $\tau \longrightarrow \infty$ each individual spike contributes to the distance computed at every subsequent spike. Conversely, for $\tau \longrightarrow 0$ spikes impact only their direct neighborhood with $\tau = 0$ causing the measure to count the number of coinciding events in the spike trains. These properties show the versatility of this distance function. Unfortunately, this also means that the choice of c, τ impacts the perceived (dis)similarity of spike trains produced by different spike encoding functions.

In order to find the set of parameters for the encoding functions we conduct Bayesian optimization using the Hyperopt software package [255]. For a pre-determined training and validation data split, the procedure can be summarized as follows:

1) Define the search space for the parameters to be optimized with Hyperopt. We constrain the search space to the following set of probability distributions:
   - level-crossing encoding: $L \sim \mathcal{U}\{4, 16\}$,
   - send-on-delta encoding: $\Delta \sim \mathcal{U}(0.02, 0.2)$,
   - LIF encoding: $\tau_{\text{int}} \sim \mathcal{U}(0.05, 0.4)$; $\tau_{\text{leak}} \sim \mathcal{U}(0.2, 0.6)$; $V_{\text{thr}} \sim \mathcal{U}(0.2, 0.4)$,
   - van Rossum distance: $\log(\tau) \sim \mathcal{U}(-1, 1)$; $c \sim \mathcal{U}(0, 1)$.

   This means that the search space is spanned by either three or five different parameters, depending on the chosen spike encoding function. Note that the level-crossing encoding is defined in terms of the number of levels and not their absolute amplitude due to the signal preprocessing steps explained below.
2) Define the scoring function. As mentioned before, we choose the k-NN classifier accuracy as the optimization criterion. For each individual record of the validation dataset the classifier decision was determined by finding $k = 7$ most similar signals from the training dataset (with respect to the multi-neuron van Rossum distance) and assigning the class label based on the results of majority voting. The number of nearest neighbors k was fixed in order to avoid introducing an additional parameter into the search space.
3) Run the Bayesian optimization process over 150 steps. Each step consists of sampling a new set of parameters, extracting spike sequences from the original time series data, evaluating the scoring function, and updating the parameter distributions based on the history of scores conditioned on the parameters. Note that it is possible that the sampled parameter set sometimes produces empty event sequences. Such failed runs have their score set to $-\infty$ and are excluded from further analysis.

Lastly, to check the robustness of the optimization procedure with respect to the data distribution, we use the stratified 10-fold cross-validation. This results in a total of 10×150 distinct optimization steps for each spike encoding function.

Prior to the signal-to-spike conversion we upsample the digital VMP signals from 1 kHz to 10 kHz in order to assign the event occurrence time instants more accurately. Furthermore, as shown in Section 5.1.2, the individual signals have either all-positive or all-negative amplitude (with small variations near the zero baseline). Hence, we take the absolute value of each signal and normalize them to range 0-1. In doing so we avoid having to consider the amplitude range variability across the different VMP sensors in our analysis. As a side effect of this signal normalization process, the LIF-negative event type cannot occur in the encoded sequences. Lastly, the spike sequences are shifted so that the first event within each sequence among all event types occurs at a relative time $t = 0$. The reason is twofold. Firstly, the sensors are placed in a series; therefore, it is important to preserve the relative time shift between channels as the

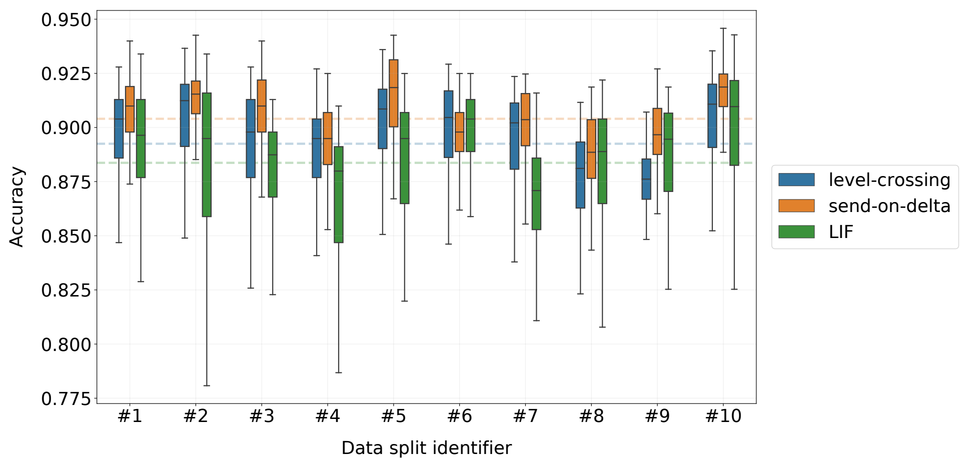


Figure 5.5: Accuracy scores of the classifiers trained during the stratified 10-fold cross-validated Bayesian optimization procedure. The dashed lines denote the average score over all data splits.

vehicle passes over the measuring system. Secondly, the information gathered by the sensor before the vehicle enters the measurement space should be discarded, enforcing an effective signal start time $t = 0$.

5.2.1.2 Analyzing the results

The results of the Hyperopt optimization runs are summarized in Figure 5.5. A wide range of obtained accuracy scores – from about 0.775 up to about 0.950 – indicates that the choice of the encoding parameters has a significant impact on the k-NN classifier performance. It is clear that some data splits are relatively more difficult to classify than others, although neither classifier significantly deviates from their respective overall-average performance. Furthermore, these results show that incorrectly choosing the LIF encoding parameters leads to significantly worse performance than for level-crossing or send-on-delta schemes. Note that at this stage it is impossible to infer whether one encoding is better than others because the Bayesian optimization process is not constrained to use the same encoding parameters across different data splits. Lastly, recall that the van Rossum distance parameters are also tuned, which influences the final classification score.

Additional insights about the coding schemes were obtained by analyzing the classification scores achieved by models trained with Hyperopt versus the average number of events produced per the VMP sensor.[b] The reason for normalizing the total number of events in the spike train by the number of VMP sensors is that the signals observed by the sensors generate roughly the same number of events after encoding, meaning that the total number of events generated in the

[b]Recall that the term "VMP sensor" can refer to either the physical IL sensor or the excitation frequency of the loop. Throughout our analyses we used four "VMP sensors" representing the R-VMP signals registered by the quad-loop system at a single excitation frequency per loop.

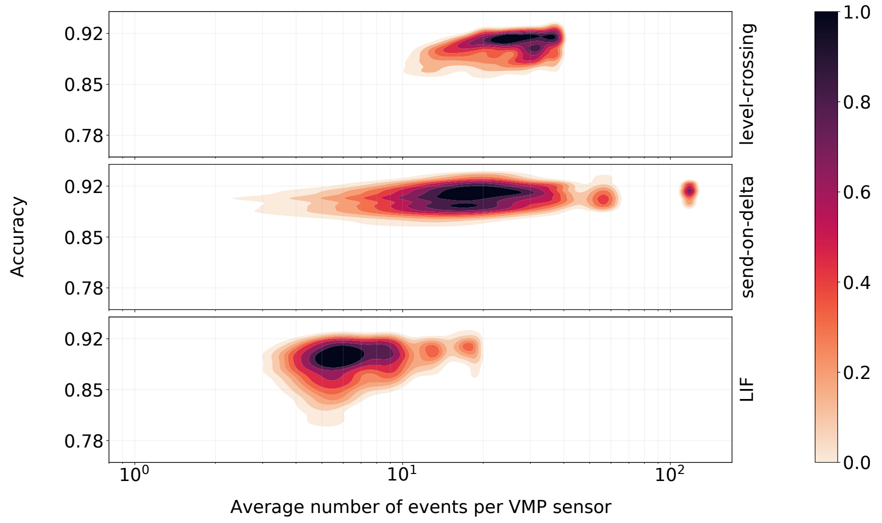


Figure 5.6: Kernel-smoothed density estimators of the k-NN classifier accuracy scores versus the average number of events produced by a given encoding scheme per the VMP sensor. Heat maps representing the estimated density were min-max normalized separately.

system scales linearly with the number of channels, regardless of the chosen coding scheme. Additionally, doing so allows assessing the computational burden of introducing an additional VMP sensor in the solution. The results presented in Figure 5.6 show that the number of events is not the sole determining factor of classification performance. Concretely, send-on-delta models exhibited similar levels of performance regardless of the number of events produced by the encoding scheme, whereas the LIF parameter sets generating event sequences of similar length can lead to a drastically different k-NN classifier performance.

In order to determine which set of encoding parameters is to be used in the downstream analysis, we first note that the results of the Bayesian optimization process over N different data splits form several independent ranked lists $Q = \{q_1, q_2, \ldots, q_N\}$. Within each ranking q a higher position (rank) is assigned to parameter sets achieving better k-NN classifier scores (ties are allowed). One strategy would be to select the parameter set that performs the best overall, across different data splits. However, the absolute value of the score depends on not only the chosen parameters – which may not have been selected by the Bayesian optimization procedure in other splits – but also on the stratified data sample. Additionally, it is preferable to choose the parameter set that performs reasonably well in all data samples. Thus, a selection strategy based on the top-performing parameters across all rankings is ill-advised.

For these reasons we devise the *local weighted median selection strategy* – a parameter selection strategy that considers the local (i.e., within-ranking) rank of each result over all data splits (Algorithm 5.1). The resulting parameter sets selected according to this strategy are presented in Table 5.1 which also includes the stratified 10-fold cross-validated k-NN classifier performance for data splits preprocessed according to the chosen encoding schemes. According to these results, the level-crossing encoding is slightly better for the VMP vehicle type classification task than the alternatives. Importantly, each signal prior to the event encoding has on average about 1590 samples, meaning that the analyzed encoding that generates the largest number of samples – send-on-delta – produces a data representation that uses about 97.8% fewer samples than the original.

Algorithm 5.1: Local weighted median selection strategy for the parameter sets found by the Bayesian optimization procedure.

```
Input: $Q$ # set of all ranking lists $\{q_1, q_2, \ldots, q_N\}$ for $N$ data splits
Output: $\mathbf{p}^*$ # selected parameter vector
Begin:
        $\mathbf{p}^* \leftarrow []$
        $\mathbf{P}_\alpha \leftarrow []$ # initialize a buffer for parameters
        $\mathbf{r}_\alpha \leftarrow []$ # initialize a buffer for rank reciprocal scores
        For $n = 1, \ldots, N$:
                $(\mathbf{P}, \mathbf{s}) \leftarrow q_n$ # unpack the ranking into parameters $\mathbf{P}$ and scores $\mathbf{s}$
                $\mathbf{r} \leftarrow 1/\text{rank}(\mathbf{s})$ # compute rank reciprocal for each score
                # update the buffers
                $\mathbf{P}_\alpha \leftarrow [\mathbf{P}_\alpha, \mathbf{P}]$
                $\mathbf{r}_\alpha \leftarrow [\mathbf{r}_\alpha, \mathbf{r}]$
        End For
        $\mathbf{w}_\alpha = \mathbf{r}_\alpha / \sum \mathbf{r}_\alpha$ # normalize the rank reciprocal values to obtain weights
        $r^* \leftarrow \text{weighted median}(\mathbf{r}_\alpha, \mathbf{w}_\alpha)$ # compute the weighted median rank reciprocal
        $(I, C) \leftarrow \text{shape}(\mathbf{P}_\alpha)$ # get the number of rows and columns
        $J \leftarrow \left\{ i : \mathbf{r}_\alpha^{[i]} \equiv r^* \right\}$ # find the rows that corresponds to $r^*$
        For $c = 1, \ldots, C$:
                # compute the median parameter value over rows in $J$
                # and update the buffer
                $\mathbf{p}^* \leftarrow \left[\mathbf{p}^*, \text{median}\left(\mathbf{P}_\alpha^{[J,c]}\right)\right]$
        End For
Return $\mathbf{p}^*$
End
```

Finally, we analyzed the expected number of events produced by the chosen encoding schemes for each VMP sensor separately. Together with the presented classification results it allows us to reason about the coding efficiency. Figure 5.7 summarizes our findings. To visualize these distributions we use letter-value plots [256] – an extension of the classical boxplot that more accurately represents distribution tails for large data samples. It can be observed that there is a clear difference between the empirical distributions of the number of event sequences produced for the two standard inductive-loops (IL1 and IL3) and the two slim loops (IL2 and IL4), regardless of the signal-to-spike encoding scheme. Furthermore, the selected set of parameters for the level-crossing and send-on-delta encoding schemes produce event sequences with a similar overall number of events, with send-on-delta sequences having

Table 5.1: Signal-to-spike encoding parameters chosen according to the local weighted median selection strategy. The multi-neuron van Rossum distance parameters used to evaluate the k-NN classifier are not presented.

Encoding type	**Chosen parameters**	**Number of events per VMP sensor**	**stratified 10-fold cross-validated k-NN performance**	
			Accuracy	**F_1-score**
level-crossing	$L = 12$	29.484 ± 8.996	0.912 ± 0.011	0.907 ± 0.016
send-on-delta	$\Delta = 0.06$	36.536 ± 9.040	0.910 ± 0.014	0.909 ± 0.019
LIF	$\tau_{\text{int}} = 0.1$ $\tau_{\text{leak}} = 0.5$ $V_{\text{thr}} = 0.2$	8.317 ± 5.673	0.905 ± 0.011	0.900 ± 0.015

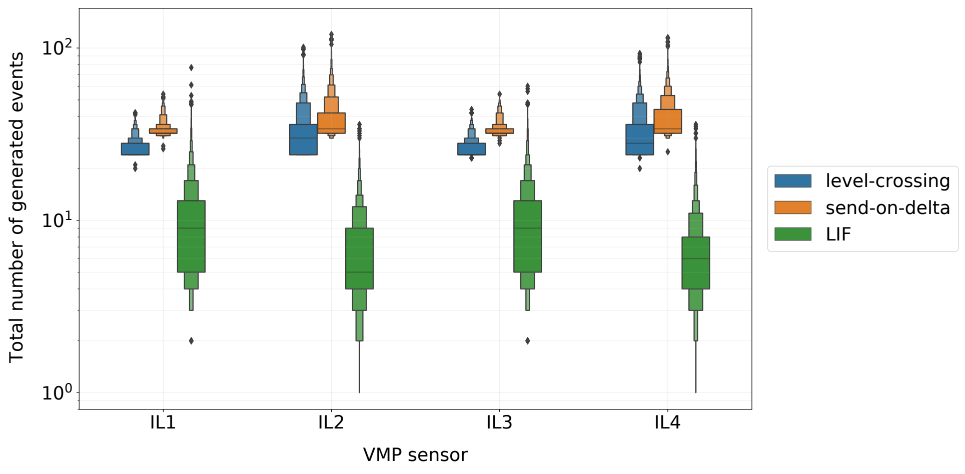


Figure 5.7: Letter-value plots of the total number of events produced by signal-to-spike encoding schemes selected according to the local weighted median strategy for each input VMP sensor.

slightly more events on average. Conversely, the LIF encoding generates significantly fewer events with the median number of events being about 3x smaller for the IL1-IL3 sensors and about 6x smaller for the IL2-IL4 sensors. Moreover, it is capable of adapting to the original signal's variability, which is expressed by the wide range of the number of produced events over all data samples. Given that all three classifiers presented in Table 5.1 have similar performance with respect to the classification accuracy, the LIF encoding seems to be more efficient in terms of the amount of information encoded by a single event.

5.2.2 Vehicle type identification with the MIMO SNN

5.2.2.1 Preliminaries

The set of parameters identified in the previously presented analysis were used to encode the VMP signals into the spiking domain prior to training the MIMO SNN models on the vehicle type identification task. Figure 5.8 shows an example of spike codes obtained for a VMP time series. Each event-sampling scheme produces a different number of distinct event types for every VMP sensor, which means that the number of spiking neurons in the SNN input layer differs between the coding schemes. In all experiments we used the same C-64-128-128-6 architecture, where the number of input neurons C depends entirely on the encoding ($C = 48$ for level-crossing, $C = 8$ for send-on-delta, and $C = 4$ for LIF). This produces a small network with $25344 + 64C$ parameters. Do note that the MIMO SNN model capacity is effectively higher than indicated solely by the number of parameters due to the repeated firing of the IF neurons parameterized by the refractory period τ_{ref}.

All layers were time-aligned according to the methodology presented in Section 3.2.2. This is done to stabilize the training by reducing the magnitude of time-values of events in the network. This removes the redundant time-shift information associated with each subsequent layer waiting for the preceding layer to elicit its first event. Additionally, we introduce a *neuron response delay factor* τ_{delay} in the first hidden layer. This parameter designates the earliest event that is capable of eliciting a response in a given neuron. This is equivalent to making the neuron start the simulation in a refractory period of duration τ_{delay}. We posit that choosing a different τ_{delay} for each neuron in a layer causes them to observe slightly different event sequences, which circumvents the issue of the early events dictating the training progress[c]. Figure 5.9 presents a raster plot example for a network trained by setting nonzero values of τ_{delay} in the first layer neurons. The parameter was varied on a linear grid. We did not use neuron response delay factors in subsequent layers because the event sequences produced by the SNN have significantly fewer events than the network input, making the neuron specialization easier.

The signal preprocessing steps were identical to the ones presented in Section 5.2.1.1. We also used the same stratified 10-fold data split to perform the cross-validation. The MIMO SNN models were trained by minimizing

$$L_{\text{total}}(z, y) = \frac{1}{N} \sum_{n=1}^{N} L_n(z, y) + \gamma R^*_{\text{spiking}} + \lambda R_{L_2}\,, \tag{5.8}$$

where γ is the synaptic regularization parameter for the spike-firing penalty R^*_{spiking} (3.67), and λ controls the strength of the L_2 regularization term R_{L_2}. All models were trained with the RMSprop optimizer with a learning rate of 0.001 over 2500 iterations with a batch size

[c]Let us demonstrate this phenomenon through a thought experiment. Assume that all neurons start in the resting state, i.e., $\tau_{\text{delay}} = 0$ for all neurons. Additionally, assume that there is a neuron that in response to the training procedure specializes in responding to a pattern that occurs relatively late in the input event sequence. Then, this neuron needs to either have synaptic weights smaller than other neurons in the network (so that it does not respond too early relative to the pattern it is detecting), or it has synaptic weights that are comparable to other neurons, but some of the spikes it generates are uninformative (because they were not made in response to patterns it is tuned for). This situation occurs whenever a neuron observes events that are ultimately not important to the message it tries to convey, such as in response to early events of the input event sequence.

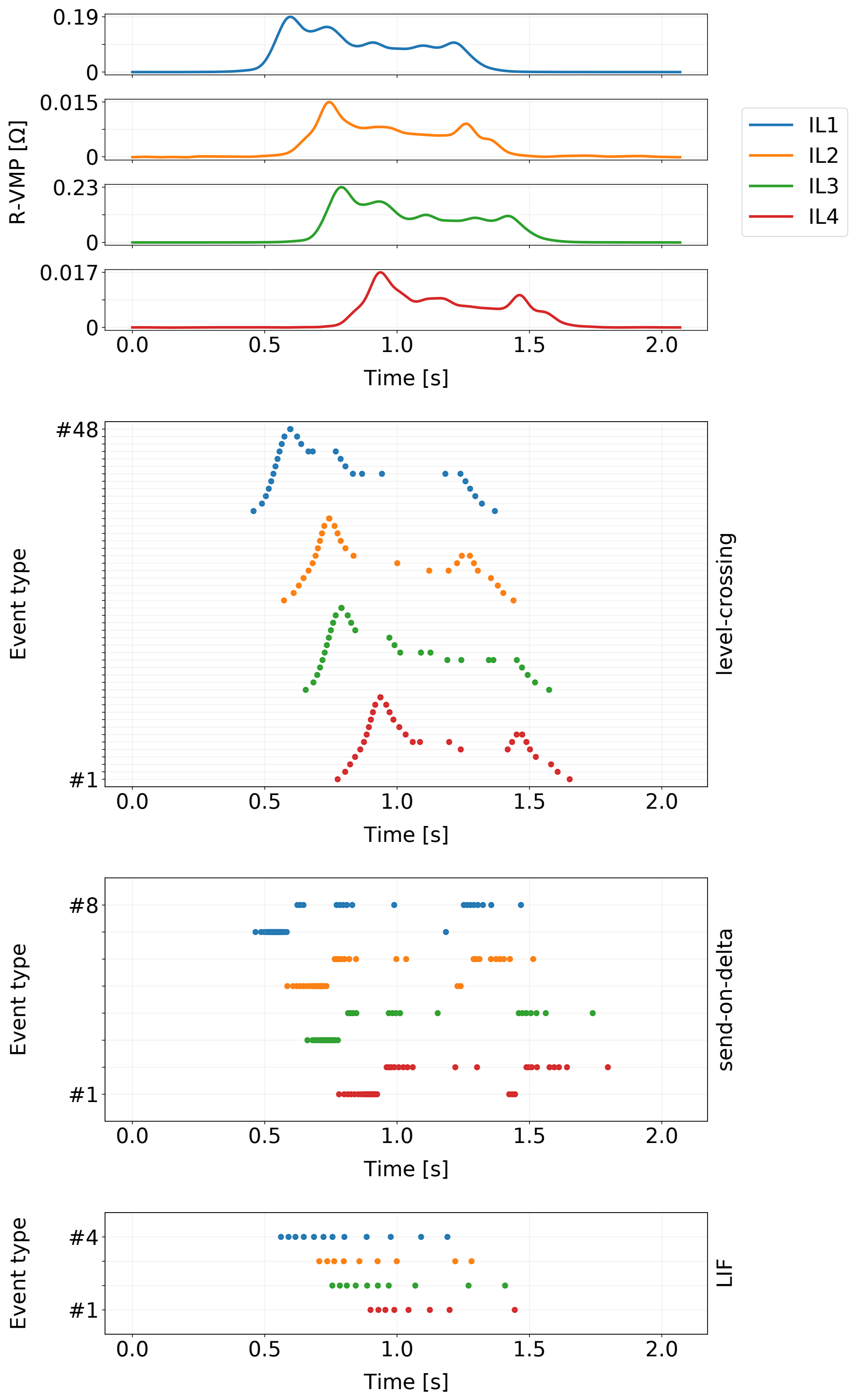


Figure 5.8: An example of a VMP signal and its different event domain representations. Curves and raster plot points are color-coded to signify the correspondence between the input VMP channel and the output event sequences. The vertical axis for the three encoding raster plots represents different event types: 48 for level-crossing, 8 for send-on-delta, and 4 for the LIF encoding. These spike trains are presented at the SNN input layer.

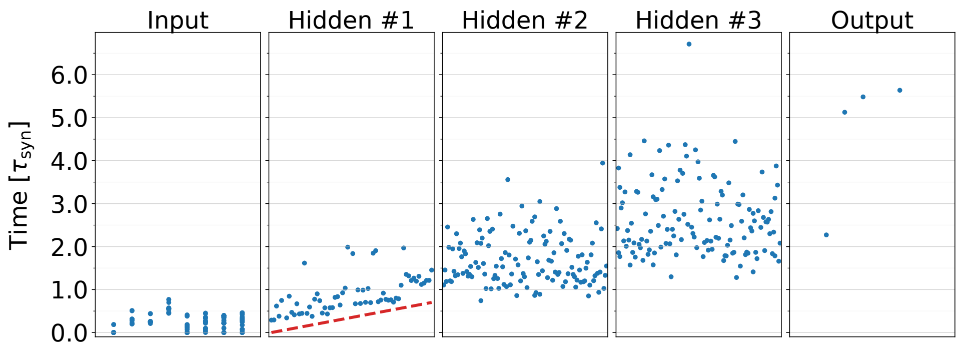


Figure 5.9: Spike raster plot for a MIMO SNN model trained with a nonzero neuron response delay factor τ_{delay} in the first hidden layer. The dashed line denotes the value of τ_{delay} assigned to each neuron in the layer. It is evident that neurons with a smaller response delay elicit their first response earlier than those with a larger τ_{delay}.

of 50 examples. We set $\lambda = 10^{-5}$ in all experiments. Similarly to training the models on the CREDO data (Section 4.4.3), the regularization parameter γ was initially set to a large value of 10^5 in order to guide the model towards a solution that is capable of propagating event throughout the entire network. Then, after η iterations, the value of γ was decreased to 10^3 in order to increase the relative importance of solving the classification task.

We used Hyperopt to optimize the parameters that control how the information is processed by the network. Specifically, the following Bayesian optimization search space was defined:

- the refractory period: $\log(\tau_{\text{ref}}) \sim \mathcal{U}(-0.6, 0)$,
- the longest neuron response delay τ_{delay} for a linearly-spaced grid: $\tau_{\text{max delay}} \sim \mathcal{N}(t_{\text{avg}}, 1)$ subject to $\tau_{\text{max delay}} \geq 0$, where t_{avg} is the average event time of input sequences in the current training dataset,
- the number of iterations to train with a larger spike-firing penalty: $\eta \sim \mathcal{U}\{300, 800\}$.

This parameter-selection process was run for 30 optimization iterations on a single data split of the 10-fold cross-validation (the same for all tested encoding types). Afterwards, five best-performing sets of parameters were selected and used to train models on all remaining data splits[d]. In total we optimized the weights of 225 models over three spike encoding schemes.

5.2.2.2 Analyzing the results

Figure 5.10a summarizes the accuracy scores of models trained as part of the Bayesian optimization procedure on a single data split of the stratified 10-fold cross-validation. On average the LIF encoding performed worse than the alternatives. Recall that all three encoding schemes parameterized according to the local weighted median selection strategy had similar accuracy

[d]When selecting the top-5 Hyperopt models we break ties according to the following heuristic rule: prioritize lower $\tau_{\text{max delay}}$, higher τ_{ref}, and lower η (in that order of importance). This corresponds to models that generate fewer events and respond faster.

scores of the k-NN classifier of about 0.908 (Table 5.1). This means that the Bayesian optimization procedure failed to find a LIF model that was better than its own baseline. Conversely, the level-crossing and send-on-delta models did perform better than their respective k-NN classifier, with a similar median accuracy between the two. Note that the presence of outliers in the lower range of accuracy scores indicates that incorrectly choosing the hyperparameters during training significantly degrades the performance of the final model. Nevertheless, it is clear that running the optimization procedure over many steps can result in a high-performing model, provided that the signal-to-spike encoding is chosen correctly.

Secondly, Figure 5.10b presents the accuracy scores for models trained with the top-5 best-performing parameters chosen by Hyperopt, separately for each encoding type and each cross-validation data split. In doing so we analyzed the robustness of the classifiers trained with these hyperparameters with respect to different input data. Here we observe several differences between models operating on differently-encoded data. Once again, the LIF models achieved much lower absolute performance scores than the other encoding types. Moreover, both the level-crossing and LIF models evaluated on data splits #2-10 indicated worse classification accuracy than that of the reference split #1 (used to find the top-5 hyperparameter sets). On the other hand, the send-on-delta models evaluated on data splits #3, #7, #8 and #10 all achieved similar levels of performance as in scenario #1. This indicates that the parameters chosen for the send-on-delta encoding are more robust with respect to the training data choice (albeit it also exhibits an extreme performance outlier in data split #9, compared to all trained models), whereas the level-crossing and LIF encoding schemes seem to require careful tuning on the available data.

Lastly, in Table 5.2 we present the top-5 parameter sets for the three encoding types, as well as the stratified 10-fold cross-validated performance metrics for models trained with these hyperparameters. We note that the top-5 parameter sets selected for the send-of-delta and LIF models are much more similar to one another than those selected for the level-crossing encoding. This implies that for this specific task 30 steps of the Bayesian optimization process are enough to converge to a good-enough (local minimum) solution for these two model types, but not enough to do so for the level-crossing scheme. Interestingly, the selected values of $\tau_{\text{max delay}}$ were smaller for the send-on-delta and LIF models when compared to level-crossing. This might suggest that a nonzero $\tau_{\text{max delay}}$ is beneficial when the number of neurons in the input layer is large (as in level-crossing), rather than when they generate a large number of events (as in send-on-delta). Finally, recall that the prior on the number of iterations to train with a larger spike-firing penalty η was $\mathcal{U}\{300, 800\}$. Almost all of the selected values of η ended up being on the higher-end, which indicates that training the model with a relatively smaller weight given to the task-specific loss for longer leads to a better performance of the

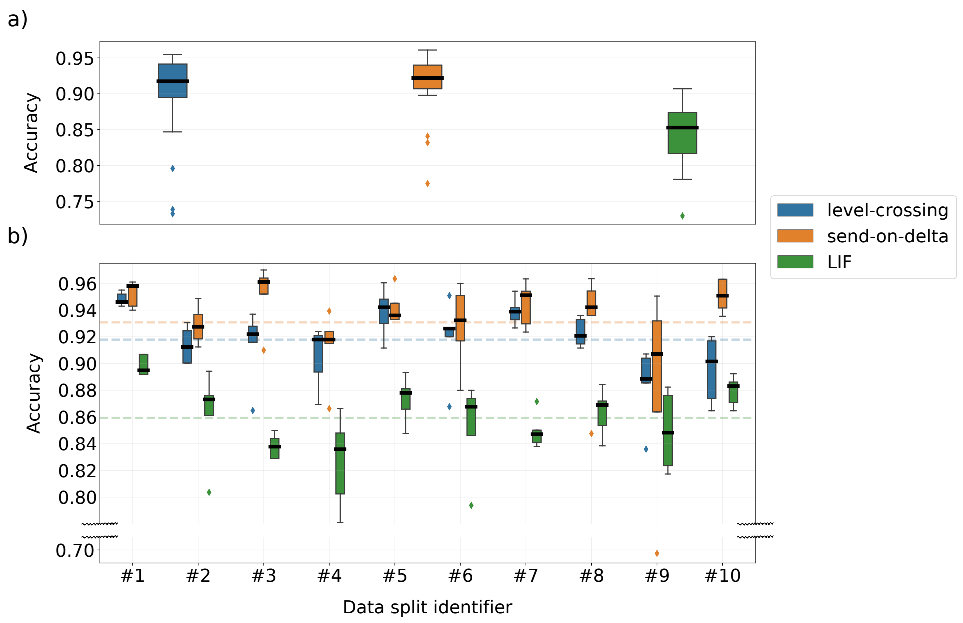


Figure 5.10: a) Summary of the SNN model performance for models trained with Hyperopt (30 optimization iterations on a single data split of the stratified 10-fold cross-validation). b) Accuracy scores of the SNN models trained with top-5 parameter sets found by the Bayesian optimization process on data split #1. The dashed lines denote the average score over all data splits.

final model.

When compared to the performance achieved by the baseline k-NN classifiers (Table 5.1), it can be seen that training the SNN leads to a reduction in error rates in the vehicle type classification task, provided that the signal-to-spike encoding scheme is selected properly. Undoubtedly, processing an example through the SNN is more computationally demanding than computing the multi-neuron van Rossum distance between a pair of examples. However, for a k-NN-based system the actual prediction time depends on the size of the reference database, which makes this solution scale poorly with the number of training examples.

Taking the above into consideration, the send-on-delta encoding emerged as the best event-triggered sampling scheme for the VMP-based vehicle type classification task. These models achieved the best stratified 10-fold cross-validation performance, they were more robust with respect to the different data splits, and the Bayesian optimization procedure was able to converge to a locally optimal solution within the constrained number of optimization steps. Note that this cannot be attributed solely to a large average number of events generated by the encoding, given that it is quite similar for the level-crossing event sequences (Figure 5.7). On the other hand, we found that machine learning solutions using level-crossing encoding are simpler to design than the tested alternatives: it is easier to assess the number of events of a new type when considering the addition of a new amplitude level to sample the signal at, rather than to estimate how a change in the send-on-delta parameter might impact the spike count distribution

Table 5.2: Summary of the top-5 parameter sets found by the Bayesian optimization process used to train the SNN models for the respective spike encoding type.

Encoding type	**Number of events per VMP sensor**	**Top-5 parameter sets**			**stratified 10-fold cross-validated SNN performance**	
		$\tau_{\text{max delay}}$	τ_{ref}	η	**Accuracy**	**F_1-score**
level-crossing	29.484 ± 8.996	0.9	1.0000	700	0.926 ± 0.019	0.929 ± 0.018
		1.2	0.2512	400	0.917 ± 0.024	0.920 ± 0.027
		0.9	0.6310	600	0.915 ± 0.035	0.917 ± 0.033
		0.9	0.2512	700	0.919 ± 0.024	0.922 ± 0.024
		0.7	0.7943	500	0.913 ± 0.028	0.919 ± 0.028
send-on-delta	36.536 ± 9.040	0.3	0.3981	600	0.937 ± 0.030	0.928 ± 0.031
		0.4	0.5012	600	0.926 ± 0.036	0.913 ± 0.044
		0.4	**0.5012**	**700**	**0.946 ± 0.018**	**0.937 ± 0.021**
		0.4	0.5012	800	0.937 ± 0.026	0.926 ± 0.027
		0.0	0.3162	500	0.909 ± 0.073	0.907 ± 0.068
LIF	8.317 ± 5.673	0.0	0.5012	300	0.861 ± 0.024	0.859 ± 0.027
		0.2	0.5012	600	0.868 ± 0.031	0.856 ± 0.040
		0.0	0.3981	700	0.854 ± 0.032	0.852 ± 0.038
		0.0	0.5012	400	0.859 ± 0.024	0.858 ± 0.024
		0.0	0.5012	700	0.855 ± 0.031	0.851 ± 0.038

of existing event types. Overall, the existence of such trade-offs suggests that the pros and cons of different signal-to-spike conversion schemes must be weighed when developing a solution that satisfies the design constraints.

5.3 Summary

In this Chapter we described a novel methodology for training event-centric machine learning models on multivariate time series data. These sequences are processed by various event-triggered sampling schemes in order to obtain a representation understood by a spiking neural network. Successfully training a model to solve a vehicle type identification task proves that the proposed time-to-first-spike MIMO SNN can be applied to data that is not naturally represented in the event data domain. Together with the previously-described results on event sequence (Chapter 3) and static image (Chapter 4) data, this validates the applicability of our model to different scenarios encountered in practical machine learning problems.

We proposed to split the signal-to-spike encoding scheme selection from the actual model training and evaluation on the target task. This provided several benefits over the alternative. Perhaps most importantly, it allowed to reduce the number of Bayesian optimization steps in the model hyperparameter search, given that training the SNN is by far the most time-consuming part of the experimental setup. Secondly, it made the model performance comparison analysis simpler by disentangling the impact of different input encoding parameters on the performance of the final model. While it can be argued that, ideally, the encoding parameters should be optimized together with the SNN hyperparameters, such approach was deemed too impractical in this study. Nevertheless, this direction is worth further investigation in the future.

Three different event-triggered sampling schemes were evaluated in this Chapter: level-crossing, send-on-delta and the Leaky Integrate-and-Fire time encoding machine. In contrast to previous studies that mostly focus on reconstructing the signal from the generated events, we were interested in understanding how the choice of the encoding impacts a machine learning solution. We found that the accuracy of the k-NN classifier on the vehicle classification task can be as low as about 0.775 and as high as about 0.950. Such a wide range of obtained results indicates that correctly setting the encoding scheme parameters is of paramount importance. Unfortunately, this also means that choosing the parameter set to be used in the downstream analysis is unintuitive, especially considering the performance variability introduced by fitting the model on different data splits. We proposed a *local weighted median selection strategy* that constructs a set of model performance ranking lists for each data split and attempts to find the median performance across different hyperparameter settings. It allowed us to settle on a single set of parameters for each encoding, with the average accuracy ranging from 0.905 (for the LIF encoding) to 0.912 (for send-on-delta). This can be considered a fair baseline for the downstream SNN models, given the comparable cross-validated performance of the three k-NN classifiers.

For the SNN trained on data transformed according to the selected event-triggered sampling schemes, the send-on-delta models emerged as the best-performing group of networks. Not only did they achieve the best overall cross-validated performance (average accuracy of 0.946), but also they were more robust with respect to the different data splits, and the Bayesian optimization procedure was able to converge to a locally optimal solution within the constrained number of optimization steps. However, they did so while exhibiting the highest average number of events generated by the sampling scheme, generating slightly more events than the second-worst, the level-crossing encoding. We found that the SNN trained on time series data sampled according to the LIF scheme perform much worse than the tested alternatives. We posit that three factors have played a major role in this outcome: a small number of distinct event types produced by the encoding (only one per input sensor), few events observed by the network (about 3-6x fewer than the alternative schemes on average), and multiple encoding parameters to optimize for (in contrast to only one parameter for send-on-delta and level-crossing). Verifying whether this assessment is correct – that these factors induce poor SNN classifier performance regardless of the encoding type – warrants further investigation. Overall, our findings suggest the existence of a trade-off between model performance with respect to classification metrics and its energy-efficiency in terms of the number of generated events. These factors must be taken into consideration when designing an end-to-end event-centric machine learning solution.

The quad inductive loop system for the VMP signal acquisition considered in this study provides significantly more data than was used to train the SNN models. Specifically, we took a subset of all channels, leaving only four of them for further processing (the R-VMP signals registered by sensors at their lowest excitation frequency). We aim to relax this constraint in future research, which might lead to an improvement in model performance on the vehicle type classification task. Another interesting research prospect is to depart from the classification context and instead apply the Siamese SNN methodology presented in Chapter 4 to solve a vehicle re-identification task. A prerequisite to such analysis is for the source data to contain information about the specific vehicle instances, i.e., unique to each and every vehicles in the dataset. An SNN model for vehicle re-identification could be used in real-time traffic flow monitoring.

Chapter 6

Conclusions and Future Work

Modern IT systems and services put an increasing emphasis on gathering user activity behavior data and monitoring system internal state by logging important event occurrence data. With the rising volume of such data there is a pressing need for the development of scalable machine learning solutions capable of analyzing, assessing, and making predictions from it. This need is further exacerbated by the rising interest in event-triggered sampling schemes for analog signal acquisition. These sampling methods provide benefits (over the typical uniform sampling approach) such as data sparsity, wide dynamic range, and low data transmission delays. Thus, event-centric machine learning methods must also be capable of processing data originating from event sensors.

This thesis explored two different approaches to machine learning on event data: a classical one based on the theory of point processes, and a recently developed solution using artificial spiking neural networks. Both methods were selected in order to address identified knowledge gaps that hinder their applicability to the aforementioned tasks.

We found that presently there is little research on the topic of point process sequence classification that focuses on the bounds on model performance with respect to the observation period length and training dataset size. To address this, we developed a novel methodology based on the Bayes theory of classification for the two-class problem for a class of spike train data characterized by non-random point process intensity functions. We examined the optimal Bayes rules, as well as a general class of plug-in classifiers that are capable of solving the classification task. We found that for a large class of intensity functions the Bayes risk converges to zero as the event observation interval increases. This reveals the perfect classification property of the Bayes rule with respect to a long observation time interval. Furthermore, the convergence of a data-based kernel classifier to the Bayes risk in terms of the training dataset size was also established. These findings are supported by a finite sample study on simulated data. Lastly, we provided empirical evidence that for the proposed kernel classifier the negative impact of boundary effects is significant only when the observation window length is comparable to the kernel bandwidth.

We also made several contributions to the SNN training framework. An existing single-spike time-to-first-spike SNN was used as the basis of our work. This model considers the network state only at time-instants related to event occurrence, which addresses one of the research questions related to training the SNN models posed in this dissertation. Several key limitations of this model were identified and addressed during the course of this study. We showed how the incumbent SNN layer computation can be vectorized to obtain the layer outputs in a single pass over model inputs. This reduces the time required to compute the result by three orders of magnitude when compared to the original algorithm description, allowing it to be used in modern deep learning frameworks for scalable machine learning. Additionally, a modification that makes the model capable of processing multiple spikes at its input and eliciting multiple spikes at its output was also presented. Our formulation is compatible with the aforementioned vectorized computation by expressing the algorithm in terms of iteratively calculating successive output spikes, which stands in contrast to other approaches that simulate the state of the entire network over a finite time window with a fixed time step.

Moreover, some key challenges associated with the training procedure of the time-to-first-spike SNN model were identified. We showed that the network exhibits numerical instability when trained with event times of large magnitude. To address this effect we proposed to either shift the events back to $t = 0$ before each layer computation, or to transform the events themselves using a bijective function that shrinks the data domain. Another identified issue with the training dynamics is the slow convergence of the loss function when the synaptic regularization parameter is kept constant throughout the entire training procedure. A solution that scales the value of this parameter based on how well the current network performs on the underlying machine learning task was proposed and evaluated.

Additionally, we introduced a Siamese SNN that adapts the Siamese network concept to the spiking domain such that it is trained end-to-end directly in the event space. This differs from other works that train a nonspiking Siamese network and then convert it to the spiking domain. In our loss function we used the Earth Mover's Distance (a special case of the Wasserstein metric) as the similarity measure to train the model. Note that any distance function that is applicable to spike trains can be used in this context, specifically those commonly utilized in computational neuroscience research.

Following-up on the aforementioned theoretical and empirical findings considered to be the main research contributions of this thesis, the models under study were applied to real event data. We considered the machine learning tasks of Twitter bot detection, the CREDO image artefact rejection, and the VMP vehicle type classification. Each case-study was described separately, highlighting the research methodology and how the encountered problem-specific challenges can be overcome. In doing so the applicability of these methods was assessed, both on the natural event data, as well as on datasets obtained by conversion into the spiking domain. The models trained on the latter dataset type were additionally analyzed with respect to the impact of the conversion scheme on the model performance and properties. Overall, the proposed methods achieved good performance on all presented research problems. In each study we found evidence of an accuracy-sparsity trade-off exhibited by the proposed SNN. This highlights the importance of evaluating the impact of signal-to-spike conversion schemes on the machine learning model performance prior to finalizing the design of the entire system.

There are several directions of further research worth considering. For the point process-based approach, a seemingly obvious idea is to consider a multi-class classification problem or classifying multivariate event sequences. This would greatly extend the applicability of this solution. Additionally, the presented algorithm uses a fixed kernel bandwidth during the shape function estimation step. A natural extension of this approach is to select different bandwidths at different timescales, given that each sequence can be arbitrarily long and the density of events may vary with time. Moreover, taking into consideration a larger class of point processes would be of great interest, including the self-exciting Hawkes process and marked point processes.

For the SNN model one unaddressed limitation is that the IF neuron's membrane voltage does not gradually return to the resting state in the absence of events once the neuron is excited by a stimulus. Incorporating a voltage decay constant into the model – and modifying the entire layer computation accordingly – would allow continuous monitoring of long-running processes without explicitly resetting the state of the network after every example. Furthermore, it might be interesting to revisit the vectorized implementation of the spiking layer introduced in this thesis. The main drawback of the presented solution is the inability to consider only the informative events in its computation. Specifically, non-events introduced by the tensor-based approach contribute to the overall computation while ultimately having no impact on the generated set of spikes. Thus a different approach is needed to scale the algorithm to even larger sets of spiking data. Solving this issue seems to also be a prerequisite to adapting the proposed SNN layer to construct its convolutional counterpart. Alongside the aforementioned research avenues concerning signal propagation through a spiking layer, one might also consider designing different task-specific loss functions of the SNN in order to adapt existing deep learning models to the spiking context.

Finally, both of the proposed models will struggle with updating their predictions as new events are observed. New data influence the shape function estimate of the kernel classifier if they occur inside the interval delineated by the kernel bandwidth centered around a past event. Similarly, a new event might change the causal set, impacting the signal propagation inside the SNN. These problems limit the applicability of our methods to offline prediction with a bounded observation interval. It might be beneficial to explore modifications that address this issue for both algorithms.

Bibliography

[1] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*, Cambridge University Press, 2014.

[2] O. Shchur, A. C. Türkmen, T. Januschowski, and S. Günnemann, "Neural temporal point processes: A review," *arXiv preprint arXiv:2104.03528*, 2021.

[3] Z. Chen, L. D. Van Khoa, E. N. Teoh, et al., "Machine learning techniques for anti-money laundering (AML) solutions in suspicious transaction detection: a review," *Knowledge and Information Systems*, vol. 57, no. 2, pp. 245–285, 2018.

[4] P. Ładyżyński, K. Żbikowski, and P. Gawrysiak, "Direct marketing campaigns in retail banking with the use of deep learning and random forests," *Expert Systems with Applications*, vol. 134, pp. 28–35, 2019.

[5] M. Mazza, S. Cresci, M. Avvenuti, et al., "RTbust: Exploiting temporal patterns for botnet detection on Twitter," in *Proceedings of the 10th ACM Conference on Web Science*, 2019, pp. 183–192.

[6] Y. Shen, E. Mariconti, P. A. Vervier, and G. Stringhini, "Tiresias: Predicting security events through deep learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 592–605.

[7] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1285–1298.

[8] B. Shickel, P. J. Tighe, A. Bihorac, and P. Rashidi, "Deep EHR: A survey of recent advances in deep learning techniques for electronic health record (EHR) analysis," *IEEE Journal of Biomedical and Health Informatics*, vol. 22, no. 5, pp. 1589–1604, 2018.

[9] A. Rajkomar, E. Oren, K. Chen, et al., "Scalable and accurate deep learning with electronic health records," *npj Digital Medicine*, vol. 1, no. 1, pp. 18, 2018.

[10] S. M. Lauritsen, M. E. Kalør, E. L. Kongsgaard, et al., “Early detection of sepsis utilizing deep learning on electronic health record event sequences,” *Artificial Intelligence in Medicine*, vol. 104, pp. 101820, 2020.

[11] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*, The MIT Press, 2001.

[12] H. G. Rey, C. Pedreira, and R. Quian Quiroga, “Past, present and future of spike sorting techniques,” *Brain Research Bulletin*, vol. 119, pp. 106–117, 2015.

[13] W. Bialek, F. Rieke, R. van Steveninck, and D. Warland, “Reading a neural code,” in *Advances in Neural Information Processing Systems*, 1989, vol. 2.

[14] S.-C. Liu, B. Rueckauer, E. Ceolini, et al., “Event-driven sensing for efficient perception: Vision and audition algorithms,” *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 29–37, 2019.

[15] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbruck, “Retinomorphic event-based vision sensors: Bioinspired cameras with spiking output,” *Proceedings of the IEEE*, vol. 102, no. 10, pp. 1470–1484, 2014.

[16] S.-C. Liu, A. van Schaik, B. A. Minch, and T. Delbruck, “Asynchronous binaural spatial audition sensor with $2 \times 64 \times 4$ channel output,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 8, no. 4, pp. 453–464, 2014.

[17] A. Vanarse, A. Osseiran, and A. Rassau, “An investigation into spike-based neuromorphic approaches for artificial olfactory systems,” *Sensors*, vol. 17, no. 11, 2017.

[18] B. Ward-Cherrier, N. Pestell, and N. F. Lepora, “NeuroTac: A neuromorphic optical tactile sensor applied to texture recognition,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 2654–2660.

[19] E. Doutsi, L. Fillatre, M. Antonini, and P. Tsakalides, “Dynamic image quantization using leaky integrate-and-fire neurons,” *IEEE Transactions on Image Processing*, vol. 30, pp. 4305–4315, 2021.

[20] M. Saeed, Q. Wang, O. Märtens, et al., “Evaluation of level-crossing ADCs for event-driven ECG classification,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 15, no. 6, pp. 1129–1139, 2021.

[21] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, “Converting static image datasets to spiking neuromorphic datasets using saccades,” *Frontiers in Neuroscience*, vol. 9, 2015.

[22] M. Pawlak, M. Pabian, and D. Rzepka, “Asymptotically optimal nonparametric classification rules for spike train data,” in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023, pp. 1–5.

[23] M. Pabian, D. Rzepka, and M. Pawlak, “Supervised training of Siamese spiking neural networks with Earth Mover’s Distance,” in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 4233–4237.

[24] M. Pabian, D. Rzepka, Ł. Bibrzycki, and M. Pawlak, “Differentiating signal from artefacts in cosmic ray detection: Applying Siamese spiking neural networks to CREDO experimental data,” *Measurement*, vol. 220, pp. 113273, 2023.

[25] L. Devroye, L. Györfi, and G. Lugosi, *A Probabilistic Theory of Pattern Recognition*, Springer New York, 1996.

[26] H. Fanaee-T and J. Gama, “Event labeling combining ensemble detectors and background knowledge,” *Progress in Artificial Intelligence*, vol. 2, no. 2, pp. 113–127, 2014.

[27] W. Greblicki, “Asymptotically optimal pattern recognition procedures with density estimates,” *IEEE Transactions on Information Theory*, vol. 24, no. 2, pp. 250–251, 1978.

[28] D. R. Cox and P. A. W. Lewis, *The Statistical Analysis of Series of Events*, Springer Dordrecht, 1966.

[29] A. G. Hawkes, “Spectra of some self-exciting and mutually exciting point processes,” *Biometrika*, vol. 58, no. 1, pp. 83–90, 1971.

[30] D. R. Cox, “Some statistical methods connected with series of events,” *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 17, no. 2, pp. 129–157, 1955.

[31] S. Byers and A. E. Raftery, “Nearest-neighbor clutter removal for estimating features in spatial point processes,” *Journal of the American Statistical Association*, vol. 93, no. 442, pp. 577–584, 1998.

[32] D. C. I. Walsh and A. E. Raftery, “Classification of mixtures of spatial point processes via partial Bayes factors,” *Journal of Computational and Graphical Statistics*, vol. 14, no. 1, pp. 139–154, 2005.

[33] T. Rajala, C. Redenbach, A. Särkkä, and M. Sormani, “Variational Bayes approach for classification of points in superpositions of point processes,” *Spatial Statistics*, vol. 15, pp. 85–99, 2016.

[34] C. Redenbach, A. Särkkä, and M. Sormani, “Classification of points in superpositions of Strauss and Poisson processes,” *Spatial Statistics*, vol. 12, pp. 81–95, 2015.

[35] M. Lukasik, P. K. Srijith, D. Vu, et al., “Hawkes processes for continuous time sequence classification: an application to rumour stance classification in Twitter,” in *54th Annual Meeting of the Association for Computational Linguistics*, 2016, pp. 393–398.

[36] S. Liu and M. Hauskrecht, “Event outlier detection in continuous time,” in *Proceedings of the 38th International Conference on Machine Learning*, 2021, vol. 139, pp. 6793–6803.

[37] J. D. Victor and K. P. Purpura, “Metric-space analysis of spike trains: theory, algorithms and application,” *Network: Computation in Neural Systems*, vol. 8, no. 2, pp. 127–164, 1997.

[38] K. E. Tranbarger Freier and F. P. Schoenberg, “On the computation and application of prototype point patterns,” *The Open Applied Informatics Journal*, vol. 4, no. 1, 2010.

[39] J. Mateu, F. P. Schoenberg, D. M. Diez, et al., “On measures of dissimilarity between point patterns: Classification based on prototypes and multidimensional scaling,” *Biometrical Journal*, vol. 57, no. 2, pp. 340–358, 2015.

[40] A. Cholaquidis, L. Forzani, P. Llop, and L. Moreno, “On the classification problem for Poisson point processes,” *Journal of Multivariate Analysis*, vol. 153, pp. 1–15, 2017.

[41] K. Pawlasová and J. Dvořák, “Supervised nonparametric classification in the context of replicated point patterns,” *Image Analysis and Stereology*, vol. 41, no. 2, pp. 57–109, 2022.

[42] X. Rong and V. Solo, “On the error rate for classifying point processes,” in *2021 60th IEEE Conference on Decision and Control (CDC)*, 2021, pp. 120–125.

[43] K. Li, S. Li, and Y. Fu, “Early classification of ongoing observation,” in *2014 IEEE International Conference on Data Mining*, 2014, pp. 310–319.

[44] B.-N. Vo, N. Dam, D. Phung, et al., “Model-based learning for point pattern data,” *Pattern Recognition*, vol. 84, pp. 136–151, 2018.

[45] D. J. Daley and D. Vere-Jones, *An Introduction to the Theory of Point Processes*, Springer New York, 2003.

[46] Á. Gajardo and H.-G. Müller, “Cox point process regression,” *IEEE Transactions on Information Theory*, vol. 68, no. 2, pp. 1133–1156, 2022.

[47] L. Birgé and P. Massart, “Rates of convergence for minimum contrast estimators,” *Probability Theory and Related Fields*, vol. 97, no. 1, pp. 113–150, 1993.

[48] S. van de Geer, “Exponential inequalities for martingales, with application to maximum likelihood estimation for counting processes,” *The Annals of Statistics*, vol. 23, no. 5, pp. 1779–1801, 1995.

[49] M. Pawlak, M. Pabian, and D. Rzepka, "Bayes risk consistency of nonparametric classification rules for spike trains data," *arXiv preprint arXiv:2308.04796*, 2023.

[50] P. Diggle and J. S. Marron, "Equivalence of smoothing parameter selectors in density and intensity estimation," *Journal of the American Statistical Association*, vol. 83, no. 403, pp. 793–800, 1988.

[51] O. Aalen, "Nonparametric inference for a family of counting processes," *The Annals of Statistics*, vol. 6, no. 4, pp. 701–726, 1978.

[52] P. K. Andersen, Ø. Borgan, R. D. Gill, and N. Keiding, *Statistical Models Based on Counting Processes*, Springer New York, 2012.

[53] M. P. Wand and M. C. Jones, *Kernel Smoothing*, Chapman and Hall/CRC, 1994.

[54] W. Greblicki and M. Pawlak, *Nonparametric System Identification*, Cambridge University Press, 2008.

[55] T. Gasser and H.-G. Müller, "Kernel estimation of regression functions," in T. Gasser and M. Rosenblatt (Eds.), *Smoothing Techniques for Curve Estimation*, pp. 23–68, Springer Berlin Heidelberg, 1979.

[56] T. Gasser and H.-G. Müller, "Estimating regression functions and their derivatives by the kernel method," *Scandinavian Journal of Statistics*, vol. 11, no. 3, pp. 171–185, 1984.

[57] M. C. Jones, "Simple boundary correction for kernel density estimation," *Statistics and Computing*, vol. 3, no. 3, pp. 135–146, 1993.

[58] J. D. Hart and T. E. Wehrly, "Kernel regression when the boundary region is large, with an application to testing the adequacy of polynomial models," *Journal of the American Statistical Association*, vol. 87, no. 420, pp. 1018–1024, 1992.

[59] E. F. Schuster, "Incorporating support constraints into nonparametric estimators of densities," *Communications in Statistics - Theory and Methods*, vol. 14, no. 5, pp. 1123–1136, 1985.

[60] J. S. Marron and D. Ruppert, "Transformations to reduce boundary bias in kernel density estimation," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 56, no. 4, pp. 653–671, 1994.

[61] E. Ferrara, O. Varol, C. Davis, et al., "The rise of social bots," *Communications of the ACM*, vol. 59, no. 7, pp. 96–104, 2016.

[62] M. Del Vicario, G. Vivaldo, A. Bessi, et al., "Echo chambers: Emotional contagion and group polarization on Facebook," *Scientific Reports*, vol. 6, no. 1, pp. 37825, 2016.

[63] V. S. Subrahmanian, A. Azaria, S. Durst, et al., “The DARPA Twitter bot challenge,” *Computer*, vol. 49, no. 6, pp. 38–46, 2016.

[64] S. Cresci, R. Di Pietro, M. Petrocchi, et al., “The paradigm-shift of social spambots: Evidence, theories, and tools for the arms race,” in *Proceedings of the 26th International Conference on World Wide Web Companion*, 2017, pp. 963–972.

[65] S. Feng, H. Wan, N. Wang, et al., “TwiBot-20: A comprehensive Twitter bot detection benchmark,” in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 4485–4494.

[66] C. A. Davis, O. Varol, E. Ferrara, et al., “BotOrNot: A system to evaluate social bots,” in *Proceedings of the 25th International Conference Companion on World Wide Web*, 2016, pp. 273–274.

[67] K.-C. Yang, O. Varol, C. A. Davis, et al., “Arming the public with artificial intelligence to counter social bots,” *Human Behavior and Emerging Technologies*, vol. 1, no. 1, pp. 48–61, 2019.

[68] J. Rodríguez-Ruiz, J. I. Mata-Sánchez, R. Monroy, et al., “A one-class classification approach for bot detection on Twitter,” *Computers & Security*, vol. 91, pp. 101715, 2020.

[69] H. S. Dutta, A. Chetan, B. Joshi, and T. Chakraborty, “Retweet us, we will retweet you: Spotting collusive retweeters involved in blackmarket services,” in *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2018, pp. 242–249.

[70] Z. Miller, B. Dickinson, W. Deitrick, et al., “Twitter spammer detection using data stream clustering,” *Information Sciences*, vol. 260, pp. 64–73, 2014.

[71] A. Minnich, N. Chavoshi, D. Koutra, and A. Mueen, “BotWalk: Efficient adaptive exploration of Twitter bot networks,” in *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, 2017, pp. 467–474.

[72] M. BalaAnand, N. Karthikeyan, S. Karthik, et al., “An enhanced graph-based semi-supervised learning algorithm to detect fake users on Twitter,” *The Journal of Supercomputing*, vol. 75, no. 9, pp. 6085–6105, 2019.

[73] Q. Cao, M. Sirivianos, X. Yang, and T. Pregueiro, “Aiding the detection of fake accounts in large scale social online services,” in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012, pp. 197–210.

[74] S. N. Firdaus, C. Ding, and A. Sadeghian, “Retweet: A popular information diffusion mechanism – a survey paper,” *Online Social Networks and Media*, vol. 6, pp. 26–40, 2018.

[75] M. Giatsoglou, D. Chatzakou, N. Shah, et al., “Retweeting activity on Twitter: Signs of deception,” in *Advances in Knowledge Discovery and Data Mining*, 2015, pp. 122–134.

[76] L. G. Stewart, A. Arif, and K. Starbird, “Examining trolls and polarization with a retweet network,” in *Proceedings of WSDM workshop on Misinformation and Misbehavior Mining on the Web (MIS2)*, 2018.

[77] N. Chavoshi, H. Hamooni, and A. Mueen, “Identifying correlated bots in Twitter,” in *Social Informatics*, 2016, pp. 14–21.

[78] S. Gupta, P. Kumaraguru, and T. Chakraborty, “MalReG: Detecting and analyzing malicious retweeter groups,” in *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, 2019, pp. 61–69.

[79] J. Pan, Y. Liu, X. Liu, and H. Hu, “Discriminating bot accounts based solely on temporal features of microblog behavior,” *Physica A: Statistical Mechanics and its Applications*, vol. 450, pp. 193–204, 2016.

[80] S. Cresci, R. D. Pietro, M. Petrocchi, et al., “Social fingerprinting: Detection of spambot groups through DNA-inspired behavioral modeling,” *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 4, pp. 561–576, 2018.

[81] L. C. Aiello and P. Wheeler, “The expensive-tissue hypothesis: The brain and the digestive system in human and primate evolution,” *Current Anthropology*, vol. 36, no. 2, pp. 199–221, 1995.

[82] J. E. Niven and S. B. Laughlin, “Energy limitation as a selective pressure on the evolution of sensory systems,” *Journal of Experimental Biology*, vol. 211, no. 11, pp. 1792–1804, 2008.

[83] S. Li, C. Yan, and Y. Liu, “Energy efficiency and coding of neural network,” *Frontiers in Neuroscience*, vol. 16, 2023.

[84] C. Koch, *Biophysics of Computation: Information Processing in Single Neurons*, Oxford University Press, 1998.

[85] L. F. Abbott, “Lapicque’s introduction of the integrate-and-fire model neuron (1907),” *Brain Research Bulletin*, vol. 50, no. 5, pp. 303–304, 1999.

[86] A. L. Hodgkin and A. F. Huxley, “A quantitative description of membrane current and its application to conduction and excitation in nerve,” *The Journal of Physiology*, vol. 117, no. 4, pp. 500–544, 1952.

[87] T. V. P. Bliss and G. L. Collingridge, “A synaptic model of memory: long-term potentiation in the hippocampus,” *Nature*, vol. 361, no. 6407, pp. 31–39, 1993.

[88] S. F. Cooke and T. V. P. Bliss, “Plasticity in the human central nervous system,” *Brain*, vol. 129, no. 7, pp. 1659–1673, 2006.

[89] G. Hinton and T. J. Sejnowski, *Unsupervised Learning: Foundations of Neural Computation*, The MIT Press, 1999.

[90] W. Gerstner, A. K. Kreiter, H. Markram, and A. V. M. Herz, “Neural codes: Firing rates and beyond,” *Proceedings of the National Academy of Sciences*, vol. 94, no. 24, pp. 12740–12741, 1997.

[91] E. D. Adrian and Y. Zotterman, “The impulses produced by sensory nerve-endings: Part II. The response of a single end-organ,” *The Journal of Physiology*, vol. 61, no. 2, pp. 151–171, 1926.

[92] A. J. Fuglevand, D. A. Winter, and A. E. Patla, “Models of recruitment and rate coding organization in motor-unit pools,” *Journal of Neurophysiology*, vol. 70, no. 6, pp. 2470–2488, 1993.

[93] E. Salinas, A. Hernández, A. Zainos, and R. Romo, “Periodicity and firing rate as candidate neural codes for the frequency of vibrotactile stimuli,” *Journal of Neuroscience*, vol. 20, no. 14, pp. 5503–5515, 2000.

[94] M. Abeles, H. Bergman, E. Margalit, and E. Vaadia, “Spatiotemporal firing patterns in the frontal cortex of behaving monkeys,” *Journal of Neurophysiology*, vol. 70, no. 4, pp. 1629–1638, 1993.

[95] P. Reinagel and R. C. Reid, “Temporal coding of visual information in the thalamus,” *Journal of Neuroscience*, vol. 20, no. 14, pp. 5392–5400, 2000.

[96] T. Gollisch and M. Meister, “Rapid neural coding in the retina with relative spike latencies,” *Science*, vol. 319, no. 5866, pp. 1108–1111, 2008.

[97] A. Riehle, S. Grün, M. Diesmann, and A. Aertsen, “Spike synchronization and rate modulation differentially involved in motor cortical function,” *Science*, vol. 278, no. 5345, pp. 1950–1953, 1997.

[98] M. Diesmann, M.-O. Gewaltig, and A. Aertsen, “Stable propagation of synchronous spiking in cortical neural networks,” *Nature*, vol. 402, no. 6761, pp. 529–533, 1999.

[99] N. S. Harper and D. McAlpine, “Optimal neural population coding of an auditory spatial cue,” *Nature*, vol. 430, no. 7000, pp. 682–686, 2004.

[100] S. Thorpe and J. Gautrais, "Rank order coding," in J. M. Bower (Ed.), *Computational Neuroscience*, pp. 113–118, Springer US, 1998.

[101] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "STDP-based spiking deep convolutional neural networks for object recognition," *Neural Networks*, vol. 99, pp. 56–67, 2018.

[102] B. Sakmann and E. Neher, *Single-Channel Recording*, Springer New York, 1995.

[103] R. E. Kass, U. T. Eden, and E. N. Brown, *Analysis of Neural Data*, Springer New York, 2014.

[104] M. J. Chacron, B. Lindner, and A. Longtin, "Noise shaping by interval correlations increases information transfer," *Physical Review Letters*, vol. 92, pp. 080601, 2004.

[105] W. Truccolo, U. T. Eden, M. R. Fellows, et al., "A point process framework for relating neural spiking activity to spiking history, neural ensemble, and extrinsic covariate effects," *Journal of Neurophysiology*, vol. 93, no. 2, pp. 1074–1089, 2005.

[106] M. P. Nawrot, C. Boucsein, V. Rodriguez-Molina, et al., "Serial interval statistics of spontaneous activity in cortical neurons in vivo and in vitro," *Neurocomputing*, vol. 70, no. 10, pp. 1717–1722, 2007.

[107] O. Avila-Akerberg and M. J. Chacron, "Nonrenewal spike train statistics: causes and functional consequences on neural coding," *Experimental Brain Research*, vol. 210, no. 3, pp. 353–371, 2011.

[108] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers in Neuroscience*, vol. 12, 2018.

[109] J. Dethier, P. Nuyujukian, S. I. Ryu, et al., "Design and validation of a real-time spiking-neural-network decoder for brain-machine interfaces," *Journal of Neural Engineering*, vol. 10, no. 3, pp. 036008, 2013.

[110] J. K. Eshraghian, M. Ward, E. O. Neftci, et al., "Training spiking neural networks using lessons from deep learning," *Proceedings of the IEEE*, vol. 111, no. 9, pp. 1016–1054, 2023.

[111] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, 2015.

[112] B. Rueckauer, I.-A. Lungu, Y. Hu, et al., "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in Neuroscience*, vol. 11, 2017.

[113] B. Rueckauer and S.-C. Liu, “Conversion of analog to spiking neural networks using sparse temporal coding,” in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–5.

[114] R. Midya, Z. Wang, S. Asapu, et al., “Artificial neural network (ANN) to spiking neural network (SNN) converters based on diffusive memristors,” *Advanced Electronic Materials*, vol. 5, no. 9, pp. 1900060, 2019.

[115] C. Stöckl and W. Maass, “Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes,” *Nature Machine Intelligence*, vol. 3, no. 3, pp. 230–238, 2021.

[116] R. Gütig and H. Sompolinsky, “The tempotron: a neuron that learns spike timing–based decisions,” *Nature Neuroscience*, vol. 9, no. 3, pp. 420–428, 2006.

[117] J. Wu, Y. Chua, M. Zhang, et al., “A spiking neural network framework for robust sound classification,” *Frontiers in Neuroscience*, vol. 12, 2018.

[118] E. Hunsberger and C. Eliasmith, “Training spiking deep networks for neuromorphic hardware,” *arXiv preprint arXiv:1611.05141*, 2016.

[119] J. H. Lee, T. Delbruck, and M. Pfeiffer, “Training deep spiking neural networks using backpropagation,” *Frontiers in Neuroscience*, vol. 10, 2016.

[120] D. Rasmussen, “NengoDL: Combining deep learning and neuromorphic modelling methods,” *Neuroinformatics*, vol. 17, no. 4, pp. 611–628, 2019.

[121] N. Rathi, G. Srinivasan, P. Panda, and K. Roy, “Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation,” *arXiv preprint arXiv:2005.01807*, 2020.

[122] E. Hunsberger and C. Eliasmith, “Training spiking deep networks for neuromorphic hardware,” *arXiv preprint arXiv:1611.05141*, 2016.

[123] H. Mostafa, “Supervised learning based on temporal coding in spiking neural networks,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 7, pp. 3227–3235, 2018.

[124] A. Javanshir, T. T. Nguyen, M. A. P. Mahmud, and A. Z. Kouzani, “Advancements in algorithms and neuromorphic hardware for spiking neural networks,” *Neural Computation*, vol. 34, no. 6, pp. 1289–1328, 2022.

[125] F. Corradi, S. Pande, J. Stuijt, et al., “ECG-based heartbeat classification in neuromorphic hardware,” in *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019, pp. 1–8.

[126] E. Donati, M. Payvand, N. Risi, et al., "Discrimination of EMG signals using a neuromorphic implementation of a spiking neural network," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 5, pp. 795–803, 2019.

[127] P. Negri, M. Soto, B. Linares-Barranco, and T. Serrano-Gotarredona, "Scene context classification with event-driven spiking deep neural networks," in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2018, pp. 569–572.

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

[129] I. Sporea and A. Grüning, "Supervised learning in multilayer spiking neural networks," *Neural Computation*, vol. 25, no. 2, pp. 473–509, 2013.

[130] S. M. Bohte, J. N. Kok, and H. La Poutré, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1, pp. 17–37, 2002.

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

[132] Y. Xu, X. Zeng, L. Han, and J. Yang, "A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks," *Neural Networks*, vol. 43, pp. 99–113, 2013.

[133] D. Huh and T. J. Sejnowski, "Gradient descent for spiking neural networks," in *Advances in Neural Information Processing Systems*, 2018, vol. 31.

[134] W. Zhang and P. Li, "Temporal spike sequence learning via backpropagation for deep spiking neural networks," in *Advances in Neural Information Processing Systems*, 2020, vol. 33, pp. 12022–12033.

[135] N. Perez-Nieves and D. Goodman, "Sparse spiking gradient descent," in *Advances in Neural Information Processing Systems*, 2021, vol. 34, pp. 11795–11808.

[136] Y. Zhu, Z. Yu, W. Fang, et al., "Training spiking neural networks with event-driven backpropagation," in *Advances in Neural Information Processing Systems*, 2022, vol. 35, pp. 30528–30541.

[137] T. C. Wunderlich and C. Pehle, "Event-based backpropagation can compute exact gradients for spiking neural networks," *Scientific Reports*, vol. 11, no. 1, pp. 12829, 2021.

[138] S. Rotter and M. Diesmann, “Exact digital simulation of time-invariant linear systems with applications to neuronal modeling,” *Biological Cybernetics*, vol. 81, no. 5, pp. 381–402, 1999.

[139] Y. Wu, L. Deng, G. Li, et al., “Spatio-temporal backpropagation for training high-performance spiking neural networks,” *Frontiers in Neuroscience*, vol. 12, 2018.

[140] H. Zheng, Y. Wu, L. Deng, et al., “Going deeper with directly-trained larger spiking neural networks,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, pp. 11062–11070, 2021.

[141] F. Zenke and S. Ganguli, “SuperSpike: Supervised learning in multilayer spiking neural networks,” *Neural Computation*, vol. 30, no. 6, pp. 1514–1541, 2018.

[142] S. B. Shrestha and G. Orchard, “SLAYER: Spike layer error reassignment in time,” in *Advances in Neural Information Processing Systems*, 2018, vol. 31.

[143] F. C. Bauer, G. Lenz, S. Haghighatshoar, and S. Sheik, “EXODUS: Stable and efficient training of spiking neural networks,” *Frontiers in Neuroscience*, vol. 17, 2023.

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

[145] B. Yin, F. Corradi, and S. M. Bohté, “Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks,” *Nature Machine Intelligence*, vol. 3, no. 10, pp. 905–913, 2021.

[146] Y. Li, Y. Guo, S. Zhang, et al., “Differentiable spike: Rethinking gradient-descent for training spiking neural networks,” in *Advances in Neural Information Processing Systems*, 2021, vol. 34, pp. 23426–23439.

[147] W. Fang, Z. Yu, Y. Chen, et al., “Deep residual learning in spiking neural networks,” in *Advances in Neural Information Processing Systems*, 2021, vol. 34, pp. 21056–21069.

[148] Y. Kim, J. Chough, and P. Panda, “Beyond classification: directly training spiking neural networks for semantic segmentation,” *Neuromorphic Computing and Engineering*, vol. 2, no. 4, pp. 044015, 2022.

[149] I. M. Comsa, K. Potempa, L. Versari, et al., “Temporal coding in spiking neural networks with alpha synaptic function,” in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 8529–8533.

[150] S. R. Kheradpisheh and T. Masquelier, “Temporal backpropagation for spiking neural networks with one spike per neuron,” *International Journal of Neural Systems*, vol. 30, no. 6, pp. 2050027, 2020.

[151] S. Zhou, X. Li, Y. Chen, et al., "Temporal-coded deep spiking neural network with easy training and robust performance," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021, vol. 35, pp. 11143–11151.

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

[153] T. Serrano-Gotarredona and B. Linares-Barranco, "Poker-DVS and MNIST-DVS. Their history, how they were made, and other details," *Frontiers in Neuroscience*, vol. 9, 2015.

[154] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 2, pp. 107–116, 1998.

[155] E. Neftci, S. Das, B. Pedroni, et al., "Event-driven contrastive divergence for spiking neuromorphic systems," *Frontiers in Neuroscience*, vol. 7, 2014.

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

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

[158] M. Abadi, P. Barham, J. Chen, et al., "TensorFlow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283.

[159] A. Paszke, S. Gross, F. Massa, et al., "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019, vol. 32.

[160] C. R. Harris, K. J. Millman, S. J. van der Walt, et al., "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[161] T. Tieleman, G. Hinton, et al., "Lecture 6.5 – RMSprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[162] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, 2015, vol. 37, pp. 448–456.

[163] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[164] F. Akopyan, J. Sawada, A. Cassidy, et al., “TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[165] M. Davies, N. Srinivasa, T.-H. Lin, et al., “Loihi: A neuromorphic manycore processor with on-chip learning,” *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[166] N. M. Timme and C. Lapish, “A tutorial for information theory in neuroscience,” *eNeuro*, vol. 5, no. 3, 2018.

[167] M. C. W. van Rossum, “A novel spike distance,” *Neural Computation*, vol. 13, no. 4, pp. 751–763, 2001.

[168] S. Schreiber, J. M. Fellous, D. Whitmer, et al., “A new correlation-based measure of spike timing reliability,” *Neurocomputing*, vol. 52-54, pp. 925–931, 2003.

[169] A. Szűcs, “Applications of the spike density function in analysis of neuronal firing patterns,” *Journal of Neuroscience Methods*, vol. 81, no. 1, pp. 159–167, 1998.

[170] A. R. Paiva, I. Park, and J. C. Príncipe, “A reproducing kernel Hilbert space framework for spike train signal processing,” *Neural Computation*, vol. 21, no. 2, pp. 424–449, 2009.

[171] E. Satuvuori and T. Kreuz, “Which spike train distance is most suitable for distinguishing rate and temporal coding?,” *Journal of Neuroscience Methods*, vol. 299, pp. 22–33, 2018.

[172] A. R. C. Paiva, I. Park, and J. C. Príncipe, “A comparison of binless spike train measures,” *Neural Computing and Applications*, vol. 19, no. 3, pp. 405–419, 2010.

[173] D. Chicharro, T. Kreuz, and R. G. Andrzejak, “What can spike train distances tell us about the neural code?,” *Journal of Neuroscience Methods*, vol. 199, no. 1, pp. 146–165, 2011.

[174] T. Kreuz, J. S. Haas, A. Morelli, et al., “Measuring spike train synchrony,” *Journal of Neuroscience Methods*, vol. 165, no. 1, pp. 151–161, 2007.

[175] T. Kreuz, D. Chicharro, M. Greschner, and R. G. Andrzejak, “Time-resolved and time-scale adaptive measures of spike train synchrony,” *Journal of Neuroscience Methods*, vol. 195, no. 1, pp. 92–106, 2011.

[176] E. Satuvuori, M. Mulansky, N. Bozanic, et al., “Measures of spike train synchrony for data with multiple time scales,” *Journal of Neuroscience Methods*, vol. 287, pp. 25–38, 2017.

[177] D. Sihn and S.-P. Kim, “A spike train distance robust to firing rate changes based on the Earth Mover’s Distance,” *Frontiers in Computational Neuroscience*, vol. 13, 2019.

[178] J. Bromley, I. Guyon, Y. LeCun, et al., “Signature verification using a "Siamese" time delay neural network,” in *Advances in Neural Information Processing Systems*, 1993, vol. 6.

[179] J. Mueller and A. Thyagarajan, “Siamese recurrent architectures for learning sentence similarity,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016, vol. 30.

[180] Y. Dong, X. Yang, H. Wu, et al., “Lightweight and edge-preserving speckle matching network for precise single-shot 3D shape measurement,” *Measurement*, vol. 210, pp. 112549, 2023.

[181] T. Jeyapoovan and M. Murugan, “Surface roughness classification using image processing,” *Measurement*, vol. 46, no. 7, pp. 2065–2072, 2013.

[182] F. Schroff, D. Kalenichenko, and J. Philbin, “FaceNet: A unified embedding for face recognition and clustering,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 815–823.

[183] G. Koch, R. Zemel, and R. Salakhutdinov, “Siamese neural networks for one-shot image recognition,” in *Proceedings of the 32nd International Conference on International Conference on Machine Learning (ICML)*, 2015, vol. 37.

[184] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, “On the surprising behavior of distance metrics in high dimensional space,” in *Database Theory — ICDT 2001*, 2001, pp. 420–434.

[185] R. Hadsell, S. Chopra, and Y. LeCun, “Dimensionality reduction by learning an invariant mapping,” in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)*, 2006, vol. 2, pp. 1735–1742.

[186] S. Chopra, R. Hadsell, and Y. LeCun, “Learning a similarity metric discriminatively, with application to face verification,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, 2005, vol. 1, pp. 539–546.

[187] B. Song, “Deep neural network for learning to rank query-text pairs,” *arXiv preprint arXiv:1802.08988*, 2018.

[188] C.-Y. Wu, R. Manmatha, A. J. Smola, and P. Krahenbuhl, “Sampling matters in deep embedding learning,” in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2840–2848.

[189] A. Hermans, L. Beyer, and B. Leibe, “In defense of the triplet loss for person re-identification,” *arXiv preprint arXiv:1703.07737*, 2017.

[190] L. Xie, Z. Wu, X. Zhang, et al., “Writer-independent online signature verification based on 2D representation of time series data using triplet supervised network,” *Measurement*, vol. 197, pp. 111312, 2022.

[191] M. Dunnhofer, M. Antico, F. Sasazawa, et al., “Siam-U-Net: encoder-decoder siamese network for knee cartilage tracking in ultrasound images,” *Medical Image Analysis*, vol. 60, pp. 101631, 2020.

[192] Y. Xu, J. Zhang, and J. Brownjohn, “An accurate and distraction-free vision-based structural displacement measurement method integrating Siamese network based tracker and correlation-based template matching,” *Measurement*, vol. 179, pp. 109506, 2021.

[193] H. Bredin, “TristouNet: Triplet loss for speaker turn embedding,” in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 5430–5434.

[194] R. C. Daudt, B. L. Saux, and A. Boulch, “Fully convolutional Siamese networks for change detection,” in *2018 25th IEEE International Conference on Image Processing (ICIP)*, 2018, pp. 4063–4067.

[195] X. Liu, J. van de Weijer, and A. D. Bagdanov, “RankIQA: Learning from rankings for no-reference image quality assessment,” in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 1040–1049.

[196] H. Doughty, D. Damen, and W. Mayol-Cuevas, “Who’s better? Who’s best? Pairwise deep ranking for skill determination,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 6057–6066.

[197] X. Lin, X. Wang, and Z. Hao, “Supervised learning in multilayer spiking neural networks with inner products of spike trains,” *Neurocomputing*, vol. 237, pp. 59–70, 2017.

[198] Y. Xing, G. Di Caterina, and J. Soraghan, “A new spiking convolutional recurrent neural network (SCRNN) with applications to event-based hand gesture recognition,” *Frontiers in Neuroscience*, vol. 14, 2020.

[199] Y. Luo, M. Xu, C. Yuan, et al., “SiamSNN: Siamese spiking neural networks for energy-efficient object tracking,” in *Artificial Neural Networks and Machine Learning – ICANN 2021*, 2021, pp. 182–194.

[200] T. Kreuz, D. Chicharro, C. Houghton, et al., “Monitoring spike train synchrony,” *Journal of Neurophysiology*, vol. 109, no. 5, pp. 1457–1472, 2013.

[201] T. Kreuz, M. Mulansky, and N. Bozanic, “SPIKY: a graphical user interface for monitoring spike train synchrony,” *Journal of Neurophysiology*, vol. 113, no. 9, pp. 3432–3445, 2015.

[202] G. Peyré and M. Cuturi, "Computational optimal transport: With applications to data science," *Foundations and Trends in Machine Learning*, vol. 11, no. 5-6, pp. 355–607, 2019.

[203] S. Cohen, *Finding Color And Shape Patterns In Images*, Ph.D. thesis, Stanford University, 1999.

[204] D. Rzepka, M. Miśkowicz, D. Kościelnik, and N. T. Thao, "Reconstruction of signals from level-crossing samples using implicit information," *IEEE Access*, vol. 6, pp. 35001–35011, 2018.

[205] Ł. Bibrzycki, D. Burakowski, P. Homola, et al., "Towards a global cosmic ray sensor network: CREDO detector as the first open-source mobile application enabling detection of penetrating radiation," *Symmetry*, vol. 12, no. 11, 2020.

[206] M. Piekarczyk, O. Bar, Ł. Bibrzycki, et al., "CNN-based classifier as an offline trigger for the CREDO experiment," *Sensors*, vol. 21, no. 14, 2021.

[207] O. Bar, Ł. Bibrzycki, M. Niedźwiecki, et al., "Zernike moment based classification of cosmic ray candidate hits from CMOS sensors," *Sensors*, vol. 21, no. 22, 2021.

[208] F. Zareef, A. Oblakowska-Mucha, and T. Szumlak, "Silicon detectors beyond LHC – RD50 status report," *Journal of Instrumentation*, vol. 17, no. 11, pp. C11004, 2022.

[209] D. L. Donoho, "Compressed sensing," *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[210] F. Marvasti, *Nonuniform Sampling: Theory and Practice*, Springer New York, 2001.

[211] M. Miśkowicz, "Reducing communication by event-triggered sampling," in M. Miśkowicz (Ed.), *Event-Based Control and Signal Processing*, pp. 37–58, CRC Press, 2015.

[212] M. Greitans and R. Shavelis, "Speech sampling by level-crossing and its reconstruction using spline-based filtering," in *2007 14th International Workshop on Systems, Signals and Image Processing and 6th EURASIP Conference focused on Speech and Image Processing, Multimedia Communications and Services*, 2007, pp. 292–295.

[213] C. Vezyrtzis and Y. Tsividis, "Processing of signals using level-crossing sampling," in *2009 IEEE International Symposium on Circuits and Systems*, 2009, pp. 2293–2296.

[214] D. Rzepka, M. Pawlak, D. Kościelnik, and M. Miśkowicz, "Reconstruction of varying bandwidth signals from event-triggered samples," in M. Miśkowicz (Ed.), *Event-Based Control and Signal Processing*, pp. 529–546, CRC Press, 2015.

[215] P. Martínez-Nuevo, S. Patil, and Y. Tsividis, "Derivative level-crossing sampling," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 1, pp. 11–15, 2015.

[216] D. Rzepka and M. Miśkowicz, "Recovery of varying-bandwidth signal from samples of its extrema," in *2013 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, 2013, pp. 143–148.

[217] J. Selva, "Efficient sampling of band-limited signals from sine wave crossings," *IEEE Transactions on Signal Processing*, vol. 60, no. 1, pp. 503–508, 2012.

[218] M. Miśkowicz, "Send-on-delta concept: An event-based data reporting strategy," *Sensors*, vol. 6, no. 1, pp. 49–63, 2006.

[219] Y. S. Suh, "Send-on-delta sensor data transmission with a linear predictor," *Sensors*, vol. 7, no. 4, pp. 537–547, 2007.

[220] M. Miśkowicz, "The event-triggered integral criterion for sensor sampling," in *Proceedings of the IEEE International Symposium on Industrial Electronics, 2005. ISIE 2005.*, 2005, vol. 3, pp. 1061–1066.

[221] M. Miśkowicz, "Efficiency of event-based sampling according to error energy criterion," *Sensors*, vol. 10, no. 3, pp. 2242–2261, 2010.

[222] A. A. Lazar and L. T. Tóth, "Time encoding and perfect recovery of bandlimited signals," in *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03).*, 2003, vol. 6, pp. VI–709.

[223] A. A. Lazar, "Time encoding with an integrate-and-fire neuron with a refractory period," *Neurocomputing*, vol. 58-60, pp. 53–58, 2004.

[224] A. A. Lazar, "Multichannel time encoding with integrate-and-fire neurons," *Neurocomputing*, vol. 65-66, pp. 401–407, 2005.

[225] A. A. Lazar, "Population encoding with Hodgkin–Huxley neurons," *IEEE Transactions on Information Theory*, vol. 56, no. 2, pp. 821–837, 2010.

[226] T. Strohmer, "Numerical analysis of the non-uniform sampling problem," *Journal of Computational and Applied Mathematics*, vol. 122, no. 1, pp. 297–316, 2000.

[227] H. Choi and D. C. Munson, "Stochastic formulation of bandlimited signal interpolation," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, no. 1, pp. 82–85, 2000.

[228] A. Aldroubi and K. Gröchenig, "Nonuniform sampling and reconstruction in shift-invariant spaces," *SIAM Review*, vol. 43, no. 4, pp. 585–620, 2001.

[229] N. T. Thao, D. Rzepka, and M. Miśkowicz, "Bandlimited signal reconstruction from leaky integrate-and-fire encoding using POCS," *IEEE Transactions on Signal Processing*, vol. 71, pp. 1464–1479, 2023.

[230] L. A. Klein, M. K. Mills, D. R. P. Gibson, et al., "Traffic Detector Handbook: Volume I," Tech. Rep., Federal Highway Administration (Turner-Fairbank Highway Research Center), 2006.

[231] J. Gajda, R. Sroka, M. Stencel, et al., "A vehicle classification based on inductive loop detectors," in *IMTC 2001. Proceedings of the 18th IEEE Instrumentation and Measurement Technology Conference. Rediscovering Measurement in the Age of Informatics (Cat. No.01CH 37188)*, 2001, vol. 1, pp. 460–464.

[232] S. Oh, S. G. Ritchie, and C. Oh, "Real-time traffic measurement from single loop inductive signatures," *Transportation Research Record*, vol. 1804, no. 1, pp. 98–106, 2002.

[233] Y.-K. Ki and D.-K. Baik, "Vehicle-classification algorithm for single-loop detectors using neural networks," *IEEE Transactions on Vehicular Technology*, vol. 55, no. 6, pp. 1704–1711, 2006.

[234] S.-T. Jeng and S. G. Ritchie, "Real-time vehicle classification using inductive loop signature data," *Transportation Research Record*, vol. 2086, no. 1, pp. 8–22, 2008.

[235] S. Meta and M. G. Cinsdikici, "Vehicle-classification algorithm based on component analysis for single-loop inductive detector," *IEEE Transactions on Vehicular Technology*, vol. 59, no. 6, pp. 2795–2805, 2010.

[236] T. M. Kwon, "Route tracking of border crossing vehicles using inductance signatures of loop detectors," in *Proceedings of the 2005 IEEE International Workshop on Measurement Systems for Homeland Security, Contraband Detection and Personal Safety Workshop, 2005. (IMS 2005)*, 2005, pp. 103–109.

[237] M. Ndoye, V. F. Totten, J. V. Krogmeier, and D. M. Bullock, "Sensing and signal processing for vehicle reidentification and travel time estimation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 1, pp. 119–131, 2011.

[238] D. Guilbert, C. Le Bastard, S.-S. Ieng, and Y. Wang, "Re-identification by inductive loop detector: Experimentation on target origin – destination matrix," in *2013 IEEE Intelligent Vehicles Symposium (IV)*, 2013, pp. 1421–1427.

[239] D. Guilbert, S.-S. Ieng, C. L. Bastard, and Y. Wang, "Robust blind deconvolution process for vehicle reidentification by an inductive loop detector," *IEEE Sensors Journal*, vol. 14, no. 12, pp. 4315–4322, 2014.

[240] C. Sun and S. G. Ritchie, "Individual vehicle speed estimation using single loop inductive waveforms," *Journal of Transportation Engineering*, vol. 125, no. 6, pp. 531–538, 1999.

[241] B. Coifman, S. Dhoorjaty, and Z.-H. Lee, "Estimating median velocity instead of mean velocity at single loop detectors," *Transportation Research Part C: Emerging Technologies*, vol. 11, no. 3, pp. 211–222, 2003.

[242] X.-Y. Lu, P. Varaiya, R. Horowitz, et al., "Estimating traffic speed with single inductive loop event data," *Transportation Research Record*, vol. 2308, no. 1, pp. 157–166, 2012.

[243] J. Gajda, P. Piwowar, R. Sroka, et al., "Application of inductive loops as wheel detectors," *Transportation Research Part C: Emerging Technologies*, vol. 21, no. 1, pp. 57–66, 2012.

[244] Z. Marszałek, R. Sroka, and T. Zeglen, "Inductive loop for vehicle axle detection from first concepts to the system based on changes in the sensor impedance components," in *2015 20th International Conference on Methods and Models in Automation and Robotics (MMAR)*, 2015, pp. 765–769.

[245] Z. Marszałek, T. Zeglen, R. Sroka, and J. Gajda, "Inductive loop axle detector based on resistance and reactance vehicle magnetic profiles," *Sensors*, vol. 18, no. 7, 2018.

[246] A. Mocholí-Salcedo, J. H. Arroyo-Núñez, V. M. Milián-Sánchez, et al., "Magnetic field generated by the loops used in traffic control systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 8, pp. 2126–2136, 2017.

[247] Z. Marszałek, "Maxwell-Wien bridge with vector voltmeter system for measurement small and rapid changes in inductive-loop sensor impedance components," *Measurement*, vol. 121, pp. 57–61, 2018.

[248] Z. Marszałek and K. Duda, "Multifrequency vector measurement system for reliable vehicle magnetic profile assessment," *Sensors*, vol. 20, no. 17, 2020.

[249] F. Mocholí Belenguer, A. Mocholí Salcedo, A. Guill Ibañez, and V. Milián Sánchez, "Advantages offered by the double magnetic loops versus the conventional single ones," *PLOS ONE*, vol. 14, no. 2, 2019.

[250] Z. Marszałek, K. Duda, P. Piwowar, et al., "Load estimation of moving passenger cars using inductive-loop technology," *Sensors*, vol. 23, no. 4, 2023.

[251] M. Cuturi, "Sinkhorn distances: Lightspeed computation of optimal transport," in *Advances in Neural Information Processing Systems*, 2013, vol. 26.

[252] J. Feydy, T. Séjourné, F.-X. Vialard, et al., "Interpolating between optimal transport and MMD using Sinkhorn divergences," in *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, 2019, vol. 89, pp. 2681–2690.

[253] C. Houghton and K. Sen, "A new multineuron spike train metric," *Neural Computation*, vol. 20, no. 6, pp. 1495–1511, 2008.

[254] C. Houghton and T. Kreuz, "On the efficient calculation of van Rossum distances," *Network: Computation in Neural Systems*, vol. 23, no. 1–2, pp. 48–58, 2012.

[255] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the 30th International Conference on Machine Learning*, 2013, vol. 28, pp. 115–123.

[256] H. Hofmann, H. Wickham, and K. Kafadar, "Letter-value plots: Boxplots for large data," *Journal of Computational and Graphical Statistics*, vol. 26, no. 3, pp. 469–477, 2017.