TUM School of Computation, Information and Technology
Technische Universität München

Low-power Time Series Processing with Spiking Neural Networks

Daniel Gustav Auge

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz:
Prof. Dr. Claudia Eckert

Prüfer*innen der Dissertation:

1. Prof. Dr.-Ing. habil Alois Chr. Knoll
2. Prof. Dr.-Ing. habil Erwin Biebl

Die Dissertation wurde am 15.07.2022 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 08.02.2023 angenommen.

Abstract

Neural network-based systems are deployed in an ever-growing number of applications. While in some cases, Artificial Neural Networks (ANNs) replace classical approaches for the benefits of higher accuracy, faster inference, or better generalizability, in others, they enable the development of totally new markets. The increasing number of deployed ANNs in server applications, cars, mobile phones, and even vacuum cleaners drives the motivation to increase the networks' energy efficiency. Especially in embedded systems, efficiency is crucial due to their restricted power budgets.

As a promising approach to increase the overall efficiency of neural network-based systems, Spiking Neural Networks (SNNs) are in the focus of many current studies. SNNs have demonstrated promising properties concerning energy efficiency and their ability to solve complex problems. Inspired by biological nervous systems, they are not only used to understand the complex relations of brain activity. Recent research also investigates their usage in commercial applications as alternatives to ANNs. In direct comparison with common networks, however, SNNs often do not reach comparable accuracy when solving the same problem. It is thus not clear whether networks based on these biologically inspired neurons are viable and efficient alternatives to current solutions.

This thesis contributes to this ongoing discussion. We analyze encoding schemes needed to translate real-world data into event representations. Thereby, we propose sparse encoding schemes, which encode data sequences into spatio-temporal event representations. Additionally, we extend existing approaches used in the training of ANNs to their application in SNNs and analyze the scaling behavior of the examined networks with a focus on the computational costs.

The proposed methods are applied in two applications: speech recognition and gesture recognition. The results show that the SNNs reach performances, which are comparable with those of ANNs while needing fewer and less expensive computational operations. Hence, our findings support the evidence that SNNs are suitable alternatives to ANNs in the field of low-power time series classification.

Zusammenfassung

Auf neuronalen Netzen basierende Systeme werden in einer ständig wachsenden Zahl von Anwendungen eingesetzt. Während in einigen Fällen künstliche neuronale Netze (ANNs) klassische Ansätze nur ersetzen, ermöglichen sie in anderen Fällen die Entwicklung völlig neuer Märkte. Die zunehmende Zahl von ANNs in Serveranwendungen, Fahrzeugen, Mobiltelefonen und sogar Staubsaugern treibt die Motivation voran, die Energieeffizienz der Netze zu erhöhen. Besonders in eingebetteten Systemen ist die Effizienz aufgrund des begrenzten Energiebudgets entscheidend.

Als ein vielversprechender Ansatz zur Steigerung der Gesamteffizienz von Systemen, die auf neuronalen Netzen basieren, stehen gepulste neuronale Netze (SNNs) im Fokus vieler aktueller Studien. Inspiriert von biologischen Nervensystemen werden sie nicht nur genutzt, um die komplexen Zusammenhänge der Gehirnaktivität zu verstehen. Jüngste Forschungsarbeiten untersuchen auch ihren Einsatz in kommerziellen Anwendungen als Alternative zu ANNs. SNNs zeigen vielversprechende Eigenschaften in Bezug auf Energieeffizienz und ihre Fähigkeit, komplexe Probleme zu lösen. Im direkten Vergleich mit herkömmlichen Netzen erreichen SNNs jedoch oft nicht die gleiche Genauigkeit bei der Lösung desselben Problems. Es ist daher nicht klar, ob Netze, die auf diesen biologisch inspirierten Neuronen basieren, universell nutzbare und effiziente Alternativen zu aktuellen Lösungen sind.

Diese Arbeit leistet einen Beitrag zu dieser laufenden Diskussion. Wir analysieren die Kodierungsschemata, die notwendig sind, um Daten aus der realen Welt in ereignisgetriebene Repräsentationen zu übersetzen. Dabei schlagen wir spärliche Kodierungsschemata vor, die Datensequenzen in räumlich-zeitliche Repräsentationen kodieren. Zusätzlich erweitern wir bestehende Ansätze aus dem Training von ANNs auf deren Anwendung in SNNs und analysieren das Skalierungsverhalten der untersuchten Netze mit Fokus auf den Rechenaufwand.

Die vorgeschlagenen Methoden werden in zwei Anwendungen, Spracherkennung und Gestenerkennung, angewendet. Die resultierenden Ergebnisse zeigen, dass die SNNs Vorhersagegenauigkeiten erreichen, die mit denen von ANNs vergleichbar sind, während sie zahlenmäßig weniger sowie weniger teure Rechenoperationen benötigen. Daher unterstützen unsere Ergebnisse die Hypothese, dass SNNs eine geeignete Alternative zu ANNs im Bereich der Klassifizierung von Zeitreihen mit geringem Stromverbrauch sind.

Acknowledgments

Foremost, I would like to thank Prof. Alois Knoll, who made this research possible in the first place. He initiated this close collaboration between industry and academia, for which I am genuinely grateful. Additionally, he provided discussions and guidance throughout the whole project. Without that, a successful conclusion of the research project would not have been possible.

The research project was funded by Infineon Technologies AG. I want to express my special thanks for this opportunity. Representatively, I would like to thank Dr. Cyprian Grassmann; also for being my mentor and helping me during every stage of the project.

Next, I would like to thank the academic and administrative staff at the Chair of Robotics, Artificial Intelligence, and Real-time Systems: Dr. Alexander Lenz, Ute Lomp, and Amy Bücherl. They provided a positive and supportive working environment, no matter the circumstances or the kind of questions I had.

In these challenging times of a global pandemic, lock-downs, and “home office”, social interaction is just as important as scientific collaboration. Therefore, I would like to thank my colleagues at the university and at Infineon, who created an enjoyable environment that promoted new ideas and the motivation to pursue further research directions. Specifically, I would like to thank Etienne Müller, Julian Hille, and Robin Dietrich for the many discussions and collaborations, both in person and in pandemic-conform phone calls.

Lastly, this work would not have been possible without the support of my partner Sarah and my family. Thank you for being there for me.

Contents

List of Figures

List of Tables

List of Acronyms

ADC Analog-to-Digital Converter

AEIF Adaptive Exponential Integrate-and-Fire

ALIF Adaptive Leaky Integrate-and-Fire

ANN Artificial Neural Network

ASIC Application-Specific Integrated Circuit

BMBF Bundesministerium für Bildung und Forschung

BPTT Backpropagation Through Time

BSA Ben’s Spiker Algorithm

CNN Convolutional Neural Network

DAC Digital-to-Analog Converter

DCT Discrete Cosine Transform

DFT Discrete Fourier Transform

DNN Deep Neural Network

DVS Dynamic Vision Sensor

EEG Electroencephalography

ELU Exponential Linear Unit

eSNN Evolving Spiking Neural Network

FC Fully Connected

FFT Fast Fourier Transform

FinFET Fin Field-Effect Transistor

FMCW Frequency Modulated Continuous Wave

FPGA Field-Programmable Gate Array

GRU Gated Recurrent Unit

HICANN High Input Count Analog Neural Network

HICANN-DLS High Input Count Analog Neural Network with Digital Learning System

HSA Hough Spiker Algorithm

IF Intermediate Frequency

IoT Internet of Things

ISI Interspike Interval

LIF Leaky Integrate-and-Fire

LOSO Leave-One-Subject-Out

LSM Liquid State Machine

LSTM Long Short-Term Memory

MAC Multiply-Accumulate

MFCC Mel-Frequency Cepstral Coefficients

MIM Metal-Insulator-Metal

NEF Neural Engineering Framework

PSTH Peri-Stimulus-Time Histogram

RBM Restricted Boltzmann Machine

RCNN Recurrent Convolutional Neural Network

RDM Range-Doppler Matrix

ReLU Rectified Linear Unit

RF Resonate-and-Fire

RNN Recurrent Neural Network

ROC Rank-Order Coding

RQ Research Question

SDR Sparse Distributed Representation

SDSP Spike-Driven Synaptic Plasticity

SNN Spiking Neural Network

SOP Synaptic Operation

STDP Spike-Timing-Dependent Plasticity

SVM Support Vector Machine

TC Temporal Contrast

TSNE T-Distributed Stochastic Neighbor Embedding

TTFS Time-to-First-Spike

VLSI Very-Large-Scale Integration

WTA Winner-Takes-All

List of Symbols

Notation

a	A scalar (real or integer)
$\underline{a}$	A scalar (complex)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
$a(t)$	A temporally changing variable (continuous time)
$a[t]$	A temporally changing variable (discrete time)
a_i	The i-th element of vector $\boldsymbol{a}$, with the index starting at 1
$a_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$a^{(i)}$	Element within the i-th layer of a network
$a^{(i,j)}$	Element with contributions from layer i to layer j
a_{descr}	A variable with distinct description
$f(a)$	A function of a
$\mathrm{Re}(\underline{a})$	Real part of complex variable $\underline{a}$
$\mathrm{Im}(\underline{a})$	Imaginary part of complex variable $\underline{a}$

Symbols

t	Time
Δt	Time interval
t^f	Firing times of a neuron
$v(t)$, $\underline{v}(t)$, $v[t]$	Membrane voltage of a neuron
v_{th}	Threshold voltage of a neuron
v_{reset}	Reset voltage of a neuron
C	Membrane capacitance of a neuron
R	Resistance of a neuron
$\delta(t)$, $z[t]$	Spike event
$i(t)$, $i[t]$	Input current of a neuron
τ	Time constant of a neuron
λ	Damping constant of a neuron
$\boldsymbol{W}$	Weight matrix containing the synaptic connection weights
f	Frequency
ω	Angular frequency
η	Learning rate
$\mathcal{L}$	Loss function
ψ_{lin}, ψ_{sig}	Pseudo gradient
s_{i}, s_{f}	Sparsity level
$\boldsymbol{x}$	Input vector
$\boldsymbol{y}$	Output vector
$\hat{\boldsymbol{y}}$	Label vector
$c_{\omega=\omega_n}(t)$	Envelope function of a neuron's response with $\omega = \omega_n$
$\sigma(x)$	Softmax function
$S(x)$	Logistic sigmoid function
N_{xxx}	Number of xxx
E_{xxx}	Energy consumption of xxx

1 Introduction

Artificial Neural Networks (ANNs) define the state-of-the-art solution in many current applications. Their superiority over classical approaches manifests in applications like pattern recognition and classification, function approximation, sequence prediction, content generation, or showing their abilities by competing in games like Chess, Go, or arcade video games [1], [2]. Their use has become indispensable in modern systems, ranging from huge compute clusters to embedded microsystems.

Despite their computational capabilities, ANNs lay way behind their biological counterparts when comparing their energy efficiency. For example, the human brain can perform the complex computations of sensory perception, cognition, and motion coordination while consuming 20 Watts. The actual consumption of the computations themselves is even lower [3], [4]. ANNs require power levels, which are orders of magnitude higher when performing complex tasks [5]. A popular example to illustrate this disparity is the historic game of Go between the human world champion Lee Sedol and his artificial opponent AlphaGo in 2016 [6]: The machine housing the artificial agent needed 1 MW to perform the necessary computations [7], roughly 10^5 times more than the human brain.

The development towards increasingly large networks facilitates solving more and more complex problems, often surpassing human capabilities. However, these large networks also necessitate large compute clusters and huge power budgets. The mismatch between the vastly growing requirements on hardware and power on the one hand, and the infinitesimal performance gains of the networks on the other hand, is addressed by the field of GreenAI [5]. There, the network's performance and requirements are always considered simultaneously.

The performance-requirements-tradeoff is particularly striking in the field of embedded systems. There, the requirements in terms of the realizable complexity of ANN-based systems in embedded applications and the available power budget are especially restricted. Small microcontroller systems offer memory in the order of a few KB and execution speeds of several MHz while consuming power in the order of mW [8]. The well known field of Internet of Things (IoT) or the advanced processing in modern driver-assisted vehicles are prominent examples for the application of these microcontroller systems.

To enable more powerful ANNs on restricted hardware, there exist several research directions to optimize distinct aspects of the system: sparse networks [9]–[11], computational optimizations [12], new network architectures [13], [14], or energy-efficient hardware accelerators [15]. A further promising approach is inspired by biological neural networks, as it mimics the functions of biological neurons more closely [16]. These networks form the next step towards their biological archetypes, which have been optimized over millions of years. As their biological counterpart, the networks have latent neuronal variables and communicate via binary all-or-nothing events, often referred to as spikes. Inspired

Table 1.1: Advantages of Spiking Neural Networks.

Claim	Reasoning	Ref.
Energy-efficient	Neuromorphic realizations of SNNs consume less energy than equivalent ANNs	[18]–[23]
Less neurons	SNNs need less neurons than ANNs to solve the same problem	[16]
Fast	SNNs operate event-based, asynchronously and highly parallelized	[24]–[26]
Time-affine	SNNs operate in the time domain; they are made for temporal pattern recognition	[27]
Adaptive	SNNs can adapt to changing input characteristics (context drift, online learning)	[28]
Robust	Communication using binary spikes is less prone to noise	[29]
Hardware-friendly	Simple neurons, local learning (in space and time)	[30], [31]

by their spike-based communication scheme, these types of networks are called Spiking Neural Networks (SNNs).

The spiking network type can be seen as the next generation of artificial network designs. Neural networks are commonly assigned to three generations [16] with increasing complexity per level. Networks based on binary threshold activation functions are defined to be part of the first generation. Perceptrons, Hopfield nets, and Boltzmann machines fall into this group of networks. The second generation distinguishes itself from the previous generation through continuous output activation functions. Examples for this generation are feedforward and recurrent networks using tanh, sigmoid, or rectified linear activation functions. SNNs form the third generation of artificial neural networks by using neurons, which encode information into temporal sequences of binary events [16]. In the following, we use the term ANN for neural networks of the first two generations, whereas SNN is used to refer to networks of the third, spiking, generation.

The underlying principles of both ANNs and SNNs have been developed several decades ago: the first binary neuron model by McChulloch and Pitts (1943), Hebbian learning (1949), multilayer perceptrons (1960s), backpropagation (1970s), SNNs (1990s) [17]. The current success of neural networks can be primarily attributed to the continuous further development of sophisticated algorithms and the rapidly increasing availability of computing power. The increased interest in SNNs for productive applications began with the development of specialized neuromorphic circuits in the 2010s.

With the biologically inspired way of conveying and processing information, SNNs have shown to have a range of advantages over ANNs of the previous generations. The most important claims are summarized in table 1.1. Many of the advantages are based on the neurons' spatially and temporally sparse communication scheme. On average, large parts of the networks are inactive and are only activated whenever new information in the form of spikes is available. Because of that, many neurons only need to be updated rarely. Based on this principle, it has been shown that realizations of SNNs consume much less energy than equivalent ANNs, resulting in a higher energy efficiency of SNNs [18]–[23]. A further advantage that results from the event-based processing scheme is the fast response to incoming stimuli. With no need for synchronization, important information can be propagated through the network efficiently, while fine-grained or less important information can enhance the resulting response during a longer interval [24]–[26]. The precise timing between spikes can also be leveraged to adapt the network continuously. Different local learning rules can, thus, be used to adapt the network to changing input characteristics with no need to propagate the error through the entire network [28]. With the distributed processing of events, it is also possible to circumvent the von Neumann bottleneck as no centralized coordination of memory and compute resources have to be performed [26], [32]. SNNs can be implemented as simple integrator units that directly communicate with each other in small local groups [30]. Larger groups of neurons can be clustered and spikes can be routed to other clusters using digital data buses [32].

However, the advantages are offset by several challenges when trying to solve the same problem with SNNs instead of ANNs [26]. In most cases, SNNs simply do not reach the same performance levels as ANNs [20], [33]–[35]. The performance gap can partially be explained by the nature of the benchmarks themselves because most benchmarks that assess the performance of neural networks comprise single frame-based image classification tasks without supporting (temporal) context [26]. While it has been shown that our brains can solve those tasks, too [36], they are much more optimized for analyzing continuous data streams that contain binary patterns of neural activations. This directly leads to the more important reason for the performance gap: Information is represented and processed differently in spike-based networks.

The challenges manifest themselves in three areas: the encoding and decoding of information, the variety of neuron models that can be deployed to construct the networks, and finally, the training of the networks to reach the desired functionality. All three areas are large fields of research in their own. Through extensive theoretical and practical works in the field of neurobiology, many details about the individual processes in our nervous system are known. However, only a minor part could be transferred to real applications so far.

It is not fully understood how sensor data can be efficiently represented by spatio-temporal spike trains. Of course, they can be encoded into temporally averaged spike rates, but this necessitates many spikes to represent only a small dynamic range, questioning the overall efficiency of the spike-based system. There are works that instead show the biological evidence and describe the importance of sparse encoding schemes in which the exact timing of each spike is crucial for the information to be transmitted [37]–[43]. Others even state, that longer periods of silence between spikes hold more information than the spikes themselves during more active intervals [44]. Implementations of these sparse encoding mechanisms verify their efficiency, however, their application is often limited to proofs of concept or toy examples. Further, because of the difference in their communication schemes, many established methods from the areas of machine learning and classical algorithms cannot be utilized directly to the use with SNNs. Backpropagation, for example, cannot be used trivially to train SNNs because spikes are non-differentiable functions, thus rendering the gradient-based error propagation impossible to use. Biologically inspired learning methods, on the other hand, often result in unstable learning performance with much need for manual tweaking [45]. Additionally, schemes that favor the time-based asynchronous communication schemes of SNNs are yet to be developed.

Thus, the nature of SNNs requires the development of new architectures, different mechanisms to represent information, and the adaption of established methods or the development of new learning algorithms. Inspired by both biological and artificial neural network realizations, plenty of solutions need to be developed to close the gap between the artificial and spiking network approaches.

Recent advances in the adoption of backpropagation-based optimization techniques for the training of SNNs [46]–[49] facilitated the utilization of SNNs in a broad range of real-world applications. With this thesis, we build upon these advances and try to get a better understanding of the tradeoffs that are involved in the use of ANNs and SNNs in embedded applications.

1.1 Research Questions and Scope of the Thesis

The overall question which motivates this work is:

> *Are neural networks based on spiking neurons viable and advantageous alternatives to common ANNs in real-world applications?*

This simple question, however, cannot be answered simply. There are works, which show, that SNNs can outperform ANNs in different individual aspects (see table 1.1). A consensus about the SNN's supremacy, however, has not been reached yet. To contribute to the discussion, this thesis considers SNNs in the context of real-world applications in small, embedded environments. Therefor, the performances of the SNNs are evaluated, and the requirements on their neuromorphic realizations are examined. To narrow the field of applications, we consider the classification of concluded data sequences. We thereby leverage the potential advantages of the time affinity of SNNs. At the same time, we have the possibility to compare our approaches with ANNs on well-established benchmarks that are not focused on frame-based image recognition.

SNNs process information using all-or-nothing pulses. Real-world data that is recorded by digital sensors, however, is not naturally present in this binary event-based form. The sensory data streams, therefore, need to be converted to a format, which can be processed by SNNs. Accordingly, our first Research Question (RQ) is:

Research Question 1
How to encode data streams into spike event representations for the processing in SNNs?

With the input being present, the network itself has to be built and trained. Biologically inspired methods as well as methods for the training of ANNs can be used to achieve this. Some of these methods can be adapted for the special requirements of SNNs. The summarizing RQ arises:

Research Question 2
How to train SNN architectures for supervised classification tasks?

One of the main reasons for the consideration of SNNs over classical ANNs is the highly efficient execution of SNNs on specialized neuromorphic hardware. The energy efficiency of the networks is thus a high priority in the design of application-oriented SNNs. However, the use of neuromorphic hardware imposes additional constraints like the available number of neurons a network can comprise or its connectivity within the neuronal populations. Similar to embedded applications, the networks should be minimal to reduce the hardware costs and power consumption. We, therefore, ask the following RQs:

Research Question 3
How can the complexity of the SNN realizations be reduced?

Research Question 4
How do the network complexity and performance scale?

1.2 Structure

This work is divided into eight chapters. An overview of the structure of this work is given in Figure 1.1.

This first chapter motivates the work and introduces the problem statement. Subsequently, research questions are formulated that address the identified gap of knowledge. The chapter ends with a list of publications that make up this work or are directly related to it.

Chapter 2 sets out the basic principles for this work. This includes an introduction of the neuron models, which are used in this thesis, as well as an overview of bio-inspired and artificial learning algorithms. Because we use pseudo-gradient-based backpropagation to train the networks in this work, we introduce the necessary background here. The chapter closes with an overview of current neuromorphic hardware accelerators and a review of encoding schemes, which can be deployed to encode information in a spike-based format.

The methodological sections of this work are described in chapters 3 and 4. Chapter 3 builds upon the encoding schemes that are reviewed in Section 2.4. Based on the characteristics of the data we will evaluate in the chosen applications, we distinguish between frame-based (Section 3.2) and stream (Section 3.3) encoding schemes. Specific to data sequences that comprise superpositions of sinusoidal components, we introduce the use of Resonate-and-Fire (RF) neurons as frequency selective input encoders in Section 3.4.

The second methodological chapter, chapter 4, introduces the algorithms and approaches that are used to train the SNNs in this work. Section 4.1 describes the underlying models and structures that are used to construct the SNNs. The algorithms to

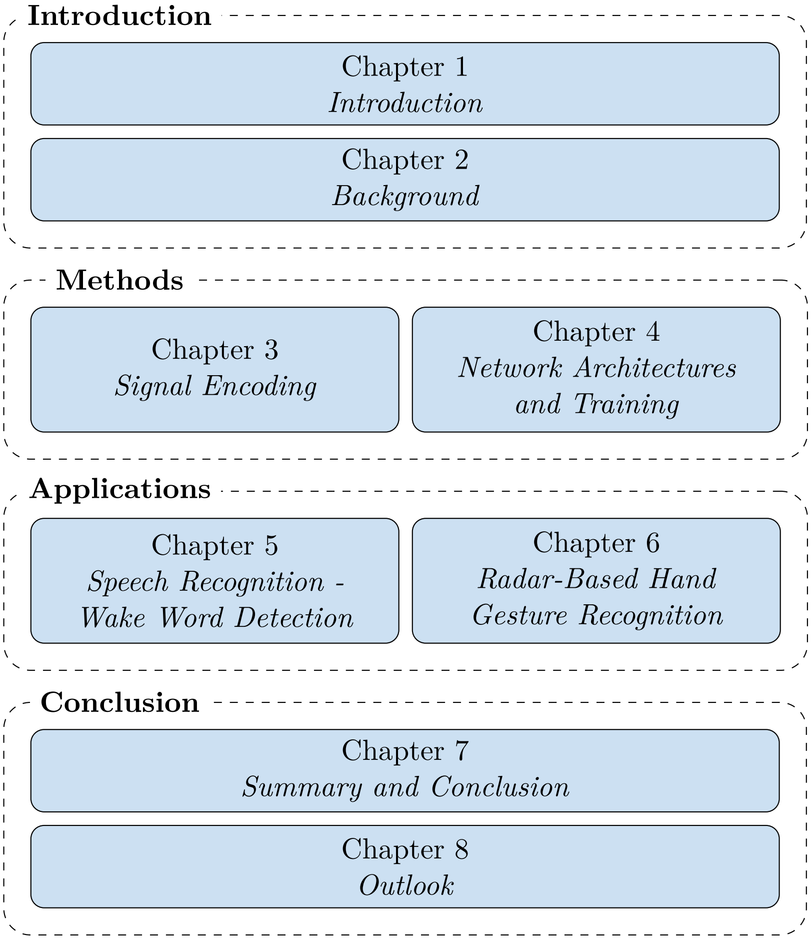


Figure 1.1: Structure of the thesis.

train these networks are subsequently introduced in Section 4.2. The chapter concludes with considerations on the subject of neuromorphic hardware. We discuss the relations between network architectures and the potential neuromorphic realizations.

Chapter 5 introduces the first application of small, low-power SNN implementations, which is discussed in this work: the recognition of keywords in audio data streams. The first two sections (5.1, 5.2) explain the motivation and the background for this application. The setup of the experiments is shown in Section 5.3. Here, we utilize the approaches that are described in the methodological chapters. Subsequently, the results of the experiments are evaluated in Section 5.4 and their implications are discussed in the last section of this chapter, Section 5.5.

In the second application, SNNs are used to detect hand gestures in sequences of radar measurements. This is elaborated in chapter 6. As in the previous chapter, the first two sections motivate the research and introduce the background for this application (Sections 6.1 and 6.2). The results of the experiments are evaluated in Section 6.4. In

Section 6.5, we discuss the results and draw parallels to the insights gained in the previous section.

In chapter 7, the results, findings, and limitations of the methodological and the application-oriented chapters are summarized. Finally, chapter 8 gives an outlook over future directions and topics of the application-oriented low-power usage of SNNs.

1.3 Contributions

Parts of this thesis have been previously published in peer-reviewed journals or at international peer-reviewed conferences.

To be able to answer the first RQ, a review article about encoding techniques in biological and artificial SNNs was composed and was published in a journal. Its content is reflected in Section 2.4 and forms the basis of chapter 3.

1. Daniel Auge, Julian Hille, Etienne Mueller, and Alois Knoll. **A Survey of Encoding Techniques for Signal Processing in Spiking Neural Networks.** *Neural Processing Letters*, 2021. [50]

The initial idea and evaluation of using RF neurons as frequency selective encoders for the use in SNNs has been published as a technical report. Its content is included in chapter 3.

2. Daniel Auge and Etienne Mueller. **Resonate-and-Fire Neurons as Frequency Selective Input Encoders for Spiking Neural Networks.** *TUM (Technical Report)*, 2020. [51]

The further analysis and application of the approach to speech recognition has been published as a conference paper and will be elaborated in chapter 5. The proposed methods are part of chapters 3 and 4.

3. Daniel Auge, Julian Hille, Felix Kreutz, Etienne Mueller, and Alois Knoll. **End-to-End Spiking Neural Network for Speech Recognition Using Resonating Input Neurons.** *30th International Conference on Artificial Neural Networks (ICANN)*, 2021. [52]

As RF neurons can encode spectral information of arbitrary signals, we show their use for the detection of interference in radar applications. The evaluation of this application, however, is not included in this thesis.

4. Julian Hille, Daniel Auge, Cyprian Grassmann, Alois Knoll. **Resonate-and-Fire Neurons for Radar Interference Detection.** *International Conference on Neuromorphic Systems (ICONS)*, 2022. [53]

The idea to classify hand gestures based on radar data using SNNs was presented as a poster:

5. Daniel Auge, Philipp Wenner, Etienne Mueller. **Hand Gesture Recognition using Hierarchical Temporal Memory on Radar Sequence Data.** *Bernstein Conference 2020*, 2020. [54]

Although the specific approach of the above-mentioned publication [54] has been discarded, we continued to pursue the idea. The resulting approach, which involved recurrently connected SNNs and an evaluation of binarization techniques, was published as a conference paper. The methodological parts of the paper are included in chapters 3 and 4, whereas the experimental part of the paper is reflected in chapter 6.

6. Daniel Auge, Julian Hille, Etienne Mueller, and Alois Knoll. **Hand Gesture Recognition in Range-Doppler Images Using Binary Activated Spiking Neural Networks.** *IEEE International Conference on Automatic Face and Gesture Recognition 2021*, 2021. [55]

The KI-ASIC project [56], funded by the German Bundesministerium für Bildung und Forschung (BMBF), is about evaluating the use of SNNs in the field of automotive radar processing. Some of the knowledge gained in this thesis is reflected in the joint journal contribution by all project members:

7. Bernhard Vogginger, Felix Kreutz, Javier López Randulfe, Chen Liu, Robin Dietrich, Hector A. Gonzalez, Daniel Scholz, Nico Reeb, Daniel Auge, Julian Hille, Muhammad Arsalan, Florian Mirus, Cyprian Grassmann, Alois Knoll, and Christian Mayr. **Automotive Radar Processing with Spiking Neural Networks: Concepts and Challenges.** *Frontiers in Neuroscience*, 2022. [57]

The following works also contribute to the field of SNNs but are not directly part of this thesis.

8. Etienne Mueller, Julius Hansjakob, Daniel Auge, Alois Knoll. **Minimizing Inference Time: Optimization Methods for Converted Deep Spiking Neural Networks.** *2021 International Joint Conference on Neural Networks (IJCNN)*, 2021. [34]

9. Etienne Mueller, Viktor Studenyak, Daniel Auge, Alois Knoll. **Spiking Transformer Networks: A Rate Coded Approach for Processing Sequential Data.** *International Conference on Systems and Informatics (ICSAI)*, 2021. [58]

10. Etienne Mueller, Daniel Auge, Simon Klimaschka, Alois Knoll. **Neural Oscillations for Energy-Efficient Hardware Implementation of Sparsely Activated Deep Spiking Neural Networks.** *AAAI's International Workshop on Practical Deep Learning in the Wild*, 2022. [59]

The following works contribute to the field of time series processing of radar or camera data in the automotive field with ANNs. They are not directly part of this thesis but discuss neighboring fields which could be applications of SNNs in the future.

11. Julian Hille, Daniel Auge, Cyprian Grassmann, Alois Knoll. **FMCW radar2radar Interference Detection with a Recurrent Neural Network.** *2022 IEEE Radar Conference (RadarConf)*, 2022. [60]

12. Saasha Nair, Sina Shafaei, Daniel Auge and Alois Knoll. **An Evaluation of "Crash Prediction Networks" (CPN) for Autonomous Driving Scenarios in CARLA Simulator.** *SafeAI 2021 - AAAI's Workshop on Artificial Intelligence Safety*, 2021. [61]

2 Background

This chapter presents the relevant background and related works for this approach. We, first, introduce the base neuron models, which are used in this work. Subsequently, we give an overview of learning algorithms for the use for SNNs. In real-world applications, SNNs are executed on specialized hardware to ensure their energy-efficient execution. An overview of current solutions in this field of research is given in the third section of this chapter. The chapter closes with a review of encoding schemes, which can be used to encode real-world information into trains of spike events.

The related works, which are specific to the respective application in the later chapters, are given in Sections 5.2 and 6.2.

2.1 Neuron Models

The dynamics of spiking neurons are described using analytical models consisting of differential equation systems. Models have been developed, which closely resemble biological behavior, whereas others show only abstracted behaviors. The choice for the respective model to be used primarily depends on the considered system. Researchers interested in simulating distinct ion movements within the neuron cells will most certainly need a different abstraction level than someone who examines the behavior of large neuron populations and their interactions on a network level.

Throughout this work, mainly two different neuron models are utilized to construct SNNs: Leaky Integrate-and-Fire (LIF) neurons and Resonate-and-Fire (RF) neurons. From Izhikevich's list of neuro-computation features [27], the LIF neuron implements tonic spiking (persistent generation of output spikes as long as the input current is present), class 1 excitability (spike frequency proportional to the input current), and the property of an integrator. The RF neuron additionally implements, as the name suggests, the properties of a resonator and further relations between stimulus and spike response. The mathematical descriptions of these neuron models are presented in the following. We introduce the Hodgkin-Huxley model for completeness because it is the first formal description of a neuron and forms the basis for all neuron models.

2.1.1 Hodgkin-Huxley Model

Hodgkin and Huxley developed the first biologically feasible neuron model in 1952. They performed experiments on the squid's nervous system to study the chemical and electrical processes of the nerve cell and its surrounding. They found that the transport of ions using channels between the inside and outside the nerve cell is used to adapt the cell's membrane potential. They identified sodium and potassium ions as the main carriers for charge transport in the cell. The Hodgkin-Huxley model describes those transports and concentrations by a four-dimensional differential equation system [62].

The equation describing the membrane potential $u(t)$ is given by

$$C\frac{\mathrm{d}v}{\mathrm{d}t} = -\sum_k i_k(t) + i(t). \tag{2.1}$$

Thereby, C describes the membrane capacity, $i(t)$ the input current and $i_k(t)$ the sum of the internal ion currents

$$\sum_k I_k(t) = g_{\mathrm{Na}} m^3 h(v - v_{\mathrm{Na}}) + g_{\mathrm{K}} n^4 (v - v_{\mathrm{K}}) + g_{\mathrm{L}}(v - v_{\mathrm{L}}) \tag{2.2}$$

introduced by the movement of sodium (Na) and potassium (K) ions as well as a leakage current (L), which mainly consists of chlorine ions. v_{Na}, v_{K}, and v_{L} are the respective reversal potentials, which have been empirically determined by Hodgkin and Huxley. The variables n, m, and h are described by differential equations themselves of the form

$$\frac{\mathrm{d}x}{\mathrm{d}t} = -\frac{1}{\tau_x(v)}\left[x - x_0(v)\right]. \tag{2.3}$$

Through measurements on the squid, they empirically determined the necessary time constants and reverse potentials to complete the formulation. In this biologically plausible model, what is often referred to as a spike is, in fact, the rapid rise and fall of the neuron's membrane potential, enabled by the interlinked movements of ions.

The four-dimensional Hodgkin-Huxley model can be reduced to a nonlinear two-dimensional model by approximating its temporal characteristics. There are two simplifying approximations involved in the reduction of the dimensions [63]: (1) The temporal evolution of the m variable is much faster than that of the two other gating variables n and h. Consequently, the variable m is treated as an instantaneous variable, thus $m(t) \mapsto m_0\left[v(t)\right]$. (2) The time constants τ_n and τ_m have similar temporal dynamics, suggesting approaching the two variables n and h by a single variable v_2. This simplifies eqs. (2.1) to (2.3) to

$$\begin{aligned} \frac{\mathrm{d}v}{\mathrm{d}t} &= \frac{1}{\tau}\left[f(v, v_2) + R\, i(t)\right] \quad \text{and} \\ \frac{\mathrm{d}v_2}{\mathrm{d}t} &= \frac{1}{\tau_2} g(v, v_2). \end{aligned} \tag{2.4}$$

2.1.2 Leaky Integrate-and-fire Model

The most simplified result of the dimensional reduction of the Hodgkin-Huxley model is the LIF neuron model. Due to its simplicity, it can be computed efficiently. The general

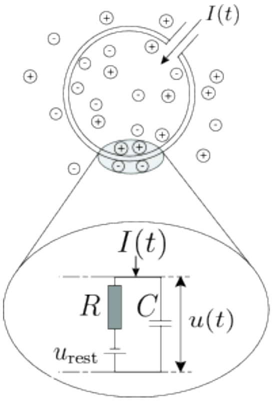


Figure 2.1: Equivalent electrical circuit of the LIF neuron. The membrane potential is present at the capacitance. The leakage current is modeled by the parallel resistor, which causes the voltage to decrease to the rest potential. (Figure from [63])

dynamic of the membrane potential $v(t)$ is given by the differential equation

$$C\frac{\mathrm{d}v}{\mathrm{d}t} = -\frac{1}{R}v(t) + i(t), \tag{2.5}$$

with C being the membrane capacitance, R being the input resistance, and $i(t)$ the input current. A schematic and the equivalent electrical circuit are depicted in Figure 2.1. Often, the membrane capacitance and the input resistance of the neuron are combined to a single value, namely the time constant τ of the neuron. The time constant defines the voltage leakage of the neuron as the factor of the exponential decay of the membrane potential. As soon as the membrane potential crosses the threshold value v_{th}, the LIF neuron fires an output spike, and the membrane potential is lowered to the reset potential v_{reset}. After generating a spike, the membrane can be fixed at the reset potential for a specific time, representing the neuron's refractory period.

The input current can be divided into a continuous stimulation i_{cont} as well as the contribution of other neurons connected via simple synapses with weights $w_{i,j}$:

$$i_i(t) = i_{\text{cont}}(t) + \sum_{f,j} w_{i,j}\delta(t - t_j^f). \tag{2.6}$$

Here, incoming spikes are modeled as delta functions $\delta(t)$, which are zero at every point in time except for $t = 0$ and $\int_{-\infty}^{\infty} \delta(x)dx = 1$. The spikes at the firing times t^f of each neuron j are thus summed and weighted. Incoming spikes, therefore, result in an instantaneous change of the membrane voltage. Weights can be positive and negative to comply with excitatory and inhibitory action potentials in their biological archetypes. An exemplary course of the membrane voltage of a neuron modeled as LIF neuron and Hodgkin-Huxley neuron is depicted in Figure 2.2.

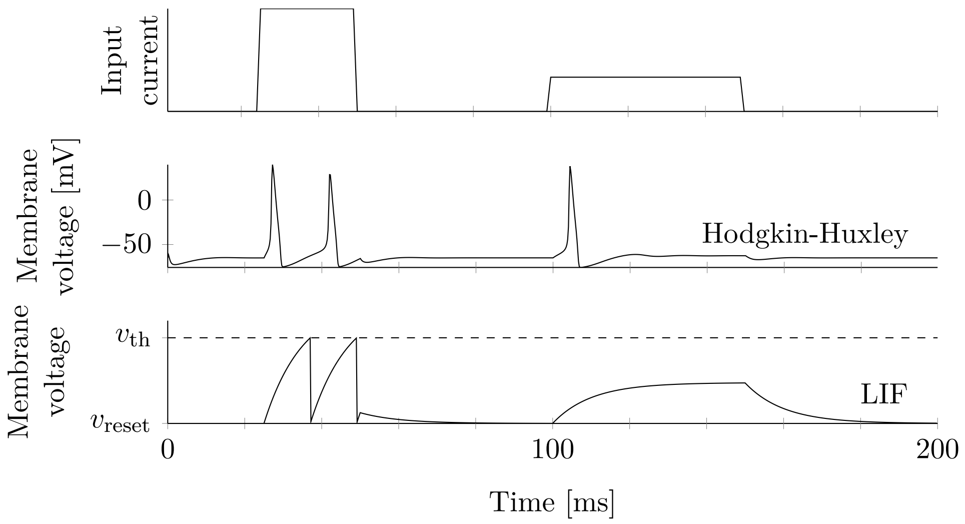


Figure 2.2: Voltage courses of LIF and Hodgkin-Huxley neurons. The exemplary current in the upper plot is applied to the two neuron models. The first current onset causes both neurons to produce spikes. The voltage course of the Hodgkin-Huxley model is more complex due to the temporal evolution of the variables n, m, and h, which are not depicted. The second current is not able to charge the LIF neuron enough to reach the threshold voltage. However, the sudden onset of the current leads to an action potential of the Hodgkin-Huxley model.

Variants

In the family of integrate-and-fire neurons exist further variants of the LIF neuron, which implement more or less neuro-computational features, as shown by Izhikevich [27]. Examples include the non-leaky integrate-and-fire neuron, which does not implement an exponential decay of the membrane voltage over time, the quadratic and exponential LIF neuron, which implement a second, bistable threshold, or the Adaptive Leaky Integrate-and-Fire (ALIF) neuron, which exhibits an adaption of the firing threshold, depending on the spike activity of the respective neuron.

2.1.3 Resonate-and-fire Model

The RF neuron is a two-dimensional neuron model, which was proposed by Izhikevich [64]. Its two cross-coupled membranes v_1 and v_2 oscillate when the neuron gets excited and begin to resonate if the stimulus matches the resonant frequency of the neuron:

$$\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = \begin{bmatrix} -\lambda & -\omega \\ \omega & -\lambda \end{bmatrix} \boldsymbol{v} + \begin{bmatrix} i(t) \\ 0 \end{bmatrix}. \tag{2.7}$$

Izhikevich describes the two variables y and v as the current- and voltage-like variables. The coupling is given by the resonant frequency $\omega = 2\pi f$ of the neuron. Additionally, each membrane is subject to damping, described by the damping constant λ. Accordingly,

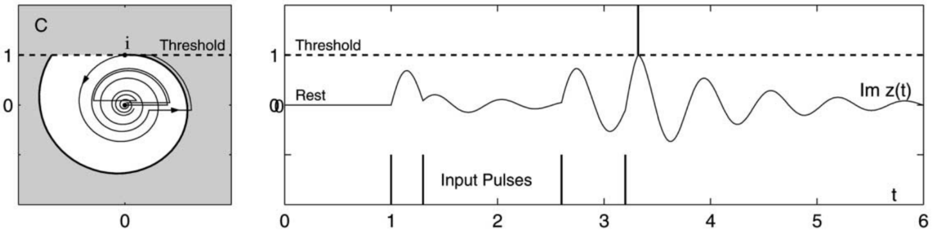


Figure 2.3: RF neuron by Izhikevich. Input spikes must arrive in the correct time interval to excite the neuron enough to produce an output spike. The left plot shows the temporal evolution of the two coupled variables in the complex plane. A spike is emitted when the course crosses the threshold at i. This corresponds to the voltage-like variable $\mathrm{Im}(z)$ crossing the threshold. (Figure from [64])

an outgoing spike is generated as soon as the voltage-like variable v reaches the firing threshold. The variables are subsequently reset to their resting potential. The equation system can alternatively be represented in a complex form with $\underline{v} = v_1 + \mathrm{i}v_2 \in \mathbb{C}$ with $\mathrm{Im}(\underline{v})$ being the voltage-like variable:

$$\frac{\mathrm{d}\underline{v}}{\mathrm{d}t} = (-\lambda + \mathrm{i}\omega)\underline{v} + i(t). \tag{2.8}$$

Figure 2.3 shows the course of the voltage-like variable when stimulated with non-resonant and resonant input pulse trains. The voltage-like variable only reaches the firing threshold if the correct time interval is present between succeeding spikes. The neuron does also generate spikes when stimulated with multiple near-simultaneous input spikes or when a combination of inhibitory and excitatory pulses at the correct time interval is present at the input.

Apart from the simple, symmetric RF mode, other neuron models can also exhibit resonant behavior [27]. Numerous works analyze the properties of these models [65]–[69]. Silicon-based integrated circuit realizations of RF neurons have been demonstrated in [70] and [71].

2.2 Training

The training of neural networks refers to adapting the networks' weights to achieve the desired input-output functionality. In the supervised learning scheme, the networks learn the desired relations using exemplary input-output pairs. In common non-spiking artificial neural networks, this is often done by propagating the error terms from the network's output back to its inputs. Using automatic differentiation tools, the gradients of the errors with respect to the network weights are used to optimize the connections strengths of the network. Since spike trains are represented by Dirac pulse combs, and the spiking neurons' activation functions are modeled as step functions, the backpropagation mechanism cannot be transferred directly to pulsed networks due to the non-existing derivative of the spike neuron's activation function. Therefore, adaptions to the known training schemes of neural networks or the development of new, often bio-plausible, methods are being developed.

2.2.1 Biologically Plausible Learning Algorithms

The most prominent biologically plausible learning rule is Spike-Timing-Dependent Plasticity (STDP), which is based on Hebb's rule [72]. STDP is best described with the well-known phrase "cells that fire together, wire together" [73]. If a postsynaptic neuron fires shortly (e.g., ≈ 10 ms) after a presynaptic spike, its weight is strengthened or weakened in the opposite case. This unsupervised learning rule uses information that is local to the synapse and local in time to adapt the synaptic weight. Masquelier and Thorpe showed the successful learning of visual features using STDP [74]. Later, Masquelier et al. showed that a single unsupervised STDP trained LIF neuron is able to recognize one or multiple repeating spike patterns in a noisy environment [75], [76]. In [77], Diehl et al. used an unsupervised STDP-trained network to classify digits in images of 28x28 pixels of the MNIST dataset. Spike rates thereby coded the pixel values. Kheradpisheh et al. applied the STDP learning rule to train a convolutional network [78]. They used Difference of Gaussians filters to convert the pixel values into spike events. Often, in STDP learning-based networks, Winner-Takes-All (WTA) circuits are used to support the learning process. In those circuits, lateral inhibitory connections between neurons of one layer are generated. They inhibit all neighboring neurons whenever the first neuron emits a postsynaptic spike. By doing so, the own connection to the neuron sending the presynaptic spike is strengthened, while all other connections are weakened. With that approach, neurons specialize more to specific inputs while having a highly diverse response among many neurons. A variant of STDP, Spike-Driven Synaptic Plasticity (SDSP) [79] adapts the synaptic weights of incoming spikes based on the current membrane voltage. The weight is strengthened if the potential is above a certain threshold during an incoming presynaptic spike; otherwise, it is weakened. At the end of the training, the weights learned by this bi-stable method are either one or zero (the minimum or maximum allowed weight, respectively), which enables an efficient implementation in analog Very-Large-Scale Integration (VLSI) circuits or Field-Programmable Gate Arrays (FPGAs) [79]. In [80], Kasabov et al. use STDP and SDSP methods, for the training of Evolving Spiking Neural Networks (eSNNs) to recognize patterns in spatio- and spectro-temporal signals. In supervised and reinforcement-based learning settings, STDP can be modulated to match the desired behavior. In these reward-modulated schemes, the teacher decides whether the closely emitted spikes were beneficial for the overall goal [81], [82]. The weight update is then adapted accordingly.

2.2.2 Artificial Learning Algorithms

ReSuMe (Remote Supervised Method) [83], Chronotron [84] and SPAN (spike pattern association neuron) [85] are supervised learning techniques and attempt to teach a neuron to fire at exact desired times. All three models use metrics to compare the desired and observed outputs following a presynaptic input. ReSuMe and SPAN are based on the

Widrow-Hoff rule [86] to compute the weight adaption Δw. The Chronotron uses the adapted Victor-Purpora distance metric [87] to gain a piecewise differentiable function. This function can subsequently be used for gradient descent methods. The distance metric is a function describing the cost of transforming one spike train into the other.

As stated earlier, backpropagation cannot be transferred to spiking neural networks due to the missing derivative of the activation function. Examples for adaptions are Lee et al., who use the membrane potential as a differentiable signal and treat jumps of the potential due to incoming spikes as noise [88]. With that, they trained a multilayered fully connected spiking neural network to reach a performance of 98.88% on the MNIST dataset. A similarly trained convolutional spiking neural network reached a higher performance of 99.31% on the same dataset, which is comparable with but still inferior to non-spiking variants.

A form of free-energy based reinforcement learning is shown in [89]. There, the free energy of a Restricted Boltzmann Machine (RBM) is decreased by temporal difference learning. The idea is explained in [90], [91] and was transferred to SNNs. Neftci et al. present in [33] the adapted "event-driven" contrastive divergence learning rule for RBMs together with STDP as weight update rule.

The synaptic weights of SNNs can also be parameterized by training a non-spiking network and transferring the weights to it. This learning method is called offline, conversion, or transfer learning. By this, the well-known learning mechanisms of common neural networks can be exploited while leveraging the power-efficient characteristics of the spiking neurons. Often, the real-valued activations are thereby replaced by rate-based coding (see Section 2.4). An example using this approach with a high achieved performance on the MNIST dataset can be found in [92] and [93].

A tighter coupling between ANN and SNN training is given in tandem learning approaches [35]. Here, the forward and backward paths during training are separated. The forward path, which computes the output of the network, is conducted within the SNN. This output forms the basis for the calculation of the training error. The backward path, which provides the gradients needed for the weight updates, is performed in an equivalent ANN. The weights are shared between both networks.

A comprehensive summary of supervised learning techniques developed until 2006 can be found in [94]. Newer, also deep learning methods are summarized in [45].

2.2.3 Backpropagation

In recent works, using the well-known backpropagation technique gained traction within the SNN research community. Backpropagation is used in standard ANNs as the main approach to train neural networks. It became viable for the use in SNNs by the introduction of surrogate gradients [46]–[48].

Backpropagation describes the network's loss function computation with respect to the connection weights. In a supervised learning approach, for example, the cost or loss function $\mathcal{L}$ is minimized over a dataset of known input and target values $\{\boldsymbol{x}, \hat{\boldsymbol{y}}\}$. Following the gradient descent algorithm, the network weights $\boldsymbol{W}$ have to be updated in the opposite direction of the losses' gradient to minimize the loss [1].

The magnitude of this update is thereby scaled by the learning rate η:

$$w_{i,j} \leftarrow w_{i,j} - \eta \Delta w_{i,j}. \tag{2.9}$$

Using the chain rule

$$\Delta w_{i,j} = \frac{\partial \mathcal{L}}{\partial w_{i,j}} = \frac{\partial \mathcal{L}}{\partial y_i} \frac{\partial y_i}{\partial a_i} \frac{\partial a_i}{\partial w_{i,j}}, \tag{2.10}$$

the gradient of the loss with respect to the network weights $\boldsymbol{W}$ can be computed. With that, the weight update $\Delta \boldsymbol{W}$ is derived. The fraction $\frac{\partial y_i}{\partial a_i}$ relates the layer's output and its activation, which corresponds to its activation function. Therefore, the activation function needs to be differentiable to be used here.

The use of the chain rule in eq. (2.10) is called backpropagation in neural networks since the gradient of the loss is backpropagated from the output towards the input of the network. This solves the spatial credit assignment problem, which states which node contributed to the final output to which extent [95].

Backpropagation Through Time

When the input to the network consists of temporal sequences and the network has got stateful variables, the temporal credit assignment problem has to be solved additionally [95]. The training algorithm has thus not only to decide which node contributed to the final output, but also at which point in time.

Backpropagation Through Time (BPTT) is a temporal extension of the gradient-based backpropagation algorithm, which solves the above-mentioned credit assignment problems. It is used to train Recurrent Neural Networks (RNNs) [96]. The recurrent network is unfolded in time to enable the backpropagation of the error through the network and its past states. Accordingly, long temporal sequences result in humongous networks comparable with the depth of deep neural networks. This introduces the problem of vanishing and exploding gradients due to the multiplicative linkage of the losses, gradients, and parameters by the chain rule [97], [98]. However, this problem can be reduced by limiting the length of the temporal sequences, by closely monitoring the backpropagated gradients, or by using specialized neuron models like the Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) cells. These specialized cells allow an unaltered backpropagation of the error signal [99], [100].

The forward path through the network comprises the computation of the neurons' activation values $\boldsymbol{a}^{(l)}[t]$ and their outputs $\boldsymbol{y}^{(l)}[t]$ for each layer l and point in time t:

$$\begin{aligned} \boldsymbol{y}^{(l)}[t] &= f^{(l)}\left(\boldsymbol{a}^{(l)}[t]\right) \\ \boldsymbol{a}^{(l)}[t] &= g^{(l)}\left(\boldsymbol{y}^{(l-1)}[t], \boldsymbol{y}^{(l)}[t-1], \boldsymbol{W}^{(l,l-1)}, \boldsymbol{W}^{(l,l)}\right). \end{aligned} \tag{2.11}$$

A schematic overview of the computation graph is shown in Figure 2.4. The activation function $f^{l}()$ can be any nonlinear function like the sigmoid function, tanh, or a step

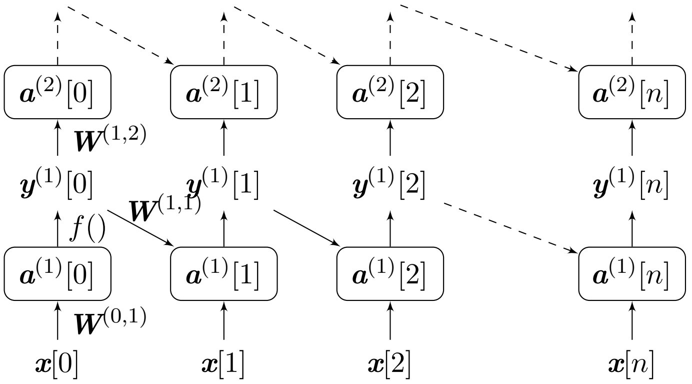

$\boldsymbol{a}^{(2)}[0]$
$\boldsymbol{a}^{(2)}[1]$
$\boldsymbol{a}^{(2)}[2]$
$\boldsymbol{a}^{(2)}[n]$
$\boldsymbol{W}^{(1,2)}$
$\boldsymbol{y}^{(1)}[0]$
$\boldsymbol{y}^{(1)}[1]$
$\boldsymbol{y}^{(1)}[2]$
$\boldsymbol{y}^{(1)}[n]$
$\boldsymbol{W}^{(1,1)}$
$f()$
$\boldsymbol{a}^{(1)}[0]$
$\boldsymbol{a}^{(1)}[1]$
$\boldsymbol{a}^{(1)}[2]$
$\boldsymbol{a}^{(1)}[n]$
$\boldsymbol{W}^{(0,1)}$
$\boldsymbol{x}[0]$
$\boldsymbol{x}[1]$
$\boldsymbol{x}[2]$
$\boldsymbol{x}[n]$

Figure 2.4: Schematic computational flow in recurrent networks. The recurrent network is unrolled in time. At every time step, the input to the layer $\boldsymbol{x}[t]$ and the output of the previous time step $\boldsymbol{y}[t-1]$ are used to compute the current layer's activation.

or delta function, as in the case of spiking networks. The activation value is computed by relating the layer's previous output $\boldsymbol{y}^{(l)}[t-1]$ and the output of the preceding layer $\boldsymbol{y}^{(l-1)}[t]$ with the respective weight matrices $\boldsymbol{W}$. In most cases, the relation $g^{(l)}()$ is a weighted summation, but it can also take on more complex functions. In the first layer, the input $\boldsymbol{x}[t]$ is used instead of $\boldsymbol{y}^{(l-1)}[t]$.

The loss for the input-target value pair x, y with the network output y_t^N is calculated by the loss function $C(y, y_t^N)$. The loss can be computed for each time step individually when a distinct ground-truth value is given for each sample. Alternatively, the network's output at the last time steps is used for the calculation for the whole sequence. The derived error is subsequently backpropagated through the unrolled network.

Pseudo Gradients

The activation function of a LIF neuron corresponds to a step function with an undefined derivative at the time of the step itself. Accordingly, the chain rule in eq. (2.10) is not applicable due to this the non-existent derivative of the relation between activation and output value $\frac{\partial y_i}{\partial a_i}$. Backpropagation therefore cannot be utilized trivially [49].

However, the introduction of surrogate gradients makes determining the weight updates possible. Proposed surrogate gradients alleviate the discontinuous true gradient based on the ratio between the neuron's current membrane voltage v and the threshold voltage v_{th}. The two most frequently used functions show linear [46], [47] or S-shaped [48] relations:

$$\begin{aligned}\text{Linear}: \psi_{\text{lin}} &= \max\left(1 - \left|\frac{v}{v_{\text{th}}}\right|, 0\right) \\ \text{Sigmoid}: \psi_{\text{sig}} &= a\ S\left(a\frac{v}{v_{\text{th}}}\right)\ S\left(-a\frac{v}{v_{\text{th}}}\right) \quad \text{with} \quad S(x) = \frac{1}{1+e^{-x}}\end{aligned} \tag{2.12}$$

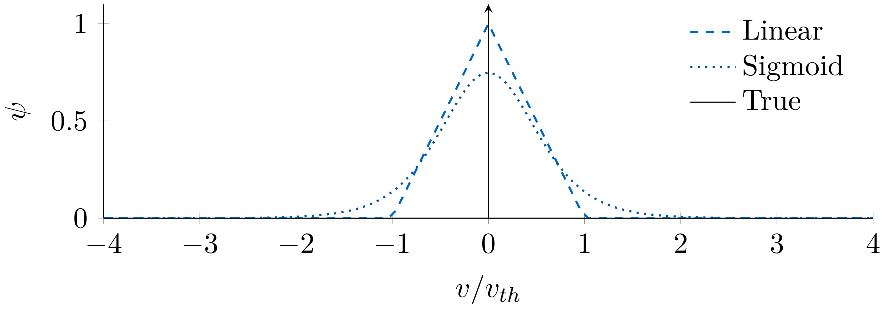


Figure 2.5: Surrogate gradients used for the training of SNNs. The gradient of the sigmoid activation function and the linear pseudo gradient can be used as a smooth out surrogate gradient of the non-differentiable step function.

as shown in Figure 2.5. For the sigmoid function, a is a factor, which can be used to control the steepness of the gradient around $\frac{v}{v_{\text{th}}} \approx 0$. The gradient does thus not depend on the spike event itself but on the relation between the membrane potential and the threshold voltage.

2.3 Neuromorphic Hardware

Neuromorphic hardware is specifically designed to leverage the distributed, event-based operations of SNNs. They often circumvent the von Neumann bottleneck of classical digital network designs by architectures using in-memory computing. Other realizations implement the network using analog structures without digital processing elements as main compute units.

Many currently available neuromorphic processing chips provide scalable multipurpose systems primarily for research purposes [21], [101], [102]. With accelerators like these, arbitrarily large networks can be realized without the hardware being the limiting factor. Projects like the Human Brain Project [103] or the Brain Initiative [104] even intend to scale these accelerators up to systems which are capable of simulating the activity of a human brain with hundreds of millions of neurons. Other works, in contrast, propose specialized circuits, which focus heavily on a high efficiency [105], [106]. The third category of accelerators comprises approaches using FPGAs, which thus do not necessitate the production of specialized Application-Specific Integrated Circuits (ASICs). Table 2.1 lists an overview of existing neuromorphic accelerators. Especially in the second category, many more approaches and demonstrators exist, highlighting new architectures and circuits, which improve specific details within the area of neuromorphic computing.

There are a multitude of reviews, which examine certain aspects of neuromorphic hardware. Most of them thereby base their analyses on the general-purpose accelerators. As one of the first reviews, [107] recapitulates the first generations of neuromorphic hardware regarding their large-scale application. Further reviews examine the topic of neuromorphic computing in a broader way, providing also the temporal context beginning with early developments in the 1980s [108], [109]. The authors of [110] and [32] focus their review on the low-power signal processing capabilities of the hardware and on the communication involved between the neurons, respectively. More recent reviews reflect the state of the art and identify the current trends in this rapidly developing field [111]–[113].

2.3.1 Software Simulation Environments

Before bringing SNNs to hardware accelerators, their time-dependent behavior has to be modeled. The four most known tools for this purpose, NEURON [114], GENESIS [115], Brian 2 [116] and NEST [117], are evaluated in [118]. Due to its good performance at large networks and its scalability to computer clusters, the NEST simulator is used in the Neurorobotics platform [119] of the Human Brain Project. Brian 2 convinces through simple operation, high efficiency in terms of code length and a comprehensive documentation. NEURON and GENESIS are the simulator, which are most used based on their citations. However, they are primarily used to precisely model biological behaviors. Many of the software frameworks have a standardized software interface, PyNN [120], and feature built-in interfaces for the use of neuromorphic hardware accelerators.

Lately, also, the automatic differentiation frameworks TensorFlow [121] and PyTorch [122] gained traction within the community. While they are primarily used for the training of second-generation networks, advances in pseudo gradient-based learning schemes in SNNs made them suitable for the simulation of third generation networks, too.

2.3.2 Multipurpose Hardware Accelerators

HICANN/BrainScaleS

High Input Count Analog Neural Networks (HICANNs) are the chiplet building blocks of the hierarchical wafer-scale neuromorphic system BrainScaleS [123]. Its architecture and communication schemes are optimized for large network systems.

HICANNs implement Adaptive Exponential Integrate-and-Fire (AEIF) neurons [124], which can be reduced to the standard integrate-and-fire model if necessary. The timescale of the implementation is quite fast, resulting in a 10^3 to 10^5 acceleration of the biological equivalent circuits. The time constants of the neurons are tuned by adjusting the capacitance of the membrane and the leakage conductance. The small time constants result from small electrical capacities, which in turn have small physical dimensions. As the membranes are implemented as Metal-Insulator-Metal (MIM)-capacitors, they do not occupy any additional surface area.

BrainScaleS' neurons are implemented in Analog Network Cores. Each core consists of 512 membranes (which can be combined to neurons), two blocks of synapse arrays and synaptic drivers. The synaptic drivers convert the digital address events to analog currents using 4-bit Digital-to-Analog Converters (DACs). At maximum, a neuron can be fed with over 14 k synapses.

One wafer houses a total of 352 HICANN chiplets resulting in up to 180k neurons to be simulated. BrainScaleS' reference system consists of 20 wafers in total.

Table 2.1: Neuromorphic hardware solutions.

Name		**Year**	**Neurons**	**Synapses**	**Models**	**Domain**	**Process** [nm]	**Die size** [mm^2]
Human brain			10^{11}	10^{15}	Various	Electrochemical		
HiCANN	[123]	2010	512	100k	AEIF	Mixed-signal	180	55
HiCANN-DLS	[125]	2016	512	131k	LIF	Mixed-signal	65	32
SpiNNaker	[126]	2014	16k	16M	Arbitrary	Digital (software)	130	100
SpiNNaker 2	[102]	2019			Arbitrary	Digital (software)	22	
NeuroGrid	[127]	2014	64k			Mixed-Signal	180	168
Braindrop	[128]	2018	4096	65k		Mixed-signal	28	0.65
TrueNorth	[101]	2014	1M	256M	LIF	Digital	28	430
DYNAP-SEL	[129]	2017	1088	78k	LIF	Mixed-signal	180/28	7.28
Loihi	[21]	2018	131k	130M	LIF	Digital	14	60
Darwin	[130]	2016	2048	4M	LIF	Digital	180	25
ODIN	[131]	2019	256	64k	LIF/Izhikevich	Digital	28	0.086*
Rolls	[132]	2015	256		AEIF	Mixed-Signal	180	51.4
Buhler2017	[133]	2017	512		LIF	Mixed-Signal	40	
Chen2018	[134]	2018	4096	1M	LIF	Digital	10	1.72
Cho2019	[135]	2019	2048	149k	LIF	Digital	40	2.56
Park2019	[136]	2019	410	200k	Sigmoid	Digital	65	10
Tianjic	[137]	2019	40k	10M	hybrid	Digital	28	14.5
μBrain	[105]	2021	336	20k	LIF	Digital	40	2.82
Kuang2021	[106]	2021	1024	1M	LIF	Digital	28	3.66
Kuang2021	[138]	2021	64k	64M	LIF	Digital	65	107
Zhong2021	[139]	2021	1024	256k	LIF	Digital	28	1.41

*Without peripherals and pads.

The second generation of the HICANN chiplet is the High Input Count Analog Neural Network with Digital Learning System (HICANN-DLS) [125]. It features newer manufacturing processes, neurons and synapses with higher bit-precision, and onboard learning capabilities.

SpiNNaker and SpiNNaker 2

SpiNNaker is an ARM microcontroller-based multicore platform with the goal to simulate large-scale networks up to the size of the human brain [126]. A SpiNNaker machine can incorporate an arbitrary number of processing nodes, making it possible to be scaled to any desirable size. Each processing node in this architecture consists of 18 cores within one package with local memory, shared memory, and a packet router, which controls the communication within the node and between nodes. The processors in the nodes are freely programmable and can simulate a few hundred neurons, depending on the complexity of the neural model, the synaptic model, and the learning rule. Depending on the application, many of those nodes can be operated in parallel, resulting in a large simulation capability.

SpiNNaker's successor, SpiNNaker 2 [102], aims to increase the simulation capacity by a factor greater than 50 with numerous improvements over the older architecture and a much smaller production technology (130 nm versus 22 nm). The architectural improvements include hardware-accelerated processing and adaptive voltage and frequency scaling depending on the current workload of the nodes.

Braindrop and Neurogrid

Braindrop [128] is a mixed-signal processor, primarily made for the simulation of Neural Engineering Framework (NEF) [140] networks. The software framework features a translation of the abstract formulation of e.g. differential equations onto the hardware. The authors claim that they thereby leverage – and are even reliant on – the mismatch of the analog circuits for the needed neuronal variability in the network. Braindrop implements 4096 neurons and a weight memory of 64 KB, which is designed to fit 16 8-bit synapses per neuron.

The power consumption of large neuromorphic systems is largely influenced by the digital communication. The authors therefore focus on sparsifying the communication schemes in space and time (sparse encoding and accumulative thinning). The communication is implemented digitally, the neurons themselves are analog implementations. The neurons are designed to work in the subthreshold region. Due to the mixed-signal design and the specific communication scheme, Braindrop shows to have a much higher neuronal density and a vastly lower energy consumption per synaptic operation compared to the fully digital architectures of its peers [128].

Braindrop is the successor of the large-scale system Neurogrid [127]. Braindrop itself will be a building block of the planned Brainstorm chip, which is designed for realizations containing millions of neurons in a multicore system [128].

TrueNorth

TrueNorth [101] is a fully digital platform, build for the simulation of one million neurons. Each chip contains 4,096 neurosynaptic cores. The cores hold a self-contained neural network comprising 256 LIF neurons with 256 input lines, 256 output lines, and a 256×256 matrix to specify the synaptic connections.

In total, a single TrueNorth chip can house one million neurons with 256 million synapses. To enable the simulation of larger networks, multiple chips can be connected to larger clusters.

DYNAP-SEL

The DYNAP-SEL chip is a mixed-signal processor with a scalable routing architecture, which makes it possible to combine multiple chips [129]. Each chip comprises four neural processing cores and one core, which features additional plastic synapses. Each of the non-plastic cores houses 256 neurons and 64 4-bit synapses per included neuron. The additional core has got 64 neurons with 128 plastic synapses with on-chip learning and 64 programmable synapses per neuron.

This unique architecture enables the simulation of large networks with efficient on-chip learning, structural plasticity, or biological evidence.

Loihi

The Loihi chip [21] implements 128 digital neuromorphic cores. Each core can simulate 1,024 neurons, which share the same fan-in connections, fan-out connections, and configurations. Ten memory blocks contain the spike traces for learning, synapse memories and mappings, and the neuronal state variables. Loihi implements the LIF model with current-based synapses. The on-chip learning scheme, however, is freely programmable using 4-bit microcode.

Whenever spikes are generated by a neuromorphic core, the events are propagated through the grid structure of the chip. Synchronizing messages are used to ensure that all events have been processed before entering the next global time step. Apart from this synchronization, the cores operate asynchronously.

Similar to SpiNNaker, the Loihi architecture is scalable. Therefore, multiple chips can be combined to form clusters. The largest system based on Loihi contains 768 chips with a total of 98,304 cores and 100M neurons.

A successor to Loihi, Loihi 2, has been brought up recently [141], but information about the underlying architecture has not been published yet.

2.3.3 Specialized Circuits

Apart from the popular general-purpose neuromorphic hardware solutions, there exists a variety other of approaches.

μBrain [105] is an interesting example of specialized, purpose-built neuromorphic hardware. It is a fully digital layer-based architecture, which uses overflowing digital accumulators as neurons and implements fixed size integer synapse weights. The architecture uses feedforward and recurrent connections between the layers. The concrete implementation of the μBrain architecture is synthesized, thus fixing most of the network directly into silicon. This includes the network topology, connectivity, and the bit precision of the individual calculations. Only neuron parameters and synaptic weights are programmable during runtime. The authors showcase μBrain's design with a 336-neuron network realization, though arbitrary architectures are possible. The core concept of the architecture is the spike arbiter and the delay cell, which make the fully asynchronous operation possible. The arbiter resolves situations in which more than one spike arrive at the layer input at the same time. As a result, spikes are delayed by several ns (in contrast to the normal operation time constants of μs to ms).

The chip presented in [134] is an example for the design of accelerating networks with ultralow power consumption. This is reached by using a 10 nm Fin Field-Effect Transistor (FinFET) manufacturing process and architectural and algorithmic approaches like sparse neuronal connections and stochastic dropping of events. Buhler et al. demonstrate a chip, which achieves the same goal using analog neuron implementations [133].

2.3.4 FPGA-based Accelerators

FPGAs provide the possibility to accelerate neuromorphic processes using off-the-shelf hardware. Works like [142]–[146] thus propose certain architectures, which exploit the properties of FPGAs. DeepSouth [146], as an example, is a scalable architecture, which has been shown to be able to simulate billions of LIF neurons on the chosen FPGA board. Depending on the temporal simulation speed and available memory, the networks could be even larger. However, the energy efficiency FPGAs is much worse compared with ASICs. The flexibility, which is introduced by the programmable hardware, is thus accompanied by a higher energy consumption.

2.3.5 Summary

There exists a variety of different approaches, which enable the efficient execution of SNNs on specialized hardware. The solutions comprising hardware or software-based many-core systems, analog compute elements, or approaches using off-the-shelf compute hardware, for example, thereby pursue vastly different goals with the inherent advantages and disadvantages of their specific implementations.

The many-core systems like Loihi [21] and TrueNorth [101] implement digital processors, which are able to compute huge networks efficiently, but have limited flexibility in terms of realizable neuron and synapse models. SpiNNaker [126] is highly flexible because the neurons and synapses are computed entirely in software. This, however, comes at the cost of a higher energy consumption. Analog realizations can mimic biological behavior at speeds, that are magnitudes higher than their biological models. They are, however, influenced by the ambient temperature, process variations, and noise.

A universal solution, which fits all possible applications, thus does not exist. In this work, we will examine networks for embedded applications, which comprise a few hundred neurons. Most of the aforementioned accelerators, however, are designed for the simulation of large-scale networks. They are, therefore, suited to be used during prototyping, but for the final deployment in products, these accelerators are not appropriate due to their sheer physical size. Additionally, much of the overhead, which is only used for the simulation of large networks (communication buses, routers, schedulers), is not needed.

2.4 Review of Encoding Schemes

The first challenge when utilizing SNNs is the representation of information. Since spikes are the main information carrier in SNNs, incoming data has to be encoded for the network to be processed.

A look at the biology already provides the intuition that there exist different coding schemes, which are specialized for the type of data they are used for. Photo receptors in our eyes encode light intensities into spike patterns, microscopic hair cells in the inner ear transform changes in the air pressure into frequency-selective spike trains, and our nose has got chemical receptors, which emit spikes as soon as specific molecules are perceived within the air. The underlying question, however, is how these spike trains exactly look like, and how they represent specific information.[1]

2.4.1 Taxonomy

Coding schemes can be categorized into two clusters: rate codes and temporal codes [63]. The main differentiation between those two schemes is the distinction whether exact spike times and the order of spikes is crucial for the information to be transmitted. In rate codes, information is embedded into the instantaneous or averaged spike rate of a single neuron or a larger population. Temporal codes, in contrast, rely on the precise timing of single spikes. The timing can be interpreted in relation to a fixed reference time, in intervals between the spikes, or by the order of arrival of different spikes. While a slight variation of a spike's timing would change nothing in a rate-based coding, a temporal coded information could be completely altered.

Rate and temporal codes can be further broken down into several distinct coding schemes, as shown in Figure 2.6. The classification of the individual schemes is often ambiguous. The distinctive definition chosen in this case is simple: if the exact timing or the order of spike arrivals is crucial for the transmitted information, a temporal code is given; otherwise a rate code is present. Often a third category, population codes, is introduced. However, this does only introduce further ambiguity since this category does not include any temporal characteristics. The distinctive property is, whether one or multiple neurons are needed to represent the desired code. Because this can be the case in both, temporal and rate codes, the category is omitted here.

In the neurobiological research area, there was a broad consensus for a long time that rate codes are mainly used in biological systems [147]. Further research, however, suggested that precise spike times are used to encode the perceptions of the sensory organs.

[1] Parts of the following review have been published in [50].

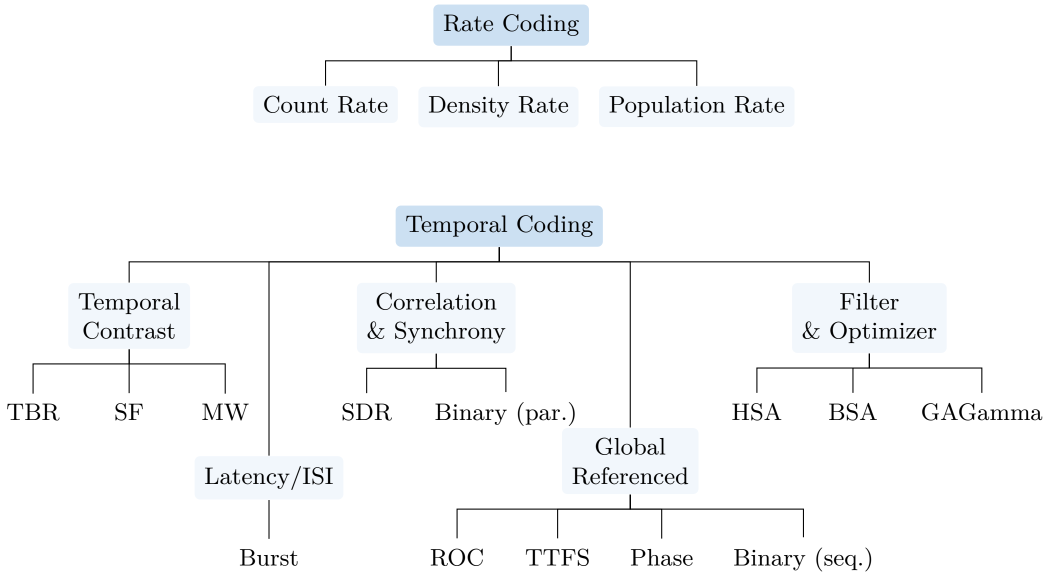


Figure 2.6: Taxonomy of rate and temporal coding techniques. Rate codes are characterized by the averaged spike activity over a temporal interval, population of neurons, or multiple executions of the same experiment. Temporal codes use the precise timing of spikes to encode information. (Figure previously published in [50])

The most well-known experiment which underpins this theory demonstrates, that the human visual system needs less than 150 ms to process new stimuli [36]. In the experiment, images were presented to the participants for 20 ms. The decision whether these images contained animals was already present after these 150 ms. Accordingly, a rate code which describes the image on the retina is highly improbable. The author of this experiment, Thorpe, was with that able to prove his earlier proposition about the relevance of precise spike times in biological systems [148]. Later, more and more publications support these findings in visual [38], [149]–[151], audio [152], tactile [37], and olfactory systems [153], [154]. The latter experiment targeting the olfactory system of mice showed an additional insight into biological coding: mice were able to discriminate between simple odors within a time of 200 ms; when the odors were similar, the discrimination took 100 ms longer. This suggests an integration of information over time [154].

Inspired by the biological examples, the assumption is evident that there exists a variety of coding schemes which we can adapt for our artificial applications. Though, depending on the application and the type of the input data, some coding schemes might be more suited than others. Networks which have to deal with fast changing inputs and rely on fast reactions will most probably not use rate-based coding schemes. For networks analyzing slowly varying data of high dimensionality, on the other hand, rate codes might come in handy. Unfortunately, there is no universal answer which coding scheme performs best yet.

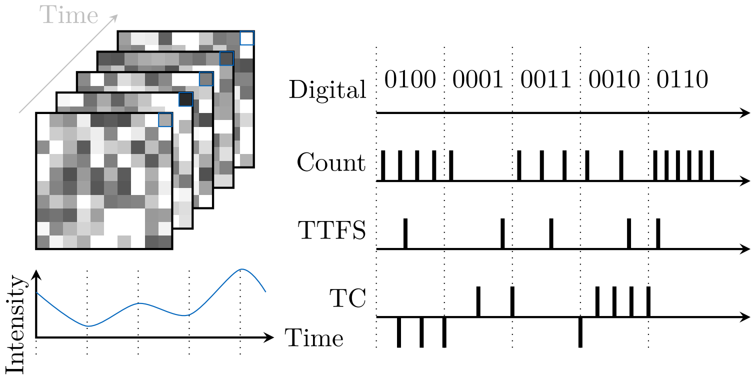


Figure 2.7: Exemplary coding schemes for a sequence of images over time. The intensity-time plot indicates the changes of the pixel value in the blue square as a continuous function. The dashed lines indicate the time instances at which the images have reached the color value. Digital, count, and TTFS spikes in correlation to the local minima and maxima in the intensity curve. The TC emits spikes if the continuous intensity change exceeds a certain threshold. (Figure previously published in [50])

Figure 2.7 illustrates the fact, that data can be encoded using different schemes. In the example, a sequence of images is encoded into spikes. In the digital domain, the brightness of each pixel in every frame is represented by a digital number. The rate-based count code translates the intensity in the number of spikes which are emitted per frame. A larger number of spikes can for example represent a higher brightness at the respective pixel. Time-to-First-Spike (TTFS) is a temporal code in which only one spike is emitted per pixel per frame. The exact timing of the spike thereby encodes the brightness information. Using the Temporal Contrast (TC) scheme, we encode the continuous change of light intensity over time instead of in fixed frames. In this case, positive or negative valued spikes are emitted as soon as a relative intensity change surpasses a threshold. Specialized cameras have been developed, which leverage this event-based type of encoding [155].

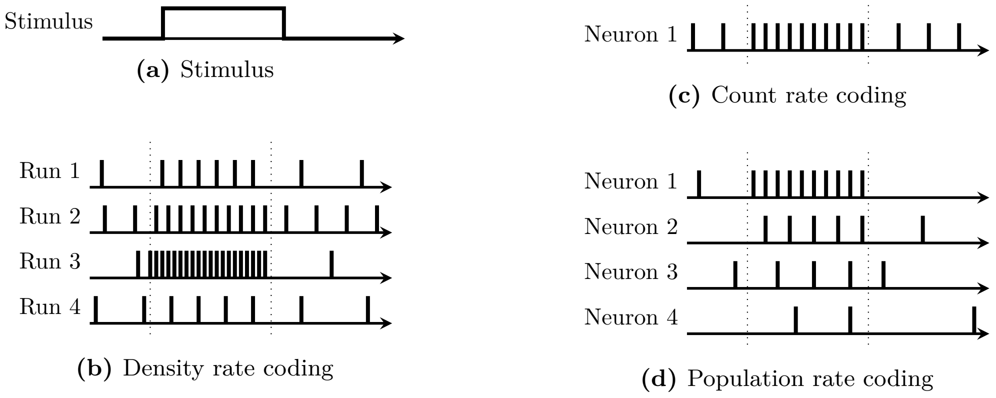
2.4.2 Rate Coding

Rate codes can be further divided into three subcategories: count rate codes, density rate codes, and population rate codes. Three schematic visualizations of the codes applied to an arbitrary input signal are depicted in Figure 2.8. The definitions are based on the work of Gerstner, Kistler, Naud, and Paninski on neuronal dynamics [63].

Count rate – average over time

Count rate codes average the spike activity of a single neuron N_{spikes} over a time interval Δt

$$r(t) = \frac{N_{\text{spikes}}}{\Delta t}. \tag{2.13}$$



(a) Stimulus

(c) Count rate coding

(b) Density rate coding

(d) Population rate coding

Figure 2.8: Visualization of rate coding techniques. The exemplary stimulus is a wide pulse (a). The dashed line in the encoding visualizations (b-d) indicates the rising and falling edge of the stimulus. (Figure previously published in [50])

This mean firing rate is often also referred to as frequency coding. This type of code can describe any value with a finite accuracy. For the representation of sequences this coding scheme is only viable for slowly varying values as many spikes are required to encode the instantaneous value of the signal.

The timing of the spikes in count rate codes can be exact or randomly distributed. In the former case, the spike times are evenly spaced, determined by the number of spikes in the given interval. The latter, randomly distributed, case is often modeled by a Poisson distribution. The error introduced by the encoding of continuous values by a discrete number of spikes decreases with the number of spikes involved in the coding $1/N_{\text{spikes}}$. With Poisson distributed spike times, however, this error is $1/\sqrt{N_{\text{spikes}}}$ due to the varying number of spikes in a given interval [156].

The activation value of Rectified Linear Unit (ReLU)-activated artificial neurons can be trivially encoded by count rate codes. Therefore, many SNNs based on conversion methods [92] use this kind of coding.

Biological evidence of this type of coding has for example been shown by Adrian and Zotterman. They observed different spike count rates when stretching a frog muscle with different forces [157].

Density rate – average over multiple runs

The density rate can be calculated by recording the neural activity of a neuron over multiple trials. The rate

$$p(t) = \frac{1}{\Delta t} \frac{N_{\text{spikes},K}(t; t + \Delta t)}{K} \tag{2.14}$$

can then be visualized in a Peri-Stimulus-Time Histogram (PSTH). Therefore, the number of spikes $N_{\text{spikes},K}$ is averaged over K runs of the same network. This coding scheme is by no means biologically plausible. Gerstner et al. illustrate this fact with an example: Imagine a frog, which attempts to catch a fly. It will certainly not base its movements on multiple computations over the exact same trajectory of the fly [63]. Yet, in artificial applications or when evaluating neuronal activity, this code can be beneficial.

Population rate – average over multiple neurons

In population rate codes, the activities of multiple neurons are averaged in a given time interval $[t; t + \Delta t]$:

$$A(t) = \frac{1}{\Delta t} \frac{N_{\text{spikes}}(t; t + \Delta t)}{N}. \tag{2.15}$$

Here, the number of spikes of N neurons is averaged.

Population rate codes are heavily used within the NEF [140]. There, the rich diversity of the spike responses of different neurons in the populations leads to unique and distinguishable spike patterns of the whole population.

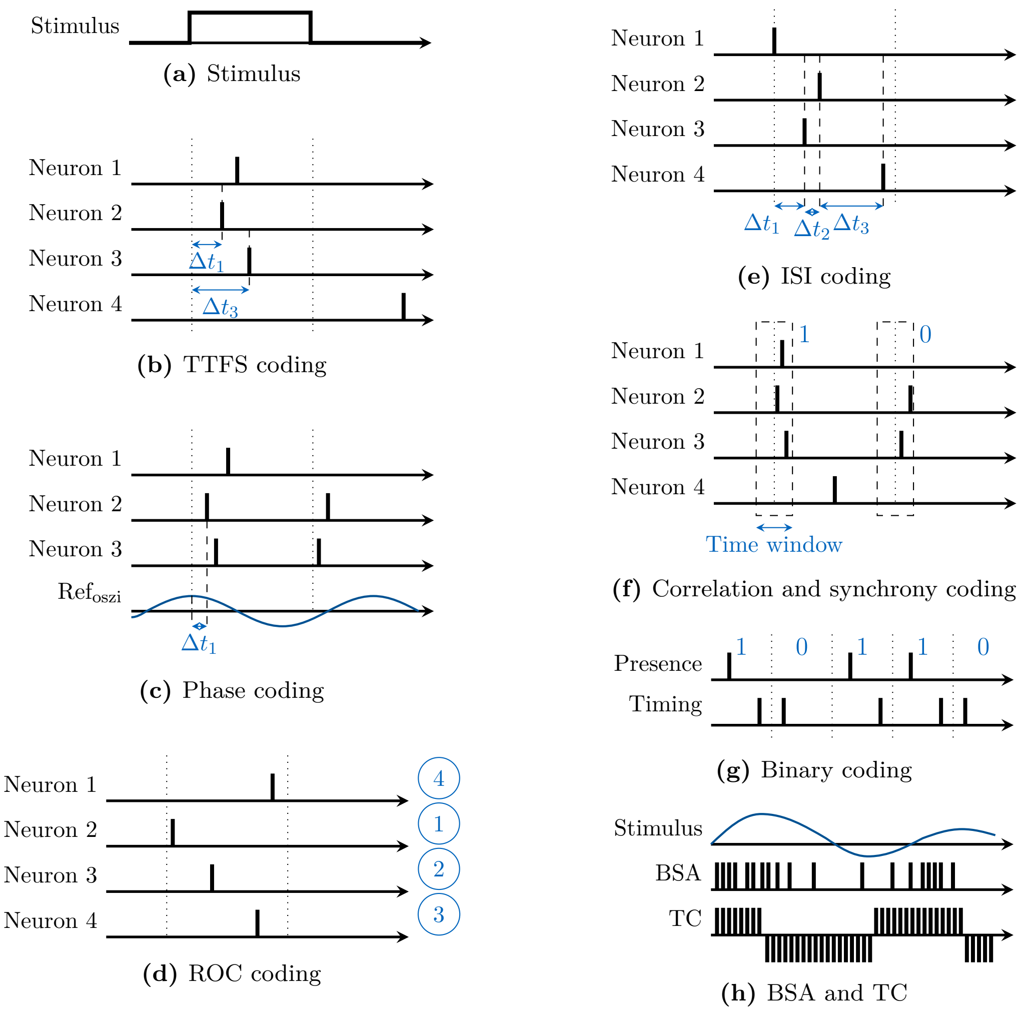
2.4.3 Temporal Coding

Temporal codes comprise all coding schemes, which rely on the precise timing of spikes or their distinct order of arrival. Thus, small variations already alter the encoded information. The schemes included in the temporal coding category are various: globally referenced codes encode information into relative intervals with respect to a fixed reference, periodical signal, or oscillation; TC codes represent the signal's derivative; in Interspike Interval (ISI) codes, the time interval between spikes is the main information carrier, while correlation codes focus on the simultaneous activity of neurons; and filter and optimizer-based approaches convolve the input signal with kernels to obtain the spike train representations. To illustrate this variety, Figure 2.9 shows exemplary temporal coded spike trains of an arbitrary input signal.

Global referenced codes

Globally referenced codes encode information into the relative interval between spikes and a stationary or periodic reference. The data to be encoded is therefore processed in packets between two successive reference points. Using the **TTFS** coding scheme, this relative interval is proportional to the represented signal amplitude. The interval between the spike and the reference point Δt can, for example, be the inverse of the amplitude a to be encoded $\Delta t = 1/a$, or a linear relation $\Delta t = 1 - a$. In these cases, a early spike time – meaning a short interval between reference and spike – corresponds to a high amplitude value. Naturally, this can of course also be defined inversely. TTFS is in some cases also named latency coding [38], though, this naming is less widely spread.

In **phase**-coded spike trains, the information is encoded with respect to the phase of a reference oscillation, rather than a single point in time [39], [40].



(a) Stimulus

(b) TTFS coding

(c) Phase coding

(d) ROC coding

(e) ISI coding

(f) Correlation and synchrony coding

(g) Binary coding

(h) BSA and TC

Figure 2.9: Visualization of temporal coding techniques. The wide pulse stimulus in (a) is used for the visualizations in (b-g). The dashed line indicates the rising and falling edge of the stimulus. Δt describes the latency between the reference point and the spike. In (d), the order of spikes is numbered on the right. The stimulus for the coding visualization in (h) is the sinusoidal wave, which is directly given in the same sub figure. (Figure previously published in [50])

Rank-Order Coding (ROC) uses the order of the spikes emitted by different neurons as the carrier of information [42], [43]. The exact spike times are thereby irrelevant as long as the spike order is maintained. The spikes are therefore often arranged in fixed time steps, resulting in equidistant discrete spike times. Because the absolute amplitude information gets lost, this scheme implements normalization properties. An exact reconstruction of the input signal is thus not possible.

The last coding scheme within the globally referenced codes are **sequential binary** codes. These codes directly implement the ones and zeros of a bit stream. For that, every bit is assigned to a time interval and the generated spikes are related to these intervals. Within the encoding scheme, two interpretations of the encoded bits are possible: (1) the ones and zeros can be encoded into the presence and absence of spikes within an interval [158], or (2) into the timing of the spikes within the interval [159]. In the second case, the interval is split into halves. Accordingly, a spike can either be present in the first or in the second half of the interval, which corresponds to the encoded bit value. Accordingly, the number of emitted spikes stays constant, independent of the encoded bit pattern.

In most globally referenced codes, early spikes represent important information or large signal amplitudes, respectively. The network can thus base its activity on the first spikes, neglecting the information provided by later, less important spikes. This introduces a speed-accuracy tradeoff, as the network is able to respond before the whole input pattern has been presented to its neurons [42].

Temporal contrast

Coding schemes belonging to the temporal contrast category produce spikes based on the temporal changes of the input signal [160]. The category can be broken down to three algorithms: **step-forward (SF)**, **moving-window (MW)**, and **threshold-based representation (TBR)** [161]. All three algorithms emit separate spikes for positive and negative changes of the signal amplitude. However, the rules defining the generation of the spikes are different.

In SF, the present signal value is compared to a moving baseline. If the difference exceeds a predefined threshold, a spike is emitted. The baseline is then updated to the current signal value and the next value is compared. The threshold is thus the only tunable parameter in this case. In MW, the calculation of the baseline is based on the signal's mean value within a moving window, which adds the size of the window as a further parameter.

TBR only takes the difference between two succeeding signal values into account. In each time step, the difference is compared to a threshold, which is based on the signal's mean value, standard deviation, and an adjustable factor. Accordingly, the characteristics of the whole sequence are used to encode the signal. This also means, that future signal values, which have not been presented to the network, influence the encoding.

Inter-spike interval coding

Similar to TTFS, **ISI** codes are also sometimes named latency codes. However, the latency is here defined as the relative interval between different spikes of a neuron group, not to a global reference [162].

Correlation and synchrony

The synchronous firing of neurons within a population is the primary information carrier of **correlation** and **synchrony** codes. Synchrony is thereby given if the ISI between the neuronal activity within the population is relatively short [63]. The decisive factor is which of the neurons is active at the same time. **Sparse Distributed Representations (SDRs)** [163], [164] also fall into this category. In SDRs a fixed subset of neurons within the population is active at any point in time. Using this scheme, a virtually infinite number of patterns can be represented by a population [165]. If only one single neuron is active at any time, the scheme is also called **amplitude** coding. If, for example, a continuous signal is encoded, each neuron within the population represents a certain amplitude range. If a neuron is active, the current signal amplitude falls in the respective range.

Parallel binary codes assign bit representations of larger words to specific neurons. The synchronous firing of a population of these neurons within a time interval can be interpreted as the ones and zeros of the binary represented word. Accordingly, the whole word is represented at once by the synchronous presence and absence of spikes. In contrast, sequential binary codes, as introduced earlier, encode bits into a stream of spikes.

Filter and optimizer-based approaches

To identify a system, a standard approach is to measure the system's response to known inputs and to derive a mathematical relation between these inputs and outputs. The characteristics of a neuron or a population of neurons can, for example, be determined by measuring the spike responses to arbitrary analog input signals. **Ben's Spiker Algorithm (BSA)** [166] and its predecessor **Hough Spiker Algorithm (HSA)** [167] use this approach to encode analog signals into spikes. There, the input signal is convoluted with an encoding filter and if the convolution exceeds a predefined threshold, a spike is generated.

In the **GaGamma** scheme, the encoding process is interpreted as a data compression task with prior knowledge [168]. The task is to maximize to transmitted information while minimizing the spike density. This optimization problem is supported by prior knowledge of the signal's characteristics.

3 Signal Encoding

In chapters 5 and 6, two different applications are examined in which we leverage the temporal characteristics of SNNs: keyword recognition of spoken words and radar-based hand gesture recognition. Both experiments comprise the classification of data sequences. However, the sequences have differing temporal and spatial characteristics and thus require the input data streams to be encoded differently. In this chapter, the encoding schemes used in the experiments are presented.[1] The implemented encoding schemes are separated in streaming and frame-based approaches and are evaluated in separate sections. In the last section of this chapter, we present an encoding scheme that uses resonating neurons to transform input signals with sinusoidal components into spatio-temporal spike trains. This scheme is part of the stream-based encoding schemes but it is introduced in a separate section.

3.1 Introduction

In general, there is no broad consensus about how to encode information to be processed by SNNs. While rate codings or TTFS are often used in image processing [78], [92], [93], BSA and TC are used for the processing of data streams [160], [169]. However, the decision for an encoding scheme is always dependent on prior knowledge about the data to be encoded.

The data sequences considered in this work consist of one-, two-, or three-dimensional features, which vary over time as shown in Figure 3.1:

- 1-dimensional vectors consisting of one scalar signal value per time step,
- 2-dimensional matrices consisting of a one-dimensional feature vector per time step, and
- 3-dimensional tensors consisting of a two-dimensional matrix per time step.

To decide on specific encoding schemes for the above-mentioned sequence types, the general characteristics need to be reflected. Although all three types mentioned above involve sequence processing, they may benefit from disparate encoding schemes. In the applications examined in this work, the 1-dimensional vectors contain fast-changing signals with high sampling rates. The latter two cases, in contrast, combine the information of several time steps into feature vectors and feature maps. Accordingly, they are updated far less often. During a fixed time interval, the 1-dimensional signal contains thus much more sample points than the more-dimensional cases. Therefore, it is reasonable to

[1] Some of the approaches have already been published in [51], [52], [55].

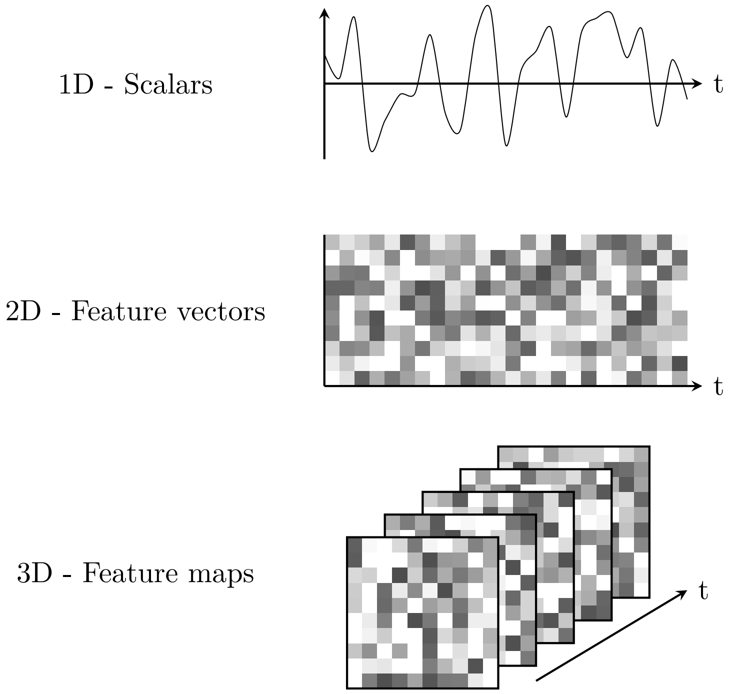


Figure 3.1: Considered data sequence formats. The input sequences in examined applications in this work consist of temporally varying scalars, feature vectors, and feature maps.

encode the first, 1-dimensional signal as a stream and the multidimensional sequences in a frame-based format in the present applications.

In the following, we, therefore, examine different temporal encoding approaches for stream and frame-based sequence processing. Rate-based encoding schemes are not considered in this work because they do not fully leverage the temporal potentials of SNNs. Although rate-based SNNs have shown to reach similar performances compared with ANNs at lower energy consumption levels [19]–[23], they use far more spikes to encode information than networks using temporal encoding schemes. A higher network activity results in higher dynamic power consumption. Davidson and Furber argue in their comparative study of ANNs and SNNs in terms of digital hardware properties: "The path towards truly mould-breaking neurally-inspired computation for artificial systems should focus on information encoding as the way forward" [23].

3.2 Frame-based Encoding

The frames in our applications consist of characteristic one- or two-dimensional feature maps. Although the frames are parts of sequences, they are converted without consideration of their predecessors or successors, but only based on the information provided by the frames themselves. The temporal changes are to be detected and evaluated by the connected SNN.

Due to the slow update frequency of the feature maps and the short length of the sequences, encoding schemes within the categories Temporal Contrast and Filter & Optimizer are not apt to be used in this case. These schemes are more suited for encoding high-frequency data streams with a high temporal variance. The remaining encoding schemes given in Figure 2.6 are examined in the following.

3.2.1 Correlation and Synchrony

Codes based on correlation and synchrony encode information in the concurrent spiking of groups of neurons. Therefore, the encoding of large structures like images requires even larger populations of neurons, depending on the accuracy of the brightness value to be encoded per pixel.

A sparse activation is given when only a few neurons are active within the population at any point in time [163]. The advantage of such sparse codes in SDRs is the high robustness against noise. If only a small subset of the population encodes the present feature, the chances for overlapping representations are negligible. Accordingly, the probability for a false activation in the preceding layer is low [165]. However, the populations need to be sufficiently large to leverage these advantages. With two active neurons within a population of 15 neurons, $\frac{15!}{(15-2)\,2!} = 105$ unique patterns (feature states) can be represented. But if the state of only one neuron is flipped, an entirely different value is encoded. With more active neurons, the size of the population also has to be increased to ensure the sparseness of the activations. This coding scheme ends up necessitating numerous neurons per feature, which contradicts the initial objective of finding an efficient encoding scheme.

3.2.2 Globally Referenced

The common reference point given by the presentation of a new frame renders globally referenced codes, an eligible encoding scheme for one- and multidimensional feature maps. The implementation of globally referenced encoding schemes expands a single frame to multiple time steps. In both considered schemes, TTFS and ROC, each neuron representing a single feature spikes at most once per frame. In both cases, the spikes representing the most relevant features are generated first, while spikes featuring less important information are sent out later or not at all. In general, the resolution of the encoded data improves the more time steps are at the disposal of the encoding process.

TTFS encodes each feature into the precise timing of one spike. The absolute value of the feature is thereby translated into an interval relative to the global reference point. This interval can, for example, be defined as the inverse of the amplitude or a linear relation, as introduced in Section 2.4.3. In a setting in which the network is executed in discrete time steps, the continuous intervals have to be rounded to the next time step. TTFS results in a fixed amplitude range of the input feature to be encoded if no normalizing preprocessing step is performed.

In ROC, the features of one frame are encoded in relation to each other. Accordingly, the amplitude range of the encoded features is flexible, but the absolute values are lost. In the vanilla case, exactly one feature representing neuron is active in each time step. For n features to be encoded, this results in $n!$ different combinations along the $t = n$ time steps. Depending on the application, the scheme can be modified so that multiple features spike simultaneously or the least significant features do not spike at all, resulting in a shorter encoding time frame.

3.2.3 Binarization

Binarization is a special case of frame-based encoding schemes, in which the encoding lasts only one time step – as opposed to globally referenced schemes – and each feature is represented by a single neuron – as opposed to correlation and synchrony coding. It thus combines the minimum specification for both temporal and spatial requirements. In each single time step, every neuron encoding a single feature can either be active or inactive. A distinction in time- or rate-based algorithms or further sub-categories is thus not practical. The binarized encoding of a frame or a sequence of frames cannot be clearly categorized regarding Figure 2.6.

The binarization encoding scheme is well suited for spatial data, which does not rely on accurate amplitude representations. An example of this kind of data is the MNIST dataset, which consists of grayscale images of handwritten digits [170]. In this simple dataset, the precise representation of each pixel intensity is not crucial for an accurate recognition; the simple notion of whether there is a stroke or not is sufficient.

The input data can also be a temporal sequence of frames. In this case, each frame is evaluated and binarized individually. An example for this input is given in the radar-based gesture recognition task, which is further elaborated in chapter 6. In that case, the input consists of a sequence of range-Doppler images, which show the distance-velocity map of the near surrounding. In this setting, the nearest objects will be represented with the highest amplitude values in the map, while the background will mostly be noise. As the absolute value of the entries does not carry relevant information, and we are only interested in the nearest objects, a binarization of the input is feasible.

The binarization requires a threshold $v_{\text{th,input}}$, which indicates the level at which every element x_i of the input $x \in \mathbb{R}^n$ is mapped to a one or zero:

$$z_i = \begin{cases} 1, & \text{if } x_i > v_{\text{th,input}} \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

The binarization threshold can thereby be fixed or adaptive. While a fixed threshold convinces with its simplicity, an adaptive threshold can better map to the spontaneous characteristics of the input signal or be used for a constant number of activations within the binarized input. Two practical threshold rules are the mean value or the α-quantile of the instantaneous input frame:

$$\text{mean:} \quad v_{\text{th,input}} = \text{mean}(v) \tag{3.2}$$

$$\alpha\text{-quantile:} \quad v_{\text{th,input}} = \alpha \cdot n\text{-largest value of } \boldsymbol{x}. \tag{3.3}$$

With mean binarization scheme, all inputs that are greater than the average value of the input vector are set active. All other entries are inactive. This leads to a normalized encoding of the input vector. The number of active entries is, thereby, not constant. A constant number of activations can be reached using the quantile-based binarization. Here, the $\alpha \cdot n$-largest values of $\boldsymbol{x}$ are set to be active, with n being the number of features within the input vector. The quantile-based binarization can additionally be extended that not only two states but three are used. In that case, the top quantile and the second quantile are used as thresholds.

Current Injection

A further special case is the current injection scheme, in which no conversion of the input into spikes takes place. The input signal is fed into the neuron as it is, resulting in a direct injection in the form of a current. The conversion into spikes is thus performed by the network itself in the first population of neurons. The current injection method can be used for both data frames and streams. We will use this approach for the stream-based encoding in Section 3.4.

3.3 Stream Encoding

The one-dimensional signals in our applications consist of temporally varying superpositions of sinusoidal components. To encode the signals, not only the instantaneous value of the signal is considered, but also its recent course. Because the stream is theoretically infinite, there is no single global reference point as in the frame-based encoding. As mentioned above, TC methods and filter and optimizer-based approaches are suited to encode these fast-changing signals.

3.3.1 Temporal Contrast

TC-based methods encode changes in the signal's amplitude into spike events (see Section 2.4.3). Polarized spikes emphasize the change above a defined threshold in the positive or the negative direction. To encode the signal's course precisely, the threshold is chosen to be small. In this way, also small changes in the signal are recognized. This causes numerous spike events if the signal's amplitude changes fast. A doubling of the signal frequency consequently results in a double number of spike events. This type of encoding is suited to encode arbitrary varying signals with high accuracy if the threshold is chosen appropriately and if many spikes are acceptable.

TC is the prevalent encoding scheme for event-based Dynamic Vision Sensor (DVS) cameras [155], [171]. There, local changes in light intensity at the specific pixels are encoded into spike events. The changes in light intensity are arbitrary; however, they occur in relatively long intervals. If there are no changes, pixels remain silent and do not emit any signals. As a result, these sensors emit rich but sparse spike sequences, which describe the temporal changes of the perceived scenery.

In our applications, the characteristics of the signals are known a priori: The signals consist of superimposed sinusoidal waves with high frequencies in the range of kHz to MHz. This knowledge should therefore be incorporated into the encoding scheme. A TC-based method can encode any signal but does not leverage this prior knowledge.

3.3.2 Filter and Optimizer

BSA is one of the most prominent examples of filter-based encoding schemes. There, the input signal is convoluted with a filter function, and a spike is generated if the resulting value surpasses a predefined threshold. With this approach, similar to TC, arbitrary signals can be encoded. It is, for example, used for the task of recognizing anomalies in Electroencephalography (EEG) data streams [169].

We extend this idea and simultaneously use multiple filters to evaluate the signal based on different characteristics. With the prior knowledge of the signal's sinusoidal components, the filters can be chosen accordingly. The resulting encoding scheme uses filters to detect different spectral components in the signal using resonating neurons. The encoding is extended by an additional spatial domain by using multiple filters. Whereas TC and BSA only use one or two spike sources to indicate a (polarized) signal change, we encode the presence of different independent features (spectral components) into multiple sources. The scheme will be introduced and evaluated in the following section.

3.4 Frequency-selective Resonating Neurons

The RF neurons introduced in Section 2.1.3 can detect spike sequences with specific frequencies or ISIs [64]. We extend this approach and use the resonating neurons to directly convert continuous analog signals into frequency-selective spike sequences.[1] The encoding into spikes and the spectral analysis are thereby performed simultaneously. Additionally, as the analysis is performed directly on the analog signal in the time domain, no analog-to-digital conversion or buffering of samples is needed. Common approaches perform a Fast Fourier Transform (FFT) and subsequently transform the result into spikes to be analyzed by SNNs [35], [172], [173]. A practical implementation using RF neurons as frequency selective input encoders within an SNN is shown in and will be demonstrated in chapter 5. There, the RF neurons are used to convert speech signals into spike trains to be recognized by the SNN.

With reference to the taxonomy of encoding schemes presented in Figure 2.6, RF neurons can be categorized as a filter-based approach.

[1] This approach has been previously published in [51].

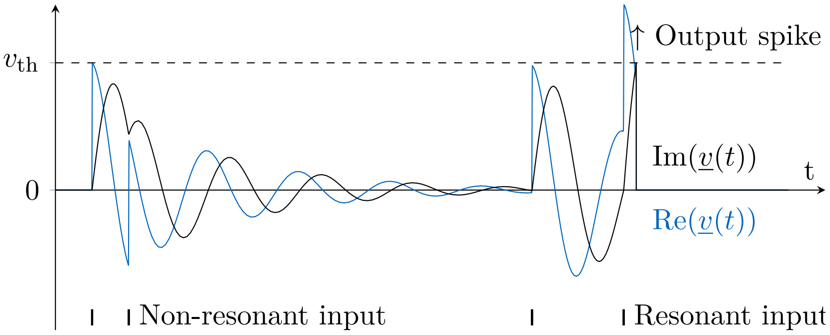


Figure 3.2: Pulsed excitation of a RF neuron. The input current in applied to the current-like variable $\text{Re}(\underline{v}(t))$, whereas the voltage-like variable $\text{Im}(\underline{v}(t))$ is used to determine the excitation of the neuron. An output spike is generated as soon as $\text{Im}(\underline{v}(t))$ crosses the threshold voltage. The excitation only reaches the required level when the input spikes arrive in the correct temporal distance.

3.4.1 Properties of the Resonator

As introduced in Section 2.1.3, the complex-valued differential equation describing the membrane $\underline{v} \in \mathbb{C}$ is given by

$$\frac{\mathrm{d}\underline{v}}{\mathrm{d}t} = (-\lambda + \mathrm{i}\omega_n)\,\underline{v} + i(t), \tag{3.4}$$

with λ being the damping constant, i the imaginary unit, ω_n the neuron's natural frequency, and $i(t)$ the analog input signal. In the following, we distinguish between four different input scenarios: (1) pulsed excitation, in which $i(t)$ consists of a spike train, (2) resonant excitation, in which $i(t) = a\cos(\omega t)$ consists of a sinusoidal signal with the frequency $\omega = \omega_n$ and the arbitrary amplitude a, and (3) non-resonant excitation with the same input as in (2) but with $\omega \neq \omega_n$. In a fourth scenario (4), an arbitrary signal is shown in which scenarios (2) and (3) superimpose.

Pulsed Excitation

Spike trains can be the direct signal input to the resonator or originate from the communication between neurons within a larger network. The resonator's response to this pulsed excitation has been shown by Izhikevich [64]. He showed that in contrast to the LIF neuron, the timing of the incoming spikes is of major importance. If two succeeding input spikes arrive with an interval of half of the period of the resonator's oscillation, the two spikes cancel each other out. If the interval is near a full period, they add up. With that, if enough spikes arrive at the correct intervals, the neuron reaches its firing threshold.

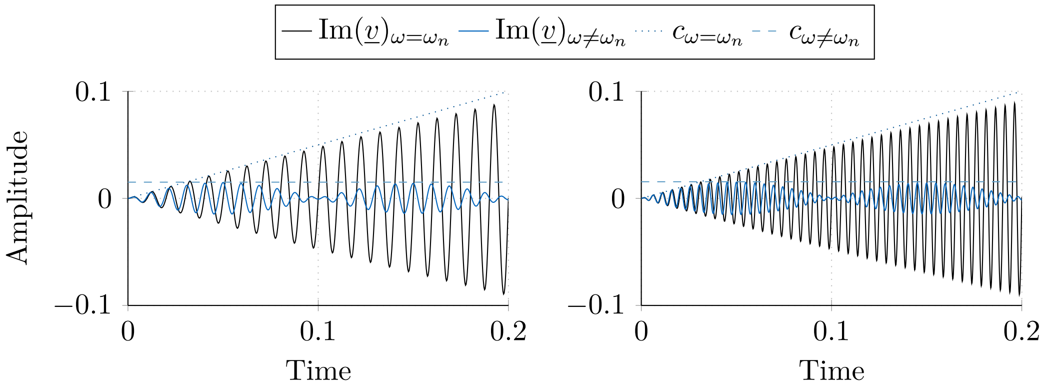


Figure 3.3: Resonant versus non-resonant excitation of the RF neuron. The temporal course of the voltage variable $\mathrm{Im}(\underline{v})$ is shown for an exemplary resonant and non-resonant excitation. The resonant frequency of the neuron is 100 Hz (left) and 200 Hz (right). The non-resonant excitation has a frequency of 10 Hz above the resonance. Additionally, the upper boundaries of the respective envelopes are given. It can be seen, that the boundaries are independent of the frequencies themselves.

Resonant Excitation

For a resonant excitation of the resonator, the input current is modeled as a single sinusoidal oscillation with the same frequency as the resonator's resonant frequency: $i(t) = a\cos(\omega_n t)$. The steady state response of the oscillator's membrane voltage results in a constant oscillation:

$$\begin{aligned} \underline{v} &= C\, e^{t\,(-\lambda+\omega_n\,\mathrm{i})} + \frac{e^{t\omega_n \mathrm{i}}}{2\lambda} + \frac{e^{-t\omega_n \mathrm{i}}}{2\lambda - 4\omega_n \mathrm{i}} \\ &\approx \frac{e^{t\omega_n \mathrm{i}}}{2\lambda} \quad \text{for } t \to \infty. \end{aligned} \tag{3.5}$$

The transient response, however, follows an exponential process. The envelope of the oscillation is given by

$$\begin{aligned} c_{\omega=\omega_n}(t) &= \frac{a}{2d}\left(1 - e^{-t\lambda}\right) \\ &\approx \frac{a}{2}t \quad \text{for } t\lambda \ll 1, \end{aligned} \tag{3.6}$$

which can be approximated linearly during the beginning of the oscillation when the product of t and λ is sufficiently small.

Non-resonant Excitation

The response to a non-resonant signal with $\omega \neq \omega_n$ reaches much lower amplitudes during the same time interval. In this case, the envelope follows a sinusoidal path that is given by

$$
\begin{aligned}
c_{\omega \neq \omega_n}(t) &\approx \frac{a\,\omega_n}{|\omega_n^2 - \omega^2|} \left(1 + e^{-t\lambda}\right) \sin\left(\frac{\omega_n - \omega}{2} t\right) \\
\max_t (c_{\omega \neq \omega_n}(t)) &\approx \frac{2a\,\omega_n}{|\omega_n^2 - \omega^2|} \quad \text{for } t\lambda \ll 1.
\end{aligned}
\tag{3.7}
$$

The amplitude difference of the oscillation's envelopes of the resonant and the non-resonant excitations is approximately independent of the absolute frequency itself. At a constant measurement period T, the amplitude difference between the resonant frequency ω_n and the non-resonant frequency $\omega = \omega_n + \Delta\omega$ remains approximately constant for all ω_n. However, if the measurement period is too small, the differences between resonant and non-resonant excitation might not develop sufficiently to discriminate between close frequencies.

Arbitrary Excitation

Considering the sum of sinusoidal signals of different frequencies, the effects mentioned previously superimpose. The neurons respond strongly if oscillatory components of their respective natural frequency are present in the signal. The decomposition of an exemplary signal is exhibited in Figure 3.4. The neuron is excited with a signal of the form $i(t) = \sum a_i \cos(\omega_i t + \phi_i)$. In each simulation, the resonant frequency of the neuron changes by 1 Hz, producing 50 simulations. At various frequencies, the superposition of the influences by the constant, resonant, and non-resonant excitation is evident. The signal component with a frequency of 10.5 Hz and amplitude 1 lacks a distinct voltage maximum because it does not match any natural frequency of the neuron. However, the neurons with natural frequencies of 10 and 11 Hz exhibit a high membrane voltage due to the near-resonant excitation. The signal component with a frequency of 43 Hz displays the same amplitude but produces a higher membrane voltage at the corresponding neuron as it coincides with a natural frequency.

3.4.2 Spike Generation

The RF neuron emits a spike as soon as its excitation exceeds a defined level. The voltage variable $\mathrm{Im}(\underline{v})$ is used to determine the neuron's excitation. Accordingly, a spike is generated as soon as the threshold is reached:

$$
\delta(t) = \begin{cases} 1, & \text{if } \mathrm{Im}(\underline{v}(t)) > v_{\text{th}} \rightarrow \underline{v} = 0 \\ 0, & \text{otherwise.} \end{cases}
\tag{3.8}
$$

The membrane is subsequently reset to zero. Due to the resonance behavior of the membrane, the threshold is reached quickly if the resonance frequency is present in the signal. During a non-resonant excitation, the threshold is reached significantly later or not at all.

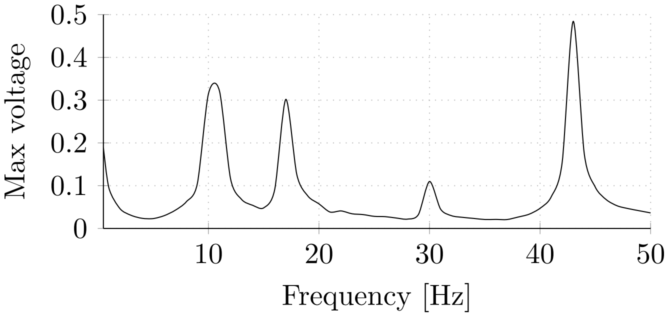


Figure 3.4: Excitation of RF neurons with superimposed sinusoidal signals. Maximum voltage of 50 neurons with $f_n = \omega_n/2\pi = 1, 2, 3, ...50$ Hz after a measurement time of one second. Excitation with sum of sine with frequencies 0 (constant excitation), 10.5, 17, 30, and 43 Hz and amplitudes 0.3, 1, 0.6, 0.2, and 1. The influences of constant excitation ($\omega = 0$), resonance ($\omega = \omega_n$) and non-resonant ($\omega \neq \omega_n$) excitation are discernible.

The parametrization of the threshold introduces the tradeoff between the processable amplitude range of the input signal and the achievable frequency resolution. A low threshold enables the detection of resonant signal components with low amplitudes. Non-resonant components with large amplitudes, however, will also be able to reach the required excitation level. A high threshold, on the other hand, results in better differentiability between neighboring frequencies, but signals with low amplitude will be almost irrelevant. The upper and lower boundaries of the threshold can be determined using the simple relations

$$\begin{aligned} v_{\text{th}} &> \frac{a_{\max} T}{2N} \quad \text{and} \\ v_{\text{th}} &< \frac{a_{\min} T}{2}. \end{aligned} \tag{3.9}$$

The lower boundary of the threshold is given by the maximum number of spikes N, which is generated during the measurement time T in the presence of the resonant signal's largest amplitude $a_{\max}$. Every threshold value higher than this will generate fewer spikes at the largest amplitude level. The upper boundary restricts the threshold that at least one spike is generated during the measurement time T at the presence of a signal with the resonant frequency and the smallest detectable amplitude $a_{\min}$. Every threshold value smaller than this will enable even smaller amplitudes to excite the neuron enough to generate at least one spike.

The tradeoff between the minimum detectable signal amplitude, spike rate saturation at too large amplitudes, and the discriminability of resonant and non-resonant frequencies can be somewhat accommodated by introducing a variable threshold value. After each spike generation, the threshold is increased to discriminate neighboring frequencies with high amplitudes more precisely. The threshold can be varied linearly or exponentially, depending on the prior knowledge of the characteristics of the input signal. If no spike is generated for an extended period of time, the threshold decreases steadily towards its initial value to enable the detection of signal components with lower amplitudes again. In accordance with the dynamics of the membranes, the temporal behavior of the threshold value is therefore described by a differential equation [52].

$$\frac{\mathrm{d}v_{\mathrm{th}}}{\mathrm{d}t} = \lambda\,(v_{\mathrm{th},0} - v_{\mathrm{th}}). \tag{3.10}$$

The threshold value thus always converges back to the initial, low, threshold $v_{\mathrm{th},0}$. Without limiting the generality, the damping coefficient is chosen to match the damping of the resonator itself, thus, coupling the two mechanisms to the same temporal scope. An exemplary evaluation of an arbitrary signal using this variable threshold is shown in Figure 3.6. The complete system of equations describing the neuron from eqs. (3.4) and (3.8) extends to

$$\begin{aligned}
\frac{\mathrm{d}\underline{v}}{\mathrm{d}t} &= (-\lambda + \mathrm{i}\omega_n)\,\underline{v} + i(t)\\
\frac{\mathrm{d}v_{\mathrm{th}}}{\mathrm{d}t} &= \lambda\,(v_{\mathrm{th},0} - v_{\mathrm{th}})\\
\delta(t) &= \begin{cases} 1, & \text{if } \mathrm{Im}(\underline{v}(t)) > v_{\mathrm{th}} \to \underline{v} = 0, v_{\mathrm{th}} = f\,(v_{\mathrm{th}}, v_{\mathrm{th},0})\\ 0, & \text{otherwise.}\end{cases}
\end{aligned} \tag{3.11}$$

The bandwidth of frequencies for which a neuron emits spikes depends on the properties of the resonator and the spike generation mechanism. The resonator is characterized by its resonant frequency and damping coefficient. The spike generation mechanism, on the other hand, is adjusted by tuning the initial firing threshold and the threshold adaptation function after each spike emission. To keep the number of spikes at the resonance frequency constant for different bandwidths, the initial threshold and the threshold adaption function have to be adapted simultaneously. A lower firing threshold enables a broader bandwidth and has to be compensated with a larger increase of the threshold after each spike emission.

In his publication, Izhikevich proposed the reset of only the voltage-like variable when an output spike has been emitted $\mathrm{Re}(\underline{v})$ [64]. This enables the RF neuron to produce spikes in the same frequency as the resonant input spike train because the excitation remains present. If both membrane variables were reset, an additional input spike would be required to initiate the neuron's oscillation. Thus, every other resonant input spike would trigger the emission of a spike. For our applications, we do not need the property of the continuous spike emission because the inputs to be analyzed are continuous sinusoidal signals. Instead, resetting both variables leads to a slightly better differentiation between resonant and non-resonant frequencies. For direct comparison, the same setup as in Figure 3.6 but with Izhikevich's reset scheme is shown in Figure 3.7. A noticeable difference is the delayed spike emission of the resonantly excited neuron with the index 100, shown in (e). A detailed view of the spike patterns shown in both figures in (d) around the neurons with the index 100, respectively, is given in Figure 3.5. The spikes that appear after the initial spike of each neuron are shifted due to the different reset schemes. Generally, the neurons following Izhikevich's reset scheme emit spikes slightly earlier. Additionally, the spike emissions of neurons with resonant frequencies close to the signal's sinusoidal components are less symmetrically centered around the resonant frequency. With that, there are less difference between the spike patterns of resonant and near-resonant excitations.

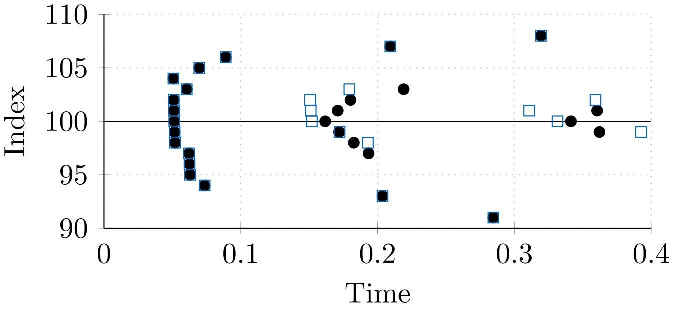


Figure 3.5: Detailed view of the RF neurons' spike times implementing different reset schemes. The spike times of the neurons implementing the full reset of both variables are depicted as black points. The spikes emitted by neurons with Izhikevich's reset scheme are depicted as blue squares. The neurons that follow Izhikevich's reset scheme emit spikes slightly earlier. Additionally, the spike response is less symmetrically centered around the frequency of the signal's component.

3.4.3 Frequency, Amplitude, and Phase Resolution

The exact timing of a generated spike contains evidence about superimposing information: (1) The presence of a specific frequency component within the signal is signalized, which matches the resonant frequency of the resonating neuron. The temporal occurrence of the frequency component can thus be localized, similarly to the Short-time Fourier transform or the wavelet transform. (2) The amplitude of the frequency component is encoded because a higher excitation results in a shorter interval to reach the firing threshold. The interval between consecutive spikes is directly influenced by the temporal development of the frequency component's amplitude and the neuron's threshold adaption. (3) The phase of the signal leads to small temporal shifts of the spike time because the neuron's membrane is more likely to reach the threshold voltage during the maxima of the oscillatory signals.

The frequency resolution of the encoding scheme is given by the sharpness of the pass band of the resonators and the number of resonators that are being used. As shown above, the resolution can additionally be influenced by the choice of the threshold voltage and its temporal variation. Similar to the Discrete Fourier Transform (DFT), the frequency resolution grows the longer the measurement interval is. As shown in eq. (3.5), the neuron's excitation is theoretically unbounded when excited with its resonant frequency, whereas in the non-resonant case, the maximum excitation is reached shortly after the onset of the signal. Thus, with longer measurement intervals, the differences get larger.

The amplitudes of the signal's frequency components are encoded in the timing of the spikes. With that, the resolution is, in theory, arbitrarily high. With the exact timing between spikes and the known temporal adaption of the spike threshold, infinitesimal changes can be expressed. However, in practical realizations, the smallest resolvable time interval will be determined by the actual hardware implementation. Additionally, the excitation of each resonator is also influenced by non-resonant signal components, akin to the effect of spectral leakage in the DFT.

The influence of the signal phase ϕ on the spike timing is comparatively small. The phase of the signal varies the maxima of the neuron's membrane oscillation by a fraction of the signal's frequency. Because the firing threshold of the neuron is most probably reached during these maxima, the timing of the spike is shifted by $\Delta t = \frac{\phi}{\omega_n}$. This temporal shift is small compared to variations introduced by tiny changes in the signal amplitude.

The encoding of sinusoidal signals using RF neurons leads to unique spatio-temporal spike trains, which can be analyzed by succeeding network populations. The interaction of input signal, membrane characteristics, and threshold adaption embed the signal information in sparse, frequency-selective sequences of events. With the characteristics of the resonators, the encoding can be used to represent the amplitude spectrum of the analyzed signal.

3.4.4 Discretized Model

The continuous differential equation model in eq. (3.11) can be transformed into a discrete representation. This transformation is performed by exact integration under the assumption that the input $i(t)$ is constant during the discrete interval n with the length Δt [174]:

$$\begin{aligned} \underline{v}[n] &= \mathrm{e}^{-\Delta t\,\lambda}\,\mathrm{e}^{\mathrm{i}\,\Delta t \omega}\,\underline{v}[n-1] + i[n] \\ &= \underline{\alpha}_{\omega}\,\underline{v}[n-1] + i[n]. \end{aligned} \tag{3.12}$$

With that, the neuron's complex membrane variable $\underline{v}$ is scaled and rotated in the complex plane. The constant resonant frequency-dependent factor $\underline{\alpha}_{\omega} \in \mathbb{C}$, thereby, defines this rotation.

In its discrete form, the computational complexity of the RF neuron can be compared to standard algorithms like the DFT or FFT, see table 3.1. The DFT has complexity $O(N^2)$ when transforming a sequence of length N. This complexity reduces to $O(MN)$ when only $M < N$ spectral components are required to be determined. Algorithms like the Goertzel Algorithm have the same complexity but require less computations than the vanilla DFT. With the FFT, the whole spectrum is calculated, but with a complexity of $O(N \log_2 N)$. The spectral transform using RF neurons also scales with $O(MN)$ as M neurons have to be updated during N time steps. For each of the M neurons, N complex multiplications and one addition are required. Additionally, the membrane voltage has

Table 3.1: Complexity of Fourier transform algorithms. The number of multiplications and additions is given per spectral component for a real-valued input signal. M refers to the number of desired spectral components, whereas N refers to the number of input samples.

Algorithm	**Complexity**	**Multiplications***	**Additions***	**Extra***
DFT	$O(MN)$	$2N$	$2(N-1)$	
Goertzel	$O(MN)$	$N+2$	$2N+2$	
FFT	$O(N\log_2 N)$	$2\log_2 N$	$3\log_2 N$	
RF neuron	$O(MN)$	$4N$	$3N$	N comparisons

*Per spectral component.

to be compared to the threshold value to generate an output spike if necessary. If the threshold is continuously adapted, additional computations are required.

As shown in table 3.1, the discrete implementation of the RF neuron is only efficient to use if only few spectral components need to be determined. The calculation of the spectral components is solved more efficiently using the Goertzel Algorithm. However, the RF neurons additionally perform the transform into temporal spike trains as introduced earlier. The properties of the resonating neuron additionally offer the possibility to perform the spectral analysis in analog circuits as shown in [71].

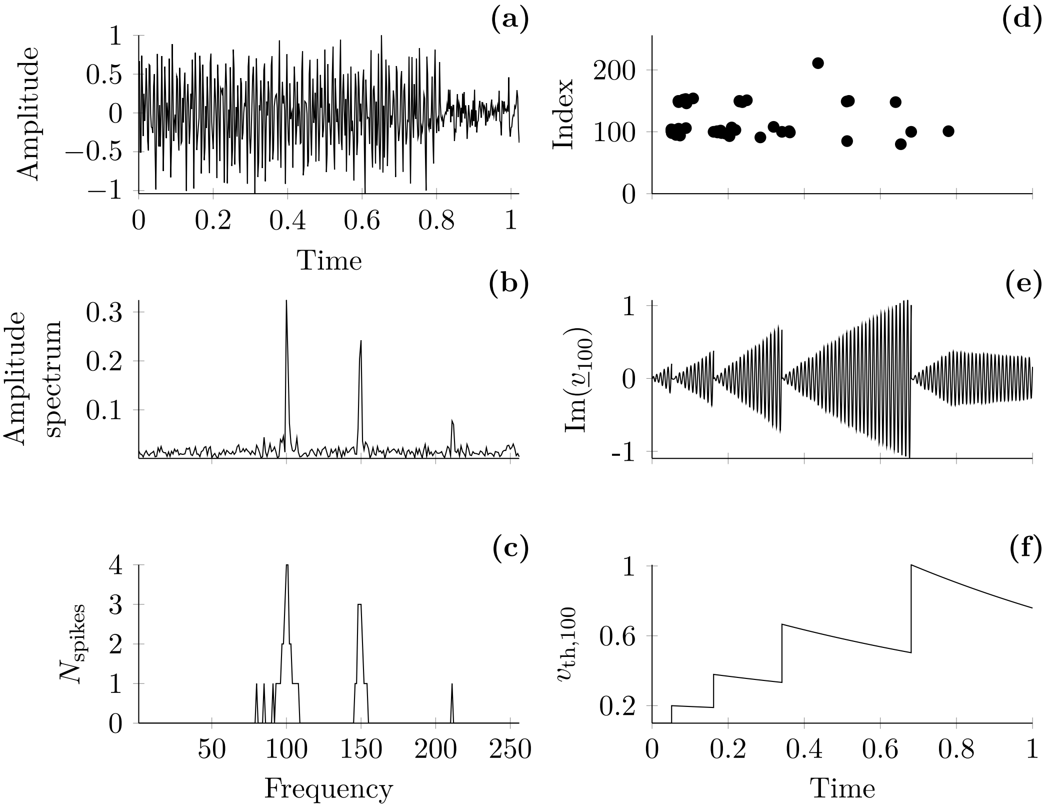


Figure 3.6: Evaluation of an arbitrary signal using RF neurons with spike generation and threshold adaption. The input signal (a) contains three distinct frequency components and a noise floor. (b) shows the real spectrum of the signal, which is obtained using an FFT. For the direct comparison, a bank of RF neurons is used, which have resonant frequencies equivalent to the grid points of the FFT. The number of emitted spikes per neuron is shown in (c). The temporal occurance of the spikes is depicted in (d). (e) and (f) show the temporal course of voltage-like variable and the threshold adaption of the neuron corresponding to the frequency of 100 Hz. The variant using Izhikevich's reset of only the current-like variable is shown in Figure 3.7.

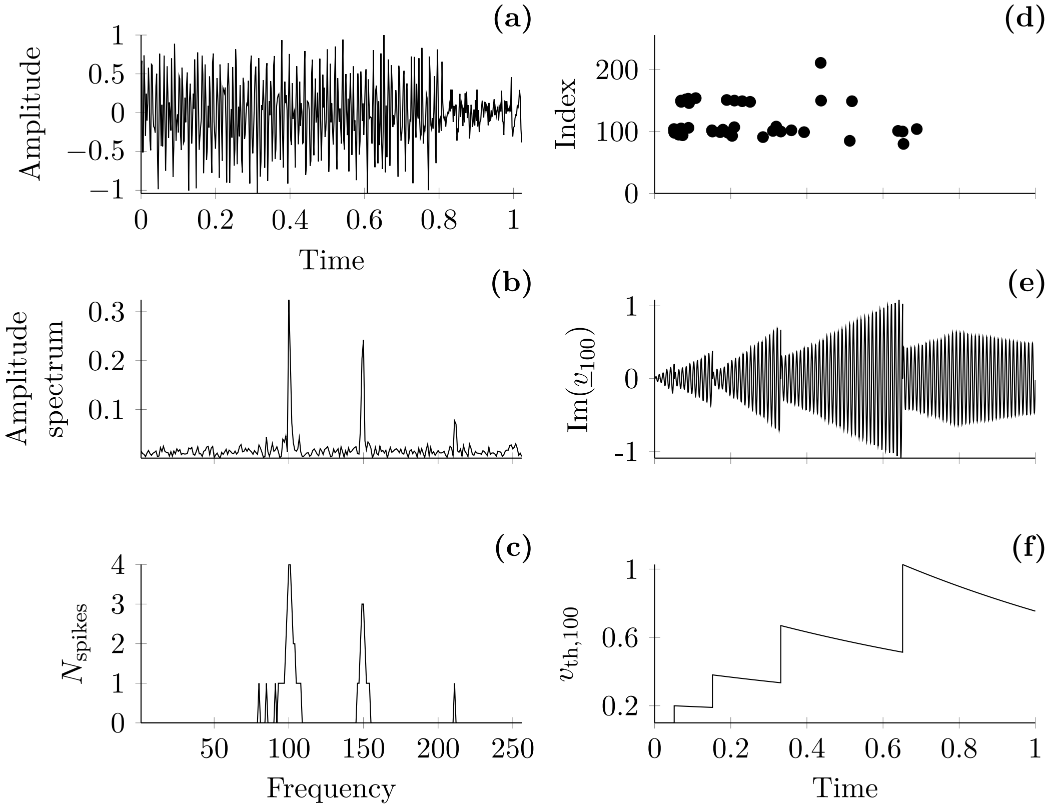


Figure 3.7: Evaluation of an arbitrary signal using RF neurons with Izhikevich's reset scheme. The setup is the same as in Figure 3.6. In this case, only the current-like variable $\mathrm{Re}(\underline{v})$ is reset after a spike emission. The exemplary neuron in (e), therefore, shows a different voltage course. The number of emitted spikes stays the same (c). The timing of the spikes shown in (d), however, differs.

4 Network Architecture and Training

In the previous chapter, we introduced the methods which we will use to encode the temporal input sequences into spike representations. The approaches used to train the SNNs to detect and classify the encoded information are introduced in this chapter. Therefore, existing approaches from the training of ANNs and SNNs are extended to improve the performance of the SNNs.[1] Additionally, the complexity of SNN implementations is analyzed, and the mapping to neuromorphic hardware is examined.

4.1 Neuron Models and Connectivity Schemes

The network architectures and the computation flows of the SNNs used in this work do not differ much from that of classical ANNs. The SNNs are batch-processed in fixed time steps, omitting the inherent benefits of sparse and event-based processing. However, this is only necessary during training to enable the credit assignment and the backpropagation of the error signal. Similar to ANNs, the spike-based networks consist of input, hidden, and output layers. The term population is often used synonymously with layer to underline the possible recurrent connectivity within the layer and to support the biological motivation.

In terms of ANNs, a spiking network is a recurrent network consisting of neurons with at least one stateful variable and a binary activation function. Note that the neurons do not necessarily have to be connected recurrently; the recurrence is already given by the neuron's state variables.

4.1.1 Discrete LIF Neuron

Our network model is based on the assumption that the main information processing takes place in the network dynamics rather than in the dynamics of single neurons. Therefore, the LIF neuron, as introduced in Section 2.1, is used as the computational element within the networks. It is one of the simplest models of spiking neurons. Additionally, spikes are modeled as binary events, which increase or decrease the neuron's membrane potential instantaneously. To increase the distinction to spike events in the continuous time domain $\delta(t - t^f)$, we describe spike events in discrete time as $z[t]$, with $z[t] = 1$ representing a spike and $z[t] = 0$ representing no spike emission.

In most parts of this work, the SNNs are simulated in discrete time steps to enable the use of BPTT. The LIF neuron model has thus to be discretized. The update of the membrane voltage $\boldsymbol{v}[t]$ at time t is performed for the whole population of n neurons at

[1] Some of the approaches have already been published in [52], [55]

the same time. The membrane voltage

$$\begin{aligned} \boldsymbol{v}[t] &= \lambda\, \boldsymbol{v}[t-1] + \boldsymbol{i}_{\text{charge}}[t] - \boldsymbol{i}_{\text{reset}}[t] \\ \boldsymbol{i}_{\text{charge}}[t] &= \boldsymbol{W}^{(l-1,l)}\, \boldsymbol{x}[t] \\ \boldsymbol{i}_{\text{reset}}[t] &= \gamma[t]\, \boldsymbol{z}[t-1] \end{aligned} \tag{4.1}$$

now contains the summation of the scaled membrane voltage of the previous time step and the charge and reset currents. The physical relation in eq. (2.5) between membrane voltage, charge current, membrane capacitance and discharge resistor is only preserved in an abstracted form without units. The membrane resistance and capacitance, and thus the time constant of the neuron, are represented by the factor λ. It controls the retention time of the membrane voltage. With no charge current, the membrane voltage decays exponentially towards the resting potential zero. The charge current $\boldsymbol{i}_{\text{charge}}[t]$ contains the weighted superposition of the input $\boldsymbol{x}[t]$. The matrix $\boldsymbol{W}^{(l-1,l)}$ contains the weights of the feedforward connections between the input and the neurons of the current population. Any zero in the matrix means that there is no connection between the corresponding neurons. The superscript l refers to the current layer, whereas $l-1$ is the preceding layer. The reset current $\boldsymbol{i}_{\text{reset}}[t]$ is a vector with non-zero entries at the respective neurons, which spiked in the previous time step. The values for the factor γ can vary based on the reset scheme, which is implemented (see Section 4.1.2). The output spike vector is calculated by comparing the current membrane voltages of each neuron j with the threshold voltage:

$$z_j[t] = \begin{cases} 1, & \text{if} \quad v_j[t] > v_{\text{th}} \\ 0, & \text{otherwise.} \end{cases} \tag{4.2}$$

The exemplary update of a LIF neuron is also shown in Figure 4.1: The layer's input $\boldsymbol{x}[t]$ at time step t is multiplied with the layer's weight matrix W to calculate the instantaneous charge currents $\boldsymbol{i}_{\text{charge}}[t]$ for the neurons within the layer. The stateful membrane voltages $\boldsymbol{v}[t]$ are updated accordingly. Subsequently, the nonlinear spike activation function is applied to generate the output spikes. The output vector $\boldsymbol{z}[t]$ is additionally used to determine the reset current in the next time step to reset the neurons that produced an output spike.

4.1.2 Reset Functionality

After each spike emission of a spiking neuron, the membrane voltage is reset. This can be done in two different ways: resetting the membrane voltage to a constant value (for example zero) or resetting the voltage by subtracting the threshold voltage [92]. The factor γ in eq. (4.1), which describes the absolute value of the reset current, is set

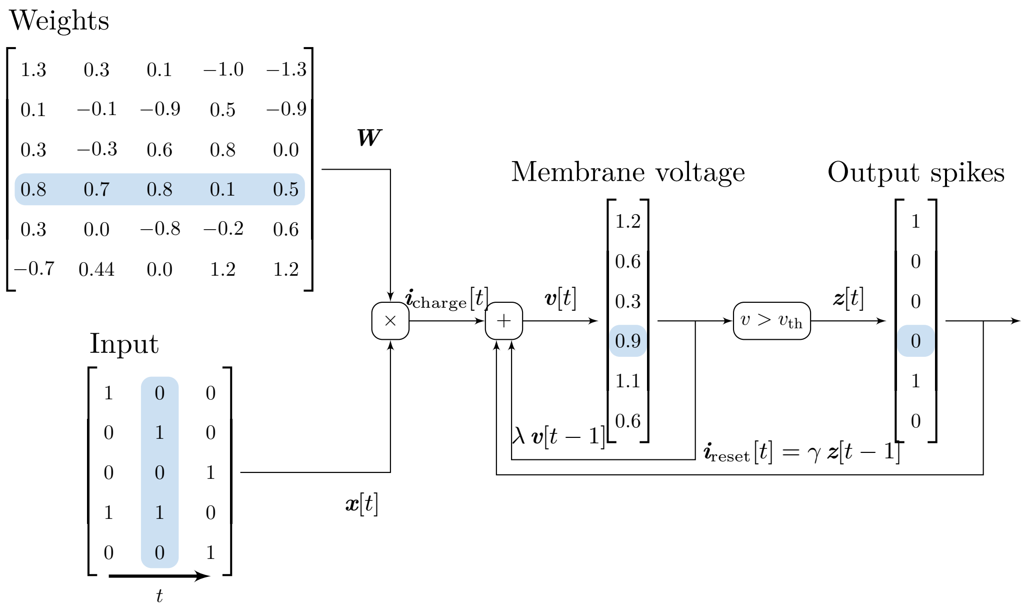


Figure 4.1: Update calculations of a population of discrete LIF neurons. The elements involved in the update of an arbitrary neuron are highlighted in blue.

accordingly:

$$\text{reset to constant/zero}: \quad \gamma_j[t] = v_j[t-1] - v_{\text{offset}} \tag{4.3}$$

$$\text{reset by threshold}: \quad \gamma_j[t] = v_{\text{th}}. \tag{4.4}$$

The reset to zero is modeled by subtracting the value of the last time step's membrane voltage for the individual neuron. This erases the neuron's memory of the past until the reset. The neuron thus discards the information about potentially very high activations. Additionally, the constant value v_{offset} can be included in the reset current to reset the membrane to a constant non-zero value. A positive value of the initial membrane voltage leads to an excited neuron after the reset. In contrast, a negative value can introduce a characteristic similar to the refractory period of a biological neuron.

By resetting the membrane voltage by subtracting the threshold voltage, each reset draws the same amount of charge from the membrane. Accordingly, if the neuron was excited to a level much higher than the threshold voltage, the membrane remains charged, thus preserving the membrane's memory. Therefore, the neuron is more likely to spike soon if the input still matches the neuron's preferred patterns. Especially when running discrete simulations with coarse time resolution, multiple input spikes can arrive at the same point in time. With the reset-by-threshold scheme, this information is preserved.

The voltages course of two LIF neurons using the two reset schemes is depicted in Figure 4.2. In the upper plot, the neuron is reset to zero after each spike emission. The neuron in the lower plot implements the reset by subtracting the threshold voltage.

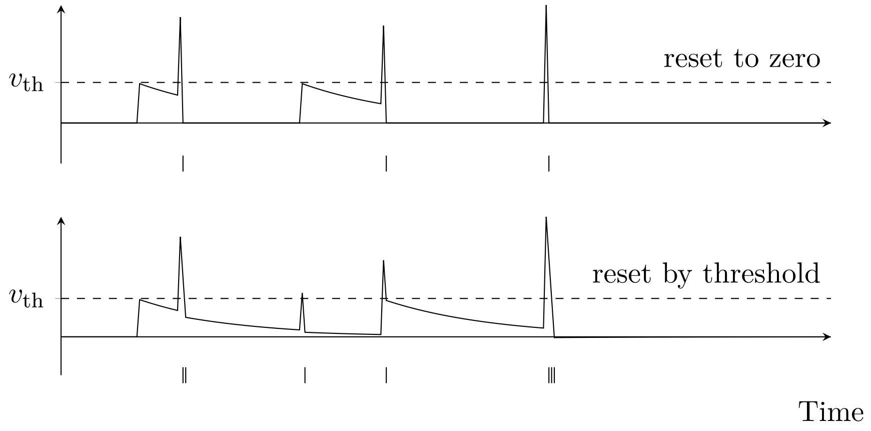


Figure 4.2: LIF neuron reset schemes. The reset-by-threshold scheme preserves the information of high activations. Accordingly, multiple spikes can follow a single excitation with a large amplitude.

4.1.3 Integrator Output Neurons

The output layers of the networks in this work are modeled as non-spiking neurons. A similar approach has been shown in other works simultaneously [175]. These neurons are LIF neurons without the spike and reset functionalities. They solely integrate the incoming weighted spike trains and provide the membrane voltage as output. The continuously evolving membrane voltages can be used to calculate the loss function accurately. This approach is suitable for both, regression and classification tasks. For classification tasks, the softmax function $\sigma : \mathbb{R}^n \to [0,1]^n$ can be applied to normalize the output layer's membrane voltages:

$$\sigma(v_j) = \frac{e^{v_j}}{\sum_j e^{v_j}}. \tag{4.5}$$

This standard method in the field of machine learning enables a suitable projection of the membrane voltages to the class probabilities that they represent.

4.1.4 Recurrence

Equation (4.1) can optionally be extended by a recurrent current $i_{t,\mathrm{rec}}$, which feeds the weighted layer's output of the previous time step $\boldsymbol{z}[t-1]$ back into the population:

$$\boldsymbol{i}_{\mathrm{rec}}[t] = \boldsymbol{W}^{(l,l)}\, \boldsymbol{z}[t-1]. \tag{4.6}$$

The square matrix $\boldsymbol{W}^{(l,l)}$ holds the weights of the recurrent connections. The entries on the main diagonal of this matrix correspond to a recurrent connection of a neuron with itself. This introduces additional processing capabilities to the population. With that, temporal sequences of spatial patterns can be learned and represented, self-exciting loops can be created, and information can be contained over a more extended period of time.

Neftci, Mostafa, and Zenke explicitly differentiate between recurrent networks and recurrently connected networks [49]. Recurrent networks implement latent variables, which introduce a recurrent dependence between the current state and past states. Recurrently connected networks additionally have synaptic connections within the same population. Thus, a recurrent dependence of different neurons between the current state and past states is established. Following this definition, a network consisting of LIF neurons is a recurrent network. In this work, however, the term recurrent refers to the more general definition of recurrence, which includes recurrent synaptic connections.

Reservoir computing, and liquid state machines in particular, heavily utilize recurrently connected populations [176]. In these approaches, a population of sparsely connected neurons, the reservoir, is randomly generated. This population projects the input pattern into a higher dimensional space. The readout layer, which is the only part of the network where learning takes place, projects this representation back to the output [177]. This paradigm can be beneficial to use when the temporal sequences are long, and using BPTT on the whole network might result in vanishing and exploding gradients. However, the network's performance greatly depends on the properties of the randomly generated reservoir. Therefore, insights and improvements of the generation process of such a network are part of active research. In this work, however, we focus on networks that are trained as a whole.

4.1.5 Convolutional Layer

Convolutional layers in neuronal networks convolve small kernels across the input tensor to extract spatial features. The kernels are shift-invariant, so they are applied across all input axes without change. The shifted application of the kernel multiple times acts as there were multiple copies of the same kernel with shared weights, which examine small parts of the input in parallel.

In biological systems, this type of operation can, for example, be found in the nervous system of the eye. Horizontal cells connect neighboring photoreceptor cells and bipolar cells in a way that they are able to detect contrasts. They form circular antagonistic center-surround receptive fields by the interplay of exciting and inhibiting neuronal activity [178]. These circuits are distributed millions of times over the retina and thus compose a convolutional neural network.

Convolutional layers in artificial SNNs are used in various works. Examples are spike-based Convolutional Neural Networks (CNNs), which operate on rate-coded data [19], [92], or networks, which apply the convolution operation directly on the raw input data and apply the binary activation function afterwards [172], [179].

The implementation of a convolutional spiking neuron does not differ much from a conventional LIF neuron. The only difference is in the calculation of the input current. Each neuron is connected to a small perceptive field instead of being connected to the whole input, and the weights are shared with other instances of the same kernel. The membrane voltage update given in eq. (4.1) therefore stays the same.

Convolutional layers in ANNs can extract small gradients and patterns within the input and reflect their presence precisely using continuous activation function outputs. Spiking convolutional layers, on the other hand, evaluate incoming spike patterns and communicate the results to the outside via binary outputs. It is obvious that compromises in the reached accuracy are made at this point if the continuous realization is directly transferred to a spiking implementation.

Most convolutional ANNs implement two-dimensional convolutions. The dimensions are, for example, the two dimensions of an image or temporal sequence of a feature vector, like the frequency and time axis of a spectrum. In the latter case, the time dependence of the input data is represented by one dimension of the convolution kernels. For this purpose, the information of a whole sequence needs to be provided as an input for the convolutional network. Since neuron models in SNNs already contain temporal dynamics, they implement a recursive approach. Accordingly, instead of common approaches using two-dimensional convolutions [172], [180] it can be sufficient to only apply a one-dimensional convolution to the feature vector dimension of the data stream and still extract temporal information. The motivation behind this is twofold: First, processing the data stream continuously reduces the memory consumption since no buffering of data or delaying of spikes is needed. Additionally, buffering or delaying large amounts of data would hardly be realizable in an analog implementation of the neuromorphic circuits. Secondly, it results in a smaller number of weights since the time dimension is not part of the filter kernel.

The convolutional connectivity scheme can additionally be extended with recurrent connections within the population. To restrict the number of connections, we distinguish between recurrent connections between neurons within the kernel but at different positions of the receptive field and connections between the neurons at the same position but between the different kernels.

4.2 Training

With the inputs being encoded and the network components being defined, the networks now have to be trained to meet the desired functionality. First, we introduce the variables which will be optimized during the training. Subsequently, we introduce the regularization techniques, which will be used to improve the ability of the network to generalize on unseen data and to lower the overall network activity. Lastly, pruning is introduced to sparsify the network connections during training.

4.2.1 Optimization Variables

Connection weights are the common optimization variables of ANNs and SNNs. They define the presence and the strength of a connection between two neurons. These weights are adapted during training to minimize the error function and form the neural network with the desired input-output function. Next to these connection weights, SNNs introduce two additional parameters, which can be optimized during training: the time constants of the neurons and their threshold values.

Weights

The network weights are commonly initialized randomly to enable a rich response to the presented input. The goal is to avoid the development of vanishing or exploding gradients, which greatly slow down or even inhibit the convergence of the error function towards the global minimum. For SNNs, the weight initialization is even more important because of their all-or-nothing activation function. Depending on their inputs, certain neurons might never be excited enough to take part in the dynamic processes of the network. Therefore, parts of the network could remain silent no matter the input.

For ANNs, a range of different initialization techniques have been proposed, which determine the variance of the initial weight distributions [181], [182]. Due to the similar derivative of the sigmoid activation function and the spike activation function, we adapt the Glorot initialization [181] to initialize the network weights. This type of initialization aims to maintain the variance of the activations and the backpropagated errors across the network. For that, weights are sampled from a uniform distribution in the interval $[-r, r]$ and

$$r = \sqrt{\frac{6}{N_{\text{in}} + N_{\text{neurons}}}}, \tag{4.7}$$

with N_{in} being the number of inputs and N_{neurons} being the number of neurons in the considered layer. For recurrently connected layers, we have

$$r = \sqrt{\frac{3}{N_{\text{neurons}}}} \tag{4.8}$$

to reflect the connections within the layer.

The firing threshold of the LIF neurons does not have to be considered for the initialization of the weights. As shown in eq. (2.12), the pseudo gradient is defined to lie between 0 and 1. Its value is derived from the current activation of the neuron, which is scaled by the threshold voltage. The output of the activation function (see eq. (4.2)) is defined to be 0 or 1. Its value is thus also dependent on the relation between current activation v_t^j and threshold voltage v_{th}, but not on the absolute values themselves.

Time Constants

A neuron's time constant specifies the leakage current that discharges the membrane over time. It thus describes the retention time of given information in the form of its membrane potential. This parameter can be varied during training to closely match the temporal characteristics of specific features to be detected. With different time constants, a stronger emphasis can be placed on short patterns or long-term relations.

The initialization of the LIF neurons' time constants is highly dependent on the temporal characteristics of the input signals as well as on the temporal scope of the temporal context, which is expected to be considered. From an initial approximation, the constants have to be searched empirically. In the analysis of temporal sequences, it is intuitive to optimize the time constants during the learning process. However, the introduction of more freely optimizable parameters also introduces the potential for overfitting. Therefore, we introduce constraints for closer consideration during the specific applications. Generally, the values can be fixed, optimized network-wide, layer-wise, or for each neuron individually. Unlike connection weights, the time constants do not have to be unique for every single neuron. To adapt for the temporal characteristics of the task to be solved, it can be sufficient to enable the optimization of one time constant, which is shared among the population or the whole network.

Thresholds

The firing threshold defines the membrane potential of a neuron that has to be surpassed to produce a spike. A high threshold voltage, therefore, reduces the probability of a spike being emitted as higher excitations have to be reached. Low thresholds favor the exchange of more spikes, as in the extreme case, a single input spike can be sufficient to excite the neuron above its threshold level.

The threshold voltage can be optimized for each neuron individually or in relation to the population or the whole network, similar to the optimization of the time constant.

In our experiments, it has been shown that a tunable threshold can help the population adapt to the input characteristics faster. Using one jointly optimized threshold per population can speed up the learning process because this adaption does not need to be handled by the optimization of each individual weight.

Joint Consideration

All three optimization variables – weights, time constants, and thresholds – are in close relation to each other. Due to their multiplicative linkage within the LIF neuron model, the same network behavior can be reached using different parameter combinations. During training, fixing one of the variables narrows the parameter space without sacrificing performance. Therefore, in our experiments, we fix the threshold to a global value that is not part of the optimization process.

In a hardware realization of the circuits, closer attention to this simplification can be paid. Neuromorphic hardware might add additional constraints to the space of possible parameters. Weights might, for example, be only feasible in a particular range, the threshold voltage might not be chosen to be arbitrarily large or small, or a different time constant per neuron might not be possible to realize. Therefore, considering all variables and constraints for the realization in hardware is advisable.

4.2.2 Regularization

Regularization is commonly used in the field of machine learning to prevent overfitting and thus reduce the overall variance of the model [1]. The goal of reducing the test error is often reached by adding further optimization objectives to the loss function. Other regularization strategies include early stopping, dataset augmentation, noise injection, or adversarial training. In this work, we use additive optimization objectives and dropout, and expand these techniques for the use in SNNs.

Additive Optimization Objectives

One of the simplest and most commonly used regularization techniques is weight decay, in which large absolute values of the synaptic weights are penalized [1]. This is done by adding the L^2 or L^1 norm of the considered weights to the loss function. Thus, simpler models with small weights are encouraged.

In SNNs, regularization can additionally be used to reduce the overall spike activity of the network, similar to activation regularization in ANNs. To penalize excessive spike activity, the loss function is extended by the activity loss, which is a function of the number of spikes of the respective population. Similar approaches have been proposed by other works simultaneously [172].

The different additive optimization objectives are summed up with the training loss to obtain the overall loss function:

$$\begin{aligned}\mathcal{L} = \mathcal{L}_{\text{train}} &+ \alpha\, \mathcal{L}_{\text{weights}} + \beta\, \mathcal{L}_{\text{activity}}\\ \mathcal{L}_{\text{train}} &= f(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum \boldsymbol{y} \log \hat{\boldsymbol{y}}\\ \mathcal{L}_{\text{weights}} &= f(\boldsymbol{W}) = \boldsymbol{W}^T\, \boldsymbol{W}\\ \mathcal{L}_{\text{activity}} &= f(\boldsymbol{Z}) = \frac{1}{N_n N_{\Delta t}} \sum_{\text{neurons},t} \boldsymbol{Z}.\end{aligned} \tag{4.9}$$

The total loss is thus given by the sum of the training loss $\mathcal{L}_{\text{train}}$, the weight loss $\mathcal{L}_{\text{weights}}$, and the activity loss $\mathcal{L}_{\text{activity}}$. The factors α and β are thereby used to adjust the relation of the losses and with that, the influence of the regularizers.

In our experiments, the training loss is given by the cross-entropy between the true one-hot-coded class labels $\hat{\boldsymbol{y}}$ and the predicted class probabilities $\boldsymbol{y}$. The weight loss is given by the L^2 norm of the weight matrices for every feedforward and recurrent connection. The activity loss is given by the L^1 norm of the spikes $\boldsymbol{Z}$ being emitted. Here, $\boldsymbol{Z}$ contains the information of whether a neuron emitted a spike for every neuron in every time step. Effectively, by normalizing the number of spikes by the number of neurons N_n and time steps $N_{\Delta t}$, we obtain the spike rate of the network and use it as the loss metric.

Our experiments showed that our tested SNNs were very sensitive to activity regularization during the first epochs of training. Especially when combined with pruning techniques, the use of activity regularization resulted in unstable training runs. Therefore, we included a regularization schedule, which adapts the loss scaling factors α and β according to the training progress. While the regularization factors are low initially, they are slowly ramped towards the end of the training run.

Depending on the neuromorphic implementation of the SNN, the types of regularization are beneficial for the overall energy consumption of the network during inference. Obviously, fewer activations lead to fewer spikes being propagated through the network and thus fewer updates of the involved neurons. In analog implementations, smaller weights might result in smaller currents which are needed to charge the membrane capacities and with that less charge being moved during this process.

Dropout Scaling

Dropout is a further regularization technique. During training, a fixed number of randomly selected neurons within a layer is deactivated at each batch [183]. This encourages the remaining neurons to generalize more instead of focusing too much on single examples. DropConnect generalizes this idea to setting a fixed part of the connections (weights) between the layers to zero, creating even more possible combinations [184].

To keep the layer's absolute activation constant during the training and testing phases, the kept node's activations are scaled by $^1/_{1-p}$, with p being the dropout probability. In SNNs, the activation values are not continuous but binary events, which are either 1 or 0. As stated earlier, information is transmitted by the presence and timing of events, not their absolute value. Scaling the values of the events would lead to spikes, which have different values during training and testing time. Due to the binary characteristic of the spikes, the scaling does not average over the whole population.

4.2.3 Network Pruning

In a densely connected neural network, not all connections contribute significantly to the overall performance of the network [185]. Often it is sufficient to use only a subset of the connections of the dense network without sacrificing significant performance losses. These sparse networks, which reach the same performance as the dense starting network when trained independently, are called lottery tickets [11]. Many approaches have been proposed to identify these sparse subnetworks. They can be divided into four categories: (1) training the sparse network from scratch, (2) pruning the network after training, (3) pruning the network during training, and (4) pruning the network even before training [186].

In SNNs, sparsity can not only be reached in the spatial domain by optimizing the structure of the networks but also in the temporal domain by constraining the network activity in the form of exchanged spikes to a minimum. This can be reached using regularization as shown in Section 4.2.2.

In the field of SNNs, populations are often initialized sparsely. This is, for example, the case for reservoir computing, where only the readout connections are trained [176], or hierarchical networks, in which sparsely connected structures are imitated [165]. A popular algorithm, which maintains the sparsity of a given network, is Deep Rewiring [187]. In a sparsely initialized network, a fixed number of connections between pairs of neurons are initialized. These connections comprise a trainable weight and a fixed sign. Deep Rewiring rewires the connections, which are optimized out during the gradient descent-based learning process: as soon as a connection weight gets a negative value, the connection is dissolved, and a new connection is established between two randomly selected neurons. In our experiments, however, this algorithm led to inferior performances.

In this work, we follow the approaches of the third category to identify sparse subnetworks: pruning the network during training. Pruning thereby refers to the deletion of connections between neurons or the neuron itself. Generally, we reached better performances using connection-based pruning. The pruning algorithm used in this work is based on Zhu and Gupta [188]. In this iterative approach, the number of pruned weights is gradually increased over the training steps until the network reaches its final sparsity value s_f:

$$s_i = s_f - s_f \left(1 - \frac{i}{i_{\text{end,prune}}}\right)^3 \quad \text{for}\, i \in \{0, \dots, i_{\text{end,prune}}\}. \tag{4.10}$$

During training, the instantaneous sparsity value s_i is increased until the last training step with pruning $i_{\text{end,prune}}$ is reached. After that, the sparsity value is kept constant to fine-tune the remaining network weights. Pruning is implemented using a binary weight mask, containing a zero at every associated pruned connection. The mask is then updated at every pruning step during training. In their approach, weights are pruned based on their absolute value, pruning the smallest values at the given step.

The advantages of sparse networks are numerous: fewer connections and weights need to be stored, fewer spikes need to be propagated through the networks, training can be less expensive due to the lower number of parameters in the later training steps, and the ability of the network to generalize can be improved because fewer highly detailed connections are available. However, to find the optimum configurations fast and cheaply, and to prevent the whole network from collapsing due to too many connections being removed is still part of ongoing research [189].

4.3 Neuromorphic Hardware Considerations

Neuromorphic hardware aims at the energy-efficient acceleration and emulation of SNNs. Some available platforms have been presented in Section 2.3. They feature a multitude of different system designs: Systems that emulate the dynamics of the SNNs purely in software and use many-core systems to parallelize this simulation. Systems that implement specialized digital circuits to emulate specific neuron models in digital hardware. Or systems that use analog circuits to model neurons, spikes, and entire networks. However, these chips will not be used to accelerate SNNs in edge devices for small applications like keyword spotting due to their sheer physical size. Instead, small accelerators that specifically fulfill the networks' requirements will be integrated into the systems.

Therefore, we discuss the network properties that will influence the design of the specific hardware realization. We will consider the hyperparameters of the network, the influence of different common layer layouts, and influences on the power consumption of neuromorphic hardware.

Table 4.1: Network parameters and their effects in neuromorphic hardware.

Parameter	Effect
Neuron model	Memory/size processing elements/update rate
Number of neurons	Memory, speed/n. processing elements
Number of tunable parameters	Memory/adjustable elements
Number of synapses	Complexity of wiring/communication
Number of spikes/synaptic events	Energy consumption

4.3.1 Mapping Network Hyperparameters to Hardware

Table 4.1 lists some relevant parameters of a network that influence the design of a neuromorphic accelerator.

The number of neurons directly defines the complexity of the accelerator. A large number of neurons necessitates the maintenance of many dynamic variables or analog membranes, which need to be updated frequently. This can either be realized in discrete compute elements or in shared elements, which are used for the calculation of multiple neurons at once. This can be done by time-multiplexing the available resources. SpiNNaker and Intel's Loihi, for example, use their compute cores to simulate the behavior of multiple neurons at once [21], [126]. μBrain, on the other hand, implements one compute element for each individual neuron [190]. Scaling the different architectures can thus be linearly or follow discrete step sizes.

The chosen neuron model defines the complexity of each neuron. The LIF neuron model is one of the simplest models to implement. It comprises one state variable (its membrane voltage), which has to be updated whenever an input to the neuron is present. If the input is a spike train, the updates can be performed in an event-based manner. This simple neuron model is used in most of the available neuromorphic hardware systems (see table 2.1). The more complex ALIF neuron model has one additional state variable: its adaptive threshold. Accordingly, two state variables have to be maintained, stored, and updated for each neuron. The advantages of the event-based processing scheme diminish if the neurons' inputs are not spike-based anymore. In that case, each neuron has to be updated continuously or in each discrete time step, depending on the implementation.

We explicitly differentiate between tunable parameters (weights) and synapses. A synapse is given at the directed connection between two neurons, whereas the weight defines its gain. Many neuromorphic hardware systems use 1-bit weights, weights, which are shared between multiple synapses, or a combination thereof to enable as much design flexibility as possible with the restricted resources [101], [123], [127]. Other tunable parameters are, for example, adjustable time constants or firing thresholds. These have to be stored, too. As the number of values to store can exceed millions for large networks, the storage is often realized off-chip. This flexibility might not necessarily be given in neuromorphic realizations with analog neuron implementations. Instead, the parameters are fixed at the time of the hardware synthesis.

Table 4.2: Comparison of network complexity for different layer types. With m, n, and n_k being the number of inputs, outputs, and kernels. The kernels have the size k.

	Weights	**Synapses**	**Neurons**	**SOPs/input spike**
Feedforward	$m\,n$	$m\,n$	n	n
Recurrent	$m\,n + n\,n$	$m\,n + n\,n$	n	n ($+n$ at output)
Convolutional	$k\,n_k$	$(m-k-1)\,k\,n_k$	$(m-k-1)\,n_k$	$k\,n_k^*$

*Ignoring padding at the edges of the input. Average will be slightly less.

As already stated, synapses are the directed connections between neurons. They enable the exchange of spikes and thus information within the network. In pure analog realizations, each connection has to be realized physically in the form of a dedicated circuit, which connects two neurons. Mixed-signal and fully digital implementations use data buses to utilize physical connections more efficiently. By using packet routers and additional circuitry, the same bus can be used to enable the connection between a multitude of neurons or an inter-chip communication between multiple nodes. Due to the sparse communication scheme of SNNs, the same set of physical wires is thus used in a time-multiplexed way. The number of synapses is limited by the physical dimension of synapse arrays and their weight storage. Additionally, the distance between neurons is important. Depending on the hardware architecture, spikes need to be handed over by multiple packet routers, thus introducing a notable propagation delay. With that, the number of synapses can also influence the reachable processing speeds.

In the event-based processing scheme, neurons are only updated when new information is present. Accordingly, the dynamic energy consumption increases with each spike being emitted. This is where the advantages of sparse communication and a sparse design become apparent. The fewer neurons a spike reaches, the fewer neurons need to be updated, and the less energy is used. At the same time, there is no neuron update if no spike had been produced in the first place. However, our experiments will show that optimizing a network for sparse connectivity increases the number of emitted spikes while maintaining or losing classification accuracy. But as the network gets more sparse, each emitted spike is forwarded to less afferent neurons, resulting in less Synaptic Operations (SOPs). This introduces an interesting design trade-off between the network performance, connection sparsity, spike activity, and SOPs.

4.3.2 Neurons, Trainable Parameters, and Synapses

In the highly parallelized realization of SNNs, there is an important distinction between the number of neurons, trainable parameters, and synapses, which are given by the network or layer design, respectively. Due to their sequential layer-wise computation scheme, this is often not addressed in ANNs. A summary of the relations between the number of weights, synapses, and neurons for different layer types is given in table 4.2.

Neuron layers following the fully connected feedforward scheme connect each input of the layer m to every output neuron n. As a result, there are as many synapses as weights involved. Recurrently connected layers increase the number of weights and synapses by the recurrent connections while keeping the same number of neurons.

The implementation of convolutional layers results in a reduced number of trainable parameters compared to the utilized number of synapses due to the sharing of the kernels' weights. The number of neurons and synapses is still high because unfolding the sliding kernel window results in numerous applications of the same kernel weights on different groups of neurons. The number of synapses and neurons in a convolutional layer greatly depends on the size and number of the kernels involved in the convolution operation. Typically, kernels are small compared to the size of the input. Therefore, even though small kernels lead to a low number of trainable parameters, the total number of parallel synapses and neurons is high.

A further important distinction concerns the number of SOPs, which are triggered by the incoming spikes of a layer. Depending on the layer architecture, this number varies greatly. The relations given in table 4.2 are based on densely connected schemes. Thus, the numbers will be smaller if the layers are sparsely connected. For both feedforward and recurrent layer types, an incoming spike triggers the update of every neuron in the population. For recurrently connected layers, an outgoing spike emitted by the population of the layer itself also triggers the update of every neuron in the population. In convolutional layers, each input feature is perceived by a subset of the neurons in the layer. Here, only those neurons have to be updated, whose receptive fields cover the input feature. Ignoring effects caused by padding at the edges of the input, each input feature is covered by a number of neurons, proportional to the size of the convolutional kernels and the number of kernels.

4.3.3 Energy Consumption

One of the most important metrics to assess the performance of a neuromorphic accelerator is its energy efficiency. Most commonly, this is expressed by the consumed energy per SOP or spike. This, of course, only reflects a part of the total energy consumption of the chip. What is not reflected is the static energy consumption, which is independent of the activity of the network. However, it gives a ballpark figure of the expectable consumption and an idea about how the approach scales.

For a comparison between ANNs and SNNs, Yin, Corradi, and Bothé provide an overview of the dynamic whole-layer energy consumption of different layer types if the SNN is realized in a common digital system [175]. An excerpt is given in table 4.3 for different layer types with m inputs and n outputs. Their considerations are based on the fact, that simple additions consume much less energy than the Multiply-Accumulate (MAC) operations used in common ANNs, thus $E_{\mathrm{AC}} < E_{\mathrm{MAC}}$. This simplification is enabled by the binary activation function of SNNs, which eliminates the need for the multiplication of the current activation value and the connection weight. Instead, the weights of active incoming connections can be summed up. The sparse processing capability of SNNs is reflected by the firing rate parameter Fr, which can be much

Table 4.3: Comparison of the computational costs for different layer types. The layers have m inputs and n outputs. (Adapted from [175])

Type	Network	Energy/layer
ANN	DNN	$mn\,E_{\mathrm{MAC}}$
	RNN	$(mn + nn)\,E_{\mathrm{MAC}}$
	LSTM	$(4mn + 4nn + 3n)\,E_{\mathrm{MAC}}$
SNN	Feedforward	$(mn)\,E_{\mathrm{add}}\,Fr$
	Recurrent	$(mn + nn)\,E_{\mathrm{add}}\,Fr$

smaller than one if a sparse activity is given. In ANNs, in contrast, the whole network is updated every time.

In a digital neuromorphic hardware implementation, the energy costs can be further broken down, as shown by Davidson and Furber [23]. They additionally include the costs of broadcasting the activations through the network, retrieving the weights and neuron states from memory, and writing the newly computed states back to memory. In a feedforward network, this totals:

$$\begin{aligned} E_{\mathrm{totalANN}} &= E_{\mathrm{broadcastActivation}} + E_{\mathrm{retrieveWeights}} \\ &\quad + N_{\mathrm{meanTargets}} \times \left(E_{\mathrm{getState}} + E_{\mathrm{multiplication}} + E_{\mathrm{addition}} + E_{\mathrm{writeState}}\right) . \end{aligned} \tag{4.11}$$

In SNNs, the multiplication operation can be omitted, but the number of operations is now scaled by the number of exchanged spikes:

$$\begin{aligned} E_{\mathrm{totalSNN}} &= N_{\mathrm{meanSpikes}} \times \left(E_{\mathrm{broadcastSpike}} + E_{\mathrm{retrieveWeights}}\right) \\ &\quad + N_{\mathrm{meanSpikes}} \times N_{\mathrm{meanTargets}} \times \left(E_{\mathrm{getState}} + E_{\mathrm{addition}} + E_{\mathrm{writeState}}\right) . \end{aligned} \tag{4.12}$$

Based on the typical energy consumption when realized in the TSMC 22FDX technology, the authors conclude the cost of an adder operation to be E, a multiplier to be $5E$, and reading and writing from and to memory to be $5E$ and E, respectively. The remaining cost components are neglected but would have a larger share in the SNN's energy balance. The total cost for an ANN thus sums up to $5E$ to retrieve the weights and $5E + 5E + E + E = 12E$ for each loop over the targets. Scaling effects and broadcasting costs are thereby neglected. In an SNN, the cost for each target computation is $5E + E + E = 7E$, but this cost is scaled by the number of emitted spikes. Thus, in an equivalent architecture, the spike rate of each neuron needs to be $< 12E/7E = 1.72$ per inference for the SNN to outperform the ANN based on these assumptions [23]. A rate-based encoding scheme contradicts this maximum spike rate. “It suggests that only networks with very low spiking activity justify the use of spikes over conventional ANNs” [23].

The above-mentioned relation only approximates the theoretical abstract energy consumption of equivalent networks, in which each artificial neuron is exchanged by a spiking one. However, this one-to-one relationship is not always practical. When sequences are analyzed, it is often beneficial to use networks that implement memory. LSTMs implement a latent cell state that can maintain information throughout several time steps. This comes at the cost of an increased number of calculations to update this cell state, as shown in table 4.3. The theoretical energy consumption is thus significantly higher than in the simple fully connected feedforward network. In direct comparison with the recurrently connected SNN, the LSTM requires over four times more calculations to be executed when the same number of neurons are used.

5 Speech Recognition – Wake Word Detection

In this chapter, we use and evaluate the methods of the foregoing chapters. Therefore, we solve the key word detection task within the topic of speech recognition. The presented approach[1] combines the usage of resonating input neurons from Section 3.4 and the improvements and approaches of SNNs and their training from chapter 4.

After motivating the topic and giving an overview over the approach, a short review of existing methods is given. Subsequently, the setup of the experiments is presented, and the results are shown. Finally, the results are discussed in the last section of this chapter.

5.1 Motivation (based on [52])

Keyword spotting, as part of speech recognition, is widely used in embedded systems for a wide range of voice-activated assistants. A detector for this purpose can be operated in an always-on mode; therefore, in addition to the recognition rate, energy efficiency is a decisive factor for evaluating a detection system. Another consideration is the detector's ability to perform the desired action in real-time.

Current implementations consist of multiple cascaded detectors of increasing complexity to cope with these requirements. Such detectors range from simple threshold switches over classical algorithmic signal processing to complex neural networks. The growing demand for smart devices and their capabilities expects even better performance with further improved energy efficiency. Many embedded ANN architectures have been proposed to resolve this [8], [191]. Ideally, also large-scale speech recognition should be performed directly in the edge device. Due to its too high complexity, this task is currently offloaded to cloud servers.

In a common, digital, speech recognition implementation, the spoken speech signal is converted into a quantized and discretized format for further processing. Therefore, the alternating pressure waves in the air are transformed into electrical signals using a microphone. An Analog-to-Digital Converter (ADC) converts this electrical signal into the digital domain. This step is followed by the spectral analysis and the generation of the auditory features, which are processed by the ANN-based detectors.

An alternative approach, which can extract complex information while remaining energy-efficient, comprises SNNs. A possible SNN-based approach can directly operate on the analog electrical signal. The RF neurons introduced in Section 3.4 can convert the signal into spatio-temporal spike trains if implemented in analog neuromorphic hardware.

[1] This approach has been previously published in [52].

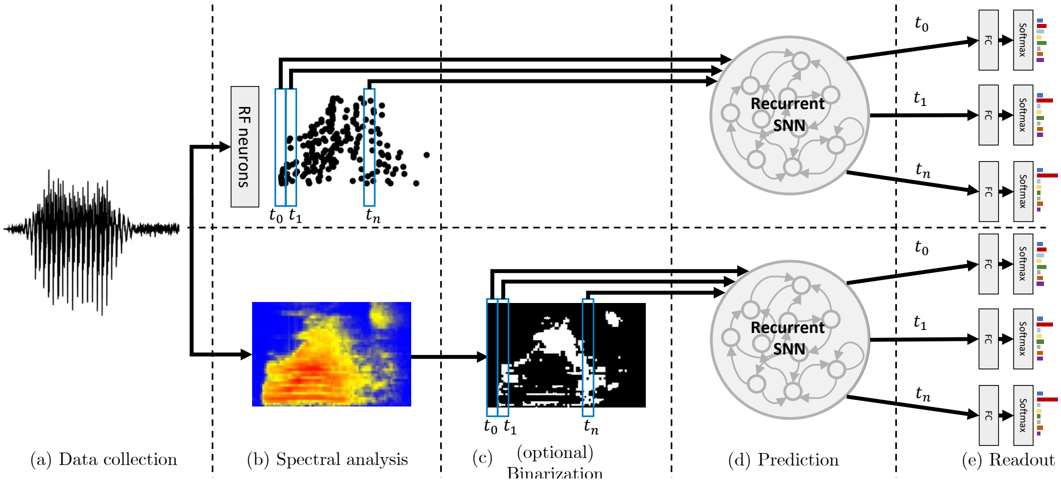


Figure 5.1: Network architecture for the speech recognition task. The speech signal samples are collected (a). Subsequently, the frequency domain features are generated using either RF neurons or classical signal processing approaches (b). Optionally, the Mel-based spectra can be binarized (c) to provide spike representations for the recurrently connected SNN to be recognized (d). Non-spiking integrator neurons, one for each class, indicate the probability of a detected gesture (e). (Figure adapted from [55])

The resulting spike trains are then analyzed by a succeeding SNN (see Figure 5.1). Here, the inherent properties of SNNs can be leveraged: the ability to detect patterns within the input while maintaining low average activity and thus operating energy-efficient.

In this chapter, we compare different modeling approaches for simulating SNN behavior. Simultaneously, we use different neuron behaviors and connectivity strategies to identify the most suitable network for an end-to-end keyword spotting on restricted hardware. Therefore, we demonstrate resonating neurons as input layer to transform an audio signal into a frequency selective spatio-temporal spike representation. Thus, the network can perform keyword detection solely with spiking neurons without using digital signal processing.

5.2 Background

In the last few years, speech processing has been a much-elaborated field in neural network-based processing. In the following, we give the condensed background for the typical generation of auditory features and provide an overview of related works. There, we focus on approaches based on SNNs to provide better comparability for our approach.

5.2.1 Auditory Feature Generation

Speech recognition is not applied directly on the auditory signal in the time domain but on features that have been generated using digital signal processing techniques. These techniques will be briefly introduced in the following. To take a step back, however, we first have a look at the biological feature generation system of our hearing.

Anatomical Example

The generation of speech in the human being is achieved by varying the tension of the vocal cords, the shape of the vocal tract, and the configuration of the lips and nostrils. The spectrum of human speech can be approximated to have nearly constant characteristics in short periods (10-30 ms) due to the anatomy of the vocal cords and the cavities involved in voice generation [192].

The perception of speech signals is carried out by our ears, which convert the alternating pressure waves of sound into spatio-temporal patterns of neural activity [193]. Leaving out complex closed loop interactions between the brain and the auditory sensory organ, the conversion is done by microscopic hair cells, which resonate at specific frequency bands. These hair cells emit spike signals proportionally to their current excitation. The hair cells are located on the basilar membrane in the cochlea of the inner ear. They are arranged in a way that the hair cells, which resonate at the highest frequencies, are located at the base of the basilar membrane, whereas low frequencies are recognized towards the apex (see Figure 5.2a).

The hair cells' excitation patterns are propagated along the auditory nerve via different paths through the brain stem to the audio cortex of the brain. It has been shown that temporal codes are thereby used to encode slowly varying changes, whereas rate codes are used for fast, instantaneous amplitude differences [194]. At the same time, the spike trains originating from hair cells representing low frequencies show phase-locked behaviors [178]. Thus, the spikes always occur at the same phase of the encoded underlying signal. Above 5 kHz, the frequencies are too high to be accurately represented, and the spikes occur at random phases.

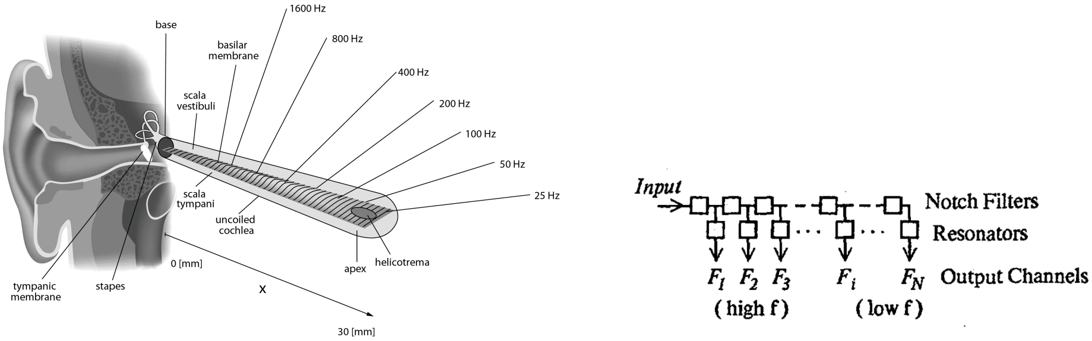
Lyon's Cochlea Model

The computational model of the cochlea by Lyon [195] models the ear as a cascade of filters, non-linearities, and gain controls for digital processing.

The first notch filter cascade mimics the spatial evolution of traveling pressure waves along the basilar membrane, starting with high frequencies (20 kHz) to low frequencies (50 Hz) as shown in Figure 5.2b. A resonator follows each filter to create a bank of sharp band-pass filters. The zero-centered oscillations are then rectified using a half-wave rectifier. This nonlinearity preserves the pitch of the fundamental oscillation while "amplitude demodulating" [195] the waveform. In the last step, a compression module is included to account for a large dynamic range of the input signal.

Mel-frequency Cepstral Coefficients

There is an important difference between the absolute signal frequency and its perceived pitch in human perception. To reflect this, the Mel-scale was introduced, in which doubling the pitch of a perceived tone results in doubling its respective Mel-value. The



(a) Schematic anatomy of the human cochlea. (Figure from [196])

(b) Filter cascade of Lyon's cochlea model. (Figure from [195])

Figure 5.2: Hearnning models. In the human cochlea (a), sound waves are translated into frequency-selective spike trains. Hair cells on the basilar membrane, which perform the translation, resonate in their resonant frequency and emit action potentials proportionally to their excitation. Their resonant frequency, thereby, decreases the farther away they are located from the base of the membrane. Lyon's cochlea model (b) tries to mimic this spatial separation of frequency components using a cascade of notch filters and resonators.

relation between the signal's frequency and its projection onto the Mel-scale is linear at low frequencies and logarithmic and higher frequencies.

Mel- or log-Mel spectra are used in digital audio processing systems to represent sounds like music or speech. In this way, the information needed to convey an understanding of the content is presented in a compressed form. To condense the representation even more, cepstral coefficients were introduced. These Mel-Frequency Cepstral Coefficients (MFCC) have been shown to represent the relevant sound information best compared to other feature generation techniques [197].

The generation of MFCC comprises five distinct steps [198]: (1) Dividing the signal into frames with constant length. Often, consecutive frames overlap by 25% to 50% to enable continuity between the frames. (2) Calculating the amplitude spectrum using the FFT algorithm. (3) Taking the logarithm of the amplitude spectrum. (4) Projecting the resulting spectrum onto the Mel-frequency spectrum. Here, the evenly spaced amplitude spectrum generated by the Fourier transform is scaled that resembles the human perception of the pitch more closely. (5) Computing the Discrete Cosine Transform (DCT) of the resulting scaled spectrum. The result is called cepstrum because of the non-linear operations between the two transforms. Often only a subset of the cepstral parameters is included in the final result.

Depending on the application and the further signal processing, only the first four steps are often conducted to generate the log-Mel spectrum of the input signal without the final calculation of the cepstral coefficients.

Common guidelines are to use 40 Mel-frequency bins computed in step 4, and 13 cepstral features as the result of the DCT computation in step 5 [199]. Additionally, the first and second derivatives of the cepstral features can be included to reflect the evolution of the cepstral features over time.

5.2.2 Related Work (based on [52])

Most modern speech recognition systems are based on non-spiking ANNs. They use recurrent or convolutional network architectures to detect spoken words in audio signals [200], [201]. For this purpose, the input signal is divided into windowed blocks, and a short-time Fourier transform is applied, resulting in a spectrum that changes over time. Typically, the spectrum is then projected to the Mel-frequency scale, or it is further processed to yield the MFCC features.

Other ANN-based approaches exist, which directly analyze the audio stream without prior feature generation. The network learns feature extraction from the ground up while still operating on fixed-sized windows of input data. The proposed deep and convolutional architectures, therefore, exceed millions of trainable parameters and result in large networks [202]–[205]. Accordingly, small-sized networks for embedded use cases are based on Mel feature generation [8].

Due to the inherent relation of speech recognition and biologically inspired networks, there exist numerous approaches for audio processing using SNNs. Early works on biologically plausible audio processing solutions based on SNNs demonstrated small, energy-efficient networks that show stimulus-specific network activities when stimulated with simple stimuli [206]. They used an artificial cochlea [207], [208] to transform the audio signal into a spiking representation.

An overview of SNN-based approaches in speech recognition is given in table 5.1. With recent advances in supervised gradient descent-based learning algorithms for SNNs [47], [49], spiking networks that perform keyword spotting in the spiking domain have been proposed [22], [35], [172]. With the use of SNNs, the focus of the benchmarks is increasingly moving towards the energy-efficient execution of the recognizers [22], [209], [210].

5.3 Setup

In the following experiments, different combinations of feature generation methods and network architectures of ANNs and SNNs are evaluated. The detailed descriptions of the individual components are given in the following.

5.3.1 Dataset

The dataset used for the evaluation of the proposed architecture is Google's Speech Commands Dataset [211]. It consists of the spoken command words *down, go, left, no, off, on, right, stop, up, yes*, background noise, and numerous arbitrary words, which are categorized as unknown. All examples have a duration of one second and are sampled with a frequency of 16 kHz. Each known word has roughly 3000 examples in the training set, whereas the unknown class amounts to 54k examples. The total number of examples

Table 5.1: Related work for speech recognition in SNNs. * = ours

Year	Ref.	Type	Features	Encoding	Learning	Task
2000	[212]	LSM	spectrum	onset, peak, offset	hand crafted	TI46 speech corpus (subset)
2005	[213]	LSM	MFCC, Lyon model	BSA	readout train	TI46 speech corpus (subset)
2005	[214]	FF	Spectrum	ROC	hand crafted	French digits
2010	[215]	FF	LPC	rate	SWAT	TI46 speech corpus (subset)
2010	[216]	CNN, DNN	waveform/wavelets	ROC	evolving	speaker identification
2012	[206]	RNN	silicon cochlea	rate	STDP	Single frequencies
2015	[217]	LSM	Lyon model	BSA	local learning	TI46 speech corpus (subset)
2017	[218]	DNN	spectrum	current to LIF	STDP	Aurora (subset)
2018	[209]	DNN	MFCC	Rate, TTFS	BP	TIMIT (subset)
2018	[219]	HMM	spectrum	rate (Poisson)	STDP	Aurora (subset)
2019	[210]	DNN	MFCC		BP (conversion)	phrase detection
2020	[35]	DNN	MFCC, log-Mel	current to LIF	BP (tandem)	TIMIT, FAME, Librispeech
2020	[180]	DNN, CNN	MFCC	current to LIF	BP (tandem)	Speech Com., Hey Snips
2020	[22]	DNN	MFCC		BP	Speech Com.
2020	[220]	RNN	MFCC	raw	BP, e-prop	TIMIT
2021*	[52]	RNN	MFCC, log-Mel, wavef.	raw, binarized, RF	BP	Speech Com.
2021	[172]	CNN	log-Mel	raw	BP	Speech Com.
2021	[141]	DNN, RF	spectrum, waveform	raw, RF	Slayer	Speech Com.
2021	[173]	RNN	MFCC, log-Mel	raw	BP	Speech Com., Heidelberg, TIMIT

Table 5.2: Parameters of the Mel- and MFCC-based feature generation.

Parameter	Value
Frame length	30 ms
Hop length	10 ms
Minimum frequency	133 Hz
Maximum frequency	6854 Hz
$N_{\mathrm{Melfrequencies}}$	40
N_{MFCC}	13 (+ 13Δ and 13$\Delta\Delta$)

in the training set is therefore 85,511. In the test set, each known word is represented by approximately 400 examples.

Class weights are introduced to account for the unbalanced dataset, which scale the training error according to their numerical over or underrepresentation within the dataset, respectively.

5.3.2 Evaluation Metrics

To enable comparability with other approaches, we report the classification accuracy on the test set. The winning class is selected by the largest value of the neurons in the last layer of each network using the softmax activation function. The number of neurons, synapses, trainable parameters, and state variables are determined by the chosen architecture. They give an idea about the complexity of the network as introduced in table 4.1. For the architectures based on SNNs, additionally, the number of spike events per example is recorded and averaged during the evaluation of the test set. The total number of synaptic events can be calculated with the information about the fan-out synaptic connections for each neuron. This value reflects the expectable dynamic power consumption during inference. For the baseline ANN architectures, the number of MAC operations is reported for comparison.

5.3.3 Input Encoding

The raw audio stream has to be encoded to be processed by the neural networks.

The RF neuron-based approach is compared with the common feature generation for speech detection based on Mel-frequencies.

Mel-frequencies and MFCC

The RF-encoded approaches are compared to traditional methods. Therefore, log-Mel features and MFCC are used. The parameters chosen are based on typical values used in the literature and in other related works [172], [180]. The parameters are summarized in table 5.2.

Table 5.3: Parameters of the RF neurons used for the speech recognition task.

Parameter	Symbol	Value set 1	Value set 2
Damping constant	λ	20 Hz	20 Hz
Initial threshold	$v_{\mathrm{th},0}$	1.0	0.5
Threshold adaption	$f(v_{\mathrm{th}})$	$v_{\mathrm{th}} \mapsto 2\, v_{\mathrm{th}}$	$v_{\mathrm{th}} \mapsto 2.1\, v_{\mathrm{th}}$
Minimum frequency	f_{min}	20 Hz	20 Hz
Maximum frequency	f_{max}	2000 Hz	4000 Hz
Frequency spacing	-	linear	linear
N_{neurons}	-	40	100

The 40 Mel-frequencies cover a frequency range between 133 Hz and 6854 Hz. Thereby, the first 13 filters are linearly spaced with their center frequencies placed between 200 Hz and 1000 Hz, and the following 27 filters are logarithmically spaced [199]. Because the bandwidth of the logarithmically spaced filters grows with higher frequencies, their amplitude is normalized so that the filter's energy stays constant. The logarithm is applied to the Mel-spaced spectrum to obtain the standard log-Mel-spectrum. Additionally, the MFCC and their first and second discrete derivatives are computed.

In contrast to the spike-based encoding using RF neurons, the traditional methods evaluate fixed sections of the signal, frames, which are successively moved along the time axis. A frame has a length of 30 ms, and the hop length equals 10 ms. Because the speech recordings last one second, this results in 98 partially overlapping frames.

To account for the spike-based communication scheme of the SNNs, the resulting input matrices are additionally binarized using the mean binarization scheme shown in Section 3.2.3. Both the raw and binarized inputs will be applied to the different architectures.

Resonate-and-fire Neurons

In the third configuration, the raw audio stream is encoded into a spike representation using resonating neurons (see Section 3.4). This approach is inspired by Lyon's cochlea model, introduced in Section 5.2.1: A bank of resonators is used to select spectral components of the input according to their resonance frequency. In contrast to Lyon's model, the amplitude demodulation is carried out by the non-linear spike generation. Additionally, the compression filter is replaced by the adaption of the spike threshold to account for a large range of input amplitudes.

The resonating neurons are simulated using exact solving of the coupled differential equations, which describe the system. Exact solving is applicable here because the input signal $i(t)$ is assumed to be constant between two succeeding samples. Although the input values are digital samples during this simulation, the observations are transferred to a pure analog implementation. The consideration of the Nyquist-Shannon sampling theorem during the sampling of the dataset and the exact solving of the neuron's equation system justify this assumption.

The parameters of the neuron model have been determined analytically using the relations described in Section 3.4, and have been tuned empirically to adapt to the characteristics of the input data. The resulting parameters are summarized in table 5.3. Two variants with different maximum frequencies and a different number of resonating neurons are implemented to enable further comparisons. In both cases, the neurons' resonant frequencies are spaced linearly. However, since more resonators are available in the second value set, the maximum frequency is increased to 4000 Hz. Simultaneously, the neurons' bandwidth for which they generate output spikes is narrowed in the second set. This ensures less overlap between adjacent resonant frequencies. The damping constants and minimum frequencies are equal in both cases because these mainly depend on the analyzed dataset.

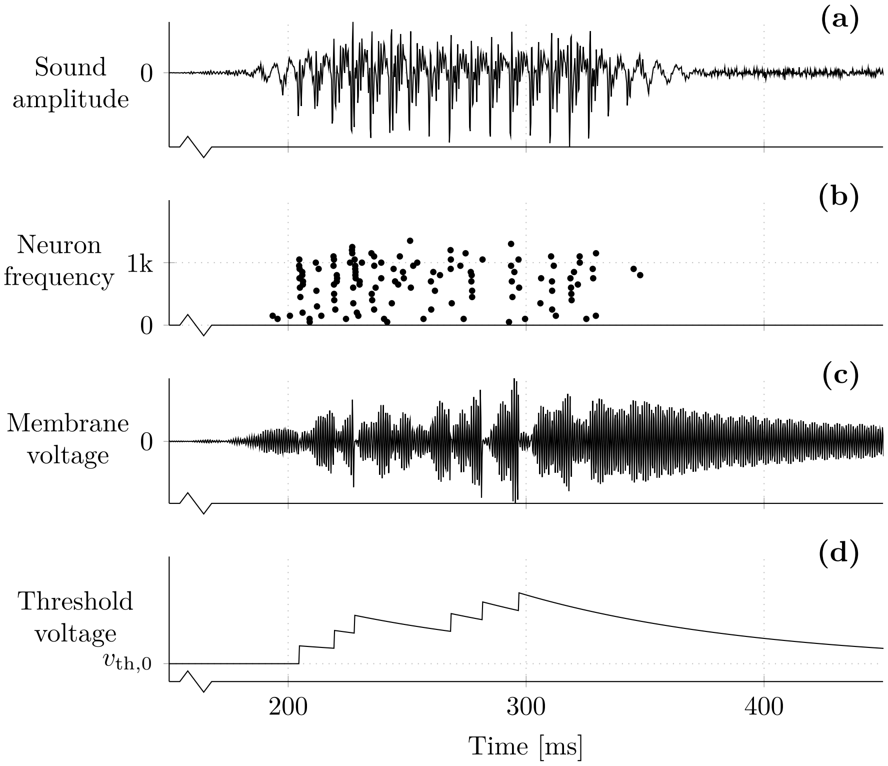


Figure 5.3: Exemplary evaluation of the encoding of a speech signal using RF neurons. (a) shows the waveform of the analyzed spoken word *right.* The corresponding spike response pattern of the encoding RF neurons is shown in (b). The course of the voltage-like variable of one of the RF neurons and its threshold voltage adaption is shown in (c) and (d). (Figure previously published in [52])

The exact spike times of the resonating neurons are discretized. The reason behind this is two-fold: (1) The SNNs are simulated in discrete time steps to enable BPTT-based learning (see Section 2.2.3). This restriction, however, is only necessary during the training phase of the networks. For inference in production, the networks can be operated continuously or with a different discretization. For this, it is necessary to adjust the neurons' parameters to ensure similar behavior. (2) For the comparison with the baseline ANN models, the input needs to be present in a discretized form. The discretization intervals are chosen to be similar to the frame length of the MFCC- and Mel-based inputs. The resulting input format for the network simulation are sparse matrices, which have non-zero entries at the time-frequency coordinate at which the neuron with the respective resonant frequency emitted a spike.

An exemplary visualization of the encoding process using RF neurons is shown in Figure 5.3. The figure thereby only shows the relevant time span in which the signal contains the voice. The raw speech signal of the arbitrarily chosen word *right* is fed into the bank of resonating neurons. The neurons start to resonate depending on their resonant frequency and the amplitude of the respective signal components. If the excitation of a single neuron is high enough, it produces an output spike, its membranes are reset, and the threshold value is increased according to the threshold adaption rule. The resulting spike train is depicted in the second row of Figure 5.3. The temporal evolution of one neuron's voltage-like membrane variable is shown in the third row of the figure. There, the positive responses to the respective frequencies in the signal are visible. After each reset, the threshold voltage of the neuron is increased, as shown in the bottom row. The threshold voltage slowly decreases to the initial threshold value if no spike is generated for a long period of time. This enables the neuron to spike again in the presence of lower excitations.

Visual Comparison of the Speech Features

Figure 5.4 shows a visual comparison of the feature generation methods using RF neurons, Mel-frequencies, and MFCC. The parameters for the feature generation are shown in tables 5.2 and 5.3. Again, the word *right*, pronounced by a female speaker, is used for demonstration. The two sets of parameters for the RF neurons are primarily used for the performance evaluation of the whole networks. However, it can be seen that the extension of the frequency range up to 4 kHz includes more distinctive features. Qualitatively, the two spike trains and the spectrum based on Mel-frequencies look comparable. A high activation can be seen during the presentation of the spoken word. During the time interval between 200 ms and 300 ms, especially low frequencies are present. Then, higher frequency components start to be recognizable. The MFCC can not be visually compared because, due to the cosine transform, the features do not represent a spectrum anymore.

5.3.4 Network Architectures

In our experiments, we compare the above-mentioned feature generation approaches in combination with different network architectures of ANNs and SNNs. The networks comprise feedforward, convolutional, and recurrently connected layer types. They thus represent a broad base of different network architectures.

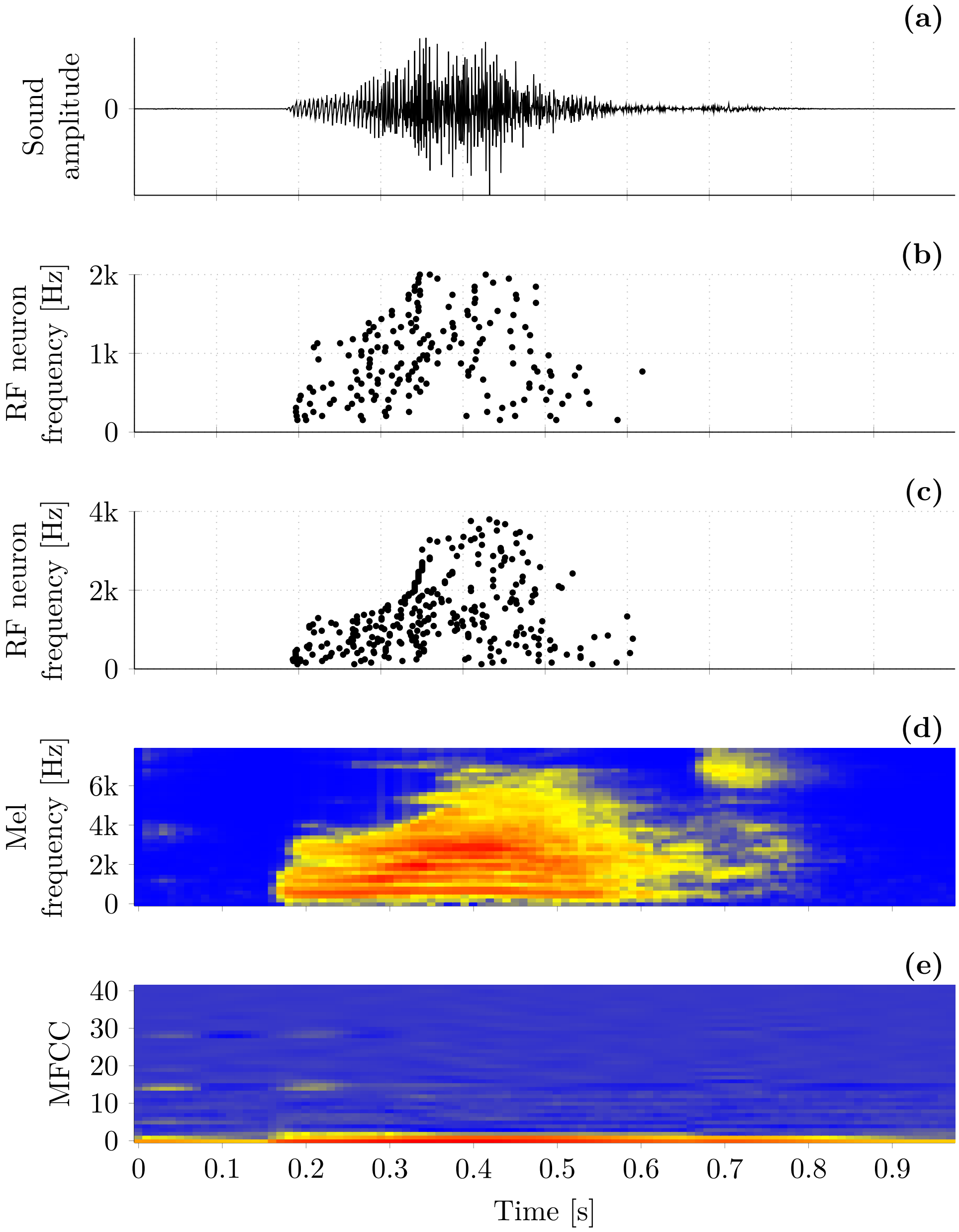


Figure 5.4: Feature generation for an exemplary spoken word. The waveform of the spoke word *right* is shown in (a). (b) and (c) show the spike response of the RF neuron banks using the two value sets in table 5.3. (d) depicts the log-Mel histogram of the audio snipped. The sequence of MFCCs and their first and second derivatives is shown in (e).

ANN Architectures

The baseline models used for comparison are non-spiking ANNs. They feature simplified architectures of common networks given in the literature. The networks do not fully reach current state-of-the-art performance but are a ballpark figure which performance to expect from the different types with the given experimental setup. The inputs to the different networks are the same as for the SNNs to enable comparability.

The feedforward type network – Deep Neural Network (DNN) – consists of one or more layers of dense Fully Connected (FC) layers. Because the network itself does not offer any temporal memory or specialized structures, the input sequences are flattened and are presented to the network at once.

The recurrent network consists of LSTM cells. The network can thus save its state in the cell's latent variables. With that, a memory is created, and it is possible to process data sequences with temporal features.

The convolutional network – CNN – comprises convolutional layers for spatial feature extraction and temporal sequence detection. Accordingly, the first convolutional layers convolve the kernels over the two-dimensional input of multiple stacked frames. The posterior layers convolve over the time domain of the sequence to extract the temporal information of the spoken words.

A fourth network architecture – Recurrent Convolutional Neural Network (RCNN) – combines the convolutional and recurrent network types. The network consists of a convolutional layer, which is used to extract spatial features within the input. Subsequently, a LSTM layer is used to extract the temporal variances.

Descriptions of the architectures are also given in table 5.4.

SNN Architectures

The benchmarked networks comprising spiking neurons include feedforward, recurrent, and convolutional structures, too (see table 5.5).

Similar to the baseline ANNs, the inputs of the SNNs are either the spike trains generated by the RF encoding layer, or temporal patterns, which are provided by the preprocessed Mel- and MFCC-based inputs. The output layer consists of non-spiking integrators – one for each class – which are evaluated after each training example to calculate the respective loss and accuracy values (see Section 4.1.3). The populations between the input and the output layer consist of LIF neurons with different connectivity schemes.

In the network implementing feedforward connectivity, weighted synapses solely connect neurons of different layers without creating backward connections. In contrast to the baseline feedforward ANN implementation, this already enables recurrence within the network. Recurrence is already given by the latent hidden variables of the neurons.

The second network implements recurrence explicitly. It includes additional connections within a population to enable communication between neurons through time.

The examined convolutional networks implement kernels with small receptive fields, which are instantiated multiple times with shared weight variables (see Section 4.1.5).

Table 5.4: Architectures of the examined ANNs for the speech detection task. FC: fully connected dense layer with (number of neurons) and ReLU activation. Output: fully connected layer with softmax activation. C: convolutional layer with (number of kernels, kernel size in time dimension, kernel size in frequency dimension, stride in time, stride in frequency) and ReLU activation. MaxPool: max pooling with (pool size in time, pool size frequency). Lin: linear layer, which is a FC layer without activation function. LSTM: recurrently connected layer with (number of LSTM cells).

Type	Layer	Parameters	MAC Operations
DNN	FC(50)	50,050	50,050
	FC(50)	2,550	2,550
	Output(12)	612	612
	Σ	53,212	53,212
LSTM	LSTM(92)	48,944	4,857,600
	Output(12)	1,116	1,104
	Σ	50,060	4,858,704
CNN	C(20,10,4,1,1)	820	2,693,600
	MaxPool(2,4)	-	136.800
	C(30,10,4,2,2)	24,030	1,296,000
	Lin(16)	25,936	25,936
	FC(64)	1,088	1,088
	Output(12)	780	780
	Σ	52,466	4,154,204
RCNN	C(30,10,4,2,2)	1,230	1,048,800
	MaxPool(1,4)	-	46,080
	LSTM(65)	48,360	2,212,600
	Output(12)	792	780
	Σ	50,382	3,308,260

Within the scheme of convolutional networks, we examine three different architectures. The first two layers comprise one-dimensional convolutions, which only extract spatial features of the frequency axis. The kernel size on the time axis is one. Accordingly, temporal features are extracted by the LIF neurons. To evaluate the relevance of recurrent connections within the convolutional kernels, one of the one-dimensional CNN incorporates recurrent connections. The architecture featuring two-dimensional convolutions of both the time and the frequency axis is a scaled-down version based on [172].

The temporal evolution of the SNNs is simulated in discrete time steps to enable backpropagation learning. This part of the network is subject to the optimization of connections, weights, and hyperparameters.

5.4 Evaluation

In this section, the proposed end-to-end approach of using solely spiking neurons for speech recognition is evaluated. Therefore, different network architectures are evaluated and compared, followed by insights into the process of training and an evaluation of the networks' complexities. The section concludes by comparing the obtained results to state-of-the-art solutions given in the literature.

5.4.1 Classification Performance

The classification performance reached by the different model and input combinations is shown in table 5.6. Detailed network descriptions of the respective models are given in the appendix in tables 5.4 and 5.5. All of our evaluated networks have the constraint of a maximum of 50,000 parameters in common. This number marks the beginning of the point of diminishing returns, such that a much larger network results only in marginally higher classification performances. We will show this relation in Section 5.4.3. To enable the comparability between Mel-based and RF-encoded inputs, only the first value set shown in table 5.3 is used. With that, the dimensionality of the input feature vector is the same for all inputs.

The top half of table 5.6 shows the classification performance of the non-spiking ANNs. The best result of each model is marked in bold. In all cases, the performance of non-binarized input data is notably higher than the binary inputs. This also applies to the input based on the RF neurons.

The reported classification performances do not reflect the current state-of-the-art performances given in the literature. Attention-based CNNs, for example, have been shown to reach even higher classification accuracies [221], [222]. However, the reached performances give an idea about what the simple architectures in our experiments are capable of. It shows that given the different preprocessing steps on the audio data, the networks are able to reach 93% accuracy on the benchmark dataset with around only 50,000 tunable parameters.

The classification performances of the SNNs is shown in the lower half of table 5.6. The networks can solve the classification problem with a comparable but consistently lower accuracy compared to the ANN-based architectures.

Table 5.5: Architectures of the examined SNNs for the speech detection task. FC: fully connected dense layer with (number of LIF neurons). Output: fully connected layer with softmax activation. C: convolutional layer with (number of kernels, kernel size in time dimension, kernel size in frequency dimension, stride in time, stride in frequency). Rec. FC: recurrently connected layer with (number of LIF neurons). SOPs/Spike refers to the number of neuron updates, which are triggered in the next layer by each spike emitted by the current layer.

Type	**Layer**	**Neurons**	**Params**	**Synapses**	**SOPs/Spike**
DNN	Input				200
	FC(200)	200	8,002	8,000	200
	FC(200)	200	40,002	40,000	12
	Output(12)	12	2,400	2,400	
	Σ	412	50,404	50,400	
RNN	Input				200
	Rec. FC(200)	200	48,002	48,000	212
	Output(12)	12	2,400	2,400	
	Σ	212	50,402	50,400	
1D-CNN	Input				$\approx 132^*$
	C(33,1,4,1,2)	627	167	2,508	70
	Rec. FC(70)	70	48,792	48,790	82
	Output(12)	12	840	840	
	Σ	708	49,798	52,138	
1D-RCNN	Input				$\approx 132^*$
	Rec. C(33,1,4,1,2)	627	1,882	20,691	101
	Rec. FC(68)	68	47,262	47,260	80
	Output(12)	12	816	816	
	Σ	706	49,334	68,767	
2D-CNN	Input				$\approx 504^*$
	C(42,4,3,1,1)	1,596	548	19,152	$\approx 504^*$
	C(42,4,3,1,1)	1,344	21,212	677,376	$\approx 504^*$
	C(42,4,3,1,1)	588	21,212	296,352	12
	Output(12)	12	7,056	7,056	
	Σ	3,540	50,028	999,936	

*Ignoring padding at the edges of the input. Average will be slightly less.

Table 5.6: **Classification performance of different architectures for the speech detection task.** For detailed architecture descriptions, see tables 5.4 and 5.5. Combinations marked with "-" did not converge at all.

Type	Model	MFCC/Δ/ΔΔ		log-Mel		RF
		raw	bin	raw	bin	(set 1)
ANN	DNN	64.94	16.97	**77.38**	43.99	50.98
	CNN	88.12	73.87	**88.92**	66.75	71.06
	LSTM	**89.96**	87.34	86.20	75.52	86.11
	RCNN	91.53	84.91	**93.09**	78.20	84.68
SNN	DNN	**88.83**	86.44	60.98	72.47	79.67
	RNN	-	**88.81**	-	78.83	84.82
	1D-CNN	**88.85**	83.46	77.95	69.82	82.56
	1D-RCNN	**90.08**	84.44	79.90	72.97	83.08
	2D-CNN	89.20	81.19	**91.19**	71.92	-

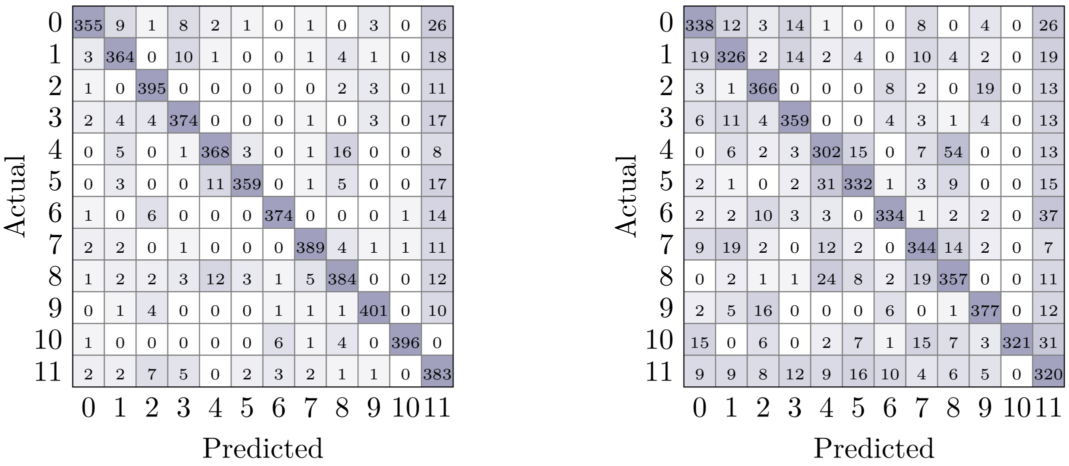
For both network types, ANNs and SNNs, the architectures based on convolutional layers show the best classification accuracy. A comparison between the spiking 1D-CNN and the 1D-RCNN networks highlights the importance of recurrent connections within a convolutional layer. Though having marginally fewer trainable parameters, the evaluation of the test set results in a higher classification accuracy.

Interestingly, the recurrent SNN, which marks the best performing spiking network architecture on the RF-encoded dataset, does not converge on raw Mel-based data. Only if the data is binarized can the network learn the essential correlations. In that case, it outperforms all other architectures, including those based on ANNs. Only with RF-encoded input features, the recurrently connected SNN is beaten by the non-spiking LSTM architecture.

The confusion matrices in Figure 5.5 show the correct and false classifications on the test set of two different networks. Both show a high misclassification rate for the unknown class (11) and the word pair {*off* (4), *up* (8)}. The SNN based on the RF input shows additional confusion pairs. The most prominent are {*down* (0), *go* (1)}, {*go* (1), *no* (3)}, and {*off* (4), *on* (5)}. The same pairs can also be found for an LSTM network on RF input (Figure A.1a), suggesting that the resonating input encoding clips relevant information. Comparing the best performing ANN and SNN networks on the log-Mel spectral input (Figures 5.5a and A.1b) shows that both network generations can achieve similar results on the same input dataset.

5.4.2 Network Dynamics of the SNN

As shown in the previous section, the SNNs in the various combinations are able to reach accuracy levels, which are comparable with those of the baseline ANNs. We now look into the dynamics of the spiking networks and the behavior of the optimization variables during training.



	0	1	2	3	4	5	6	7	8	9	10	11
0	355	9	1	8	2	1	0	1	0	3	0	26
1	3	364	0	10	1	0	0	1	4	1	0	18
2	1	0	395	0	0	0	0	0	2	3	0	11
3	2	4	4	374	0	0	0	1	0	3	0	17
4	0	5	0	1	368	3	0	1	16	0	0	8
5	0	3	0	0	11	359	0	1	5	0	0	17
6	1	0	6	0	0	0	374	0	0	0	1	14
7	2	2	0	1	0	0	0	389	4	1	1	11
8	1	2	2	3	12	3	1	5	384	0	0	12
9	0	1	4	0	0	0	1	1	1	401	0	10
10	1	0	0	0	0	0	6	1	4	0	396	0
11	2	2	7	5	0	2	3	2	1	1	0	383

(a) ANN - RCNN on log-Mel spectogram

	0	1	2	3	4	5	6	7	8	9	10	11
0	338	12	3	14	1	0	0	8	0	4	0	26
1	19	326	2	14	2	4	0	10	4	2	0	19
2	3	1	366	0	0	0	8	2	0	19	0	13
3	6	11	4	359	0	0	4	3	1	4	0	13
4	0	6	2	3	302	15	0	7	54	0	0	13
5	2	1	0	2	31	332	1	3	9	0	0	15
6	2	2	10	3	3	0	334	1	2	2	0	37
7	9	19	2	0	12	2	0	344	14	2	0	7
8	0	2	1	1	24	8	2	19	357	0	0	11
9	2	5	16	0	0	0	6	0	1	377	0	12
10	15	0	6	0	2	7	1	15	7	3	321	31
11	9	9	8	12	9	16	10	4	6	5	0	320

(b) SNN - RNN on RF inputs

Figure 5.5: Confusion matrices for the best performing ANN and SNN. Both networks comprise 50,000 parameters. The ANN shown in (a) is a RCNN and is applied on log-Mel spectrograms. The SNN in (b) is a recurrently connected network. Its inputs are RF-encoded spike trains. Both networks show the most misclassifications at wrongly predicted unknown class (11) or pair-wise confusions like {off (4), up(8)}. The labels correspond in ascending order to the words *down, go, left, no, off, on, right, stop, up, yes*, background noise, unknown.

Temporal Behavior

To give a general impression of the processes in the spiking network, an example evaluation is shown in Figure 5.6. The depicted network consists of a single hidden population with 200 recurrently connected LIF neurons. The second row shows the sparse activity within this population. The class probabilities are obtained by the membrane voltages of the neurons in the output layer. The voltages are normalized using the softmax activation function. The course of output corresponding to the correct output class is colored in green. The bottom row of the figure displays the membrane voltage course of an arbitrary neuron within the hidden population. The stroked horizontal line represents the threshold voltage of the neuron. The stroked vertical lines illustrate the points in time when the membrane voltage crosses this threshold and the neuron, thus, produces an outgoing spike.

Figure 5.6 shows the network activity when excited by the command *off*. The evaluation of the network output underlines the results of the confusion matrices shown earlier. Up to a certain point, the network favored an output that did not match the correct class, as depicted in red. This output corresponds to the word *up*. During the phonetically similar pronunciation of the two words, the network predicts the wrong class until the phoneme is present, which allows for a distinction between the words.

Another aspect, which is visible in the graphic, is the sparse network activity during the inference. In fact, there is no activity at all during the majority of the simulation time. Only during the time frame in which the excitation of the RF neurons is high enough to produce spikes does the network exchanges spikes for further information processing.

Training

During training, the trainable variables are optimized to approximate the desired input-output relation. In this setup, the trainable variables are the weights between the input layer and the hidden population, the recurrent connections within the population and the connections between the hidden population and the output layer. Additionally, the firing threshold and the time constants of the hidden neurons are optimized. In Section 4.2, we introduced these additional optimization variables and stated, that these can be optimized either for each neuron individually or jointly for the whole population. In our experiments, a joint optimization, thus, optimizing one threshold and one time constant for the hidden layer, lead to the best results.

The adaption of the connection weights of the same network as in the previous figure during training is shown in Figure 5.7. The figure shows the evolution of the weights and their average absolute change in its three columns. The rows correspond to the different groups of weights: the connections between the RF neurons (input), the recurrent connections of the hidden layer, and the connections between the hidden layer and the output neurons. While the distributions of the weights stay roughly constant at the input and output connections, the weights of the recurrent connections show larger fluctuations. As the histogram shows, most of the weights are located close to the average value, which is nearly zero. At the same time, the minimum and maximum values of the weights are

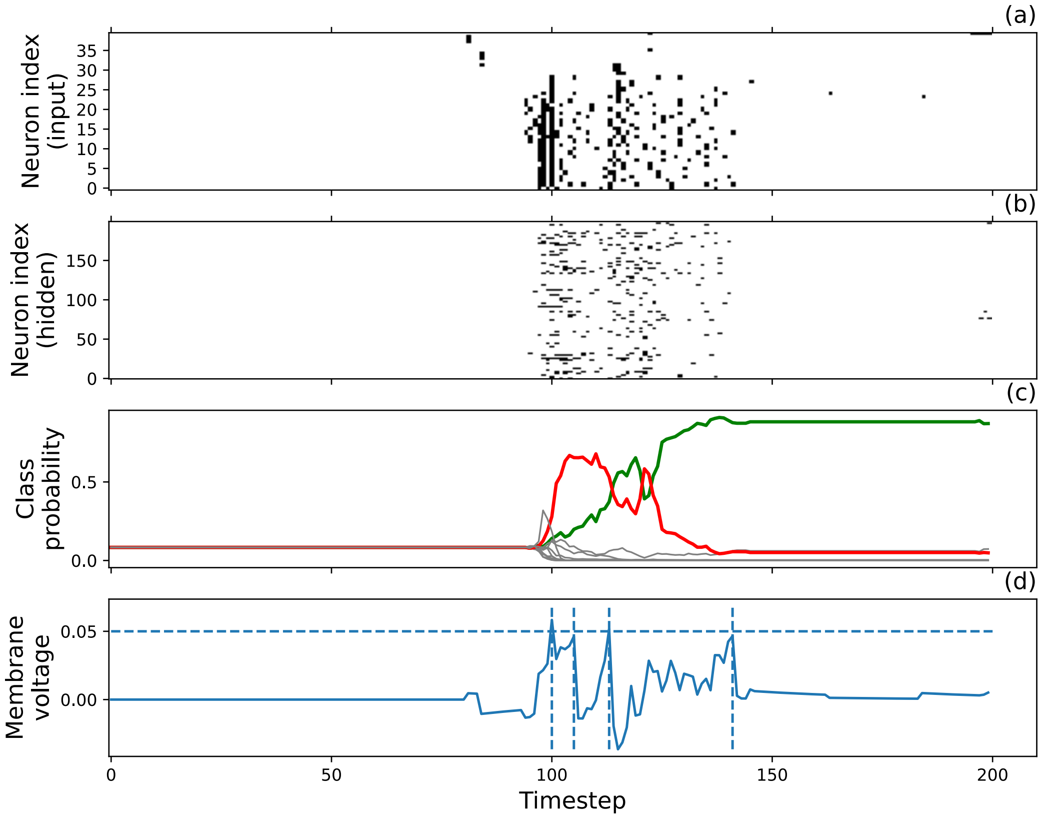


Figure 5.6: Exemplary evaluation of the SNN's inference of a command. (a) shows the spike train of the word *off* using the RF neuron value set 1 in table 5.3. (b) visualizes the spike activity in the hidden layer of the SNN. The spike emissions are sparse in time and across the neurons. The courses of the normalized membrane voltage of the non-spiking output neurons are shown in (c). The course of the neuron corresponding to the correct label is shown in green. The red line corresponds to the word *up*, which is preferred during the first half of the prediction. The plot at the bottom (d) shows the course of the membrane voltage of one arbitrary neuron in the hidden layer. The stroked horizontal line visualizes the threshold voltage. The stroked vertical lines illustrate the spike events.

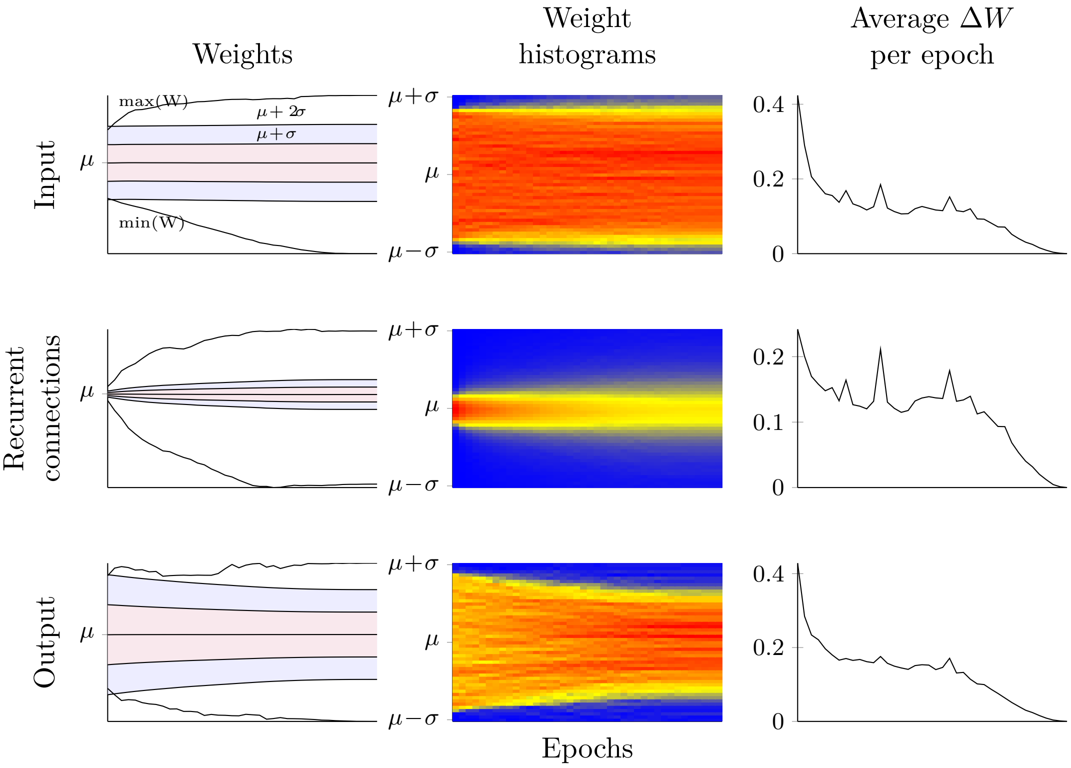


Figure 5.7: Weight change during training. The left column shows the course of the average values of weights (μ), the areas of the standard deviations ($\mu + \sigma$ and $\mu + 2\sigma$) as well as the minimum and maximum values of the weights (min(W) and max(W)) of the input connections, recurrent connections within the hidden population and the output connections during training. In the mid column, the histograms of the weight distributions are shown. The displayed range corresponds to the red shaded area in the left column. The average weight change of the respective connection group in each epoch is shown in the right column.

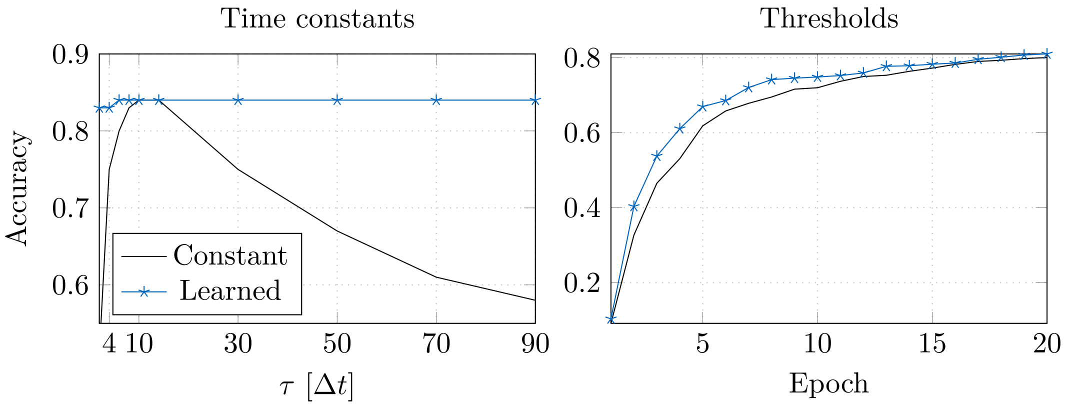


Figure 5.8: Trainable time constants and thresholds. The influence of different initial time-constants $\lambda = e^{-1/\tau}$ and on the final test-set accuracy is shown in the left plot. The graphs show the reached classification accuracy for constant decay values or time constants which were learned during training. The right plot shows the evolution of the training error during training. Both configurations reach the same accuracy, however, the network with the trainable threshold converges faster.

significantly larger than in the other two cases. Thus, most connections are not relevant for the processing of the information. Those, which are important, however, are adapted accordingly. In the next section, we will prune these unnecessary, near-zero connections.

Figure 5.8 shows the influence of trainable time constants and thresholds. The left plot shows the reached accuracy for network configurations with different initial time constants of the neurons in the hidden population. Thereby, the constants were either fixed or adaptive during the training phase. The networks with trainable time constants consistently reached an accuracy of about 84%. The networks with constant configurations, however, showed significantly worse performances. The intersection point of both graphs is in the region around $\tau = 12\Delta t$. This also marks the final value of the time constant in the networks that were able to optimize the constant. We observed no difference between individually learned time constants and layer-wise shared ones in this application. However, in a setting with richer temporal variances within the input, the networks might benefit from the individual per-neuron optimization.

The evolution of the training error for a fixed firing threshold and a population-wise optimized threshold is shown in the right plot of Figure 5.8. Both graphs show the averaged course over multiple runs. In both cases, the same error is reached. However, due to the introduced freedom of the trainable threshold, the error converges faster if the optimal threshold has not been guessed initially. Our experiments' differences in the convergence time introduced through the two approaches are reproducible but small. The actual advantage of the introduced degree of freedom might manifest itself in much larger network configurations.

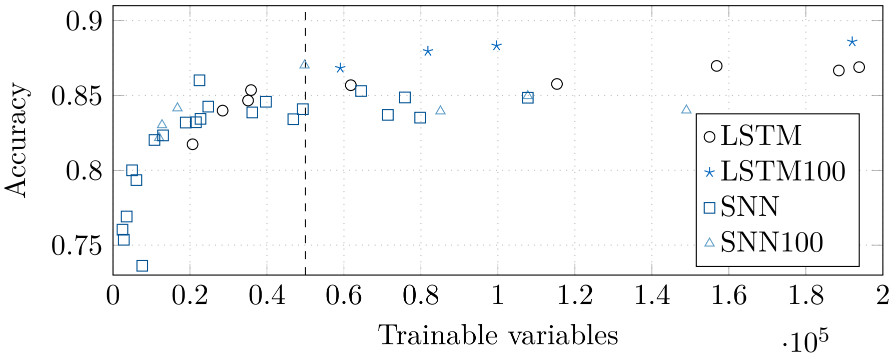


Figure 5.9: Network size sweep. Classification performance versus network size of LSTMs and recurrently connected SNNs on RF-encoded input data. Here, the baseline configuration as well as the second value set consisting fo 100 RF neurons is examined. The sweep is performed for both value sets for the RF neurons (see table 5.3). The stroked vertical line marks the number of 50,000 trainable variables. After that point all networks show diminishing returns when further increasing the network size. Generally, the networks, which use the input spike trains of 100 RF neurons perform slightly better than the variants using only 40 neurons.

5.4.3 Ablation Study

The following studies examine individual aspects of the evaluated network to get a more profound understanding of the involved SNNs. We base these experiments on the non-spiking LSTM and the spiking RNN using RF-encoded input features. We use this combination because of the simple structures of the two networks and their affinity to the RF-based input.

Network Size

First, the relation between network size and the reachable classification performance is analyzed. In addition to the first value set in table 5.3, we also evaluate the classification performance using the second value set with 100 RF neurons.

Figure 5.9 shows the result of this experiment. The four classes represent the LSTM and recurrently connected SNN, with the RF encoding being parametrized according to the two value sets in table 5.3. All four classes experience a saturation of the reached classification accuracy with higher numbers of trainable variables. For the two non-spiking LSTM variants, the saturation is qualitatively reached at larger network sizes, resulting in an overall higher achieved accuracy.

Generally, higher accuracies can be reached using the second value set of the RF neurons. With the 100 RF neurons, a better representation of the sound spectrum is given. Because more neurons are used, the bandwidth for which each individual neuron emits spikes is tuned to be narrower, resulting in a more specific spike response. The larger number of encoding neurons also increases the number of input spikes, which are fed into the network. This also increases the overall network activity, resulting in potentially higher energy consumption.

As mentioned above, the number of 50,000 trainable variables and the first RF value set is used for all further analyses. At this network size, the reached classification accuracy of all variants starts to saturate, resulting in a diminishing rate of returns above this value. Although those variants using 100 RF input neurons tend to reach higher accuracies, we proceed with 40. By that, the input feature vectors have the same size for the Mel- and RF-based schemes. However, this again illustrates the ubiquitous tradeoff between network complexity, network activity, and classification performance.

Pruning

The influence of connection pruning on the networks' performance is examined in this experiment. Here, the iterative pruning method introduced in Section 4.2.3 is used. Multiple networks using the LSTM and RNN architectures are therefore trained, all starting as dense networks with 50,000 trainable parameters. Aiming at the target sparsity, the number of non-zero variables is gradually reduced during the training until 50% of the total epochs are reached. The second 50% of the total training steps are used to fine-tune the model at the target sparsity.

The result of this experiment is shown in Figure 5.10. In the upper plot, the achieved classification accuracy is shown in relation to the final sparsity of the network's connections. Above a sparsity of 80%, the reached classification accuracy starts to drop significantly for both network types. This sparsity corresponds to only 10,000 of the initial 50,000 neuronal connections being active. Interestingly, the SNN can maintain a better accuracy in the range between 80% and 90% sparsity, whereas the LSTM network already loses performance. This might hint at the advantages of sparsely connected SNNs. However, the best accuracies overall could be achieved by more densely connected LSTM networks.

The lower plot in Figure 5.10 shows the relation between network spike activity and the sparsity of the networks. Here, only the spiking RNN is depicted for apparent reasons. The network activity is shown in terms of spikes and SOPs. Spikes, thereby, correspond to the outgoing spike events emitted by a neuron. SOPs, on the other hand, refer to the incoming spikes, which trigger the computations at the receiving neuron. The distinction is made because depending on the connection density of the network, each emitted spike leads to a smaller or larger number of SOPs. In a sparse network, for example, only a few neurons are connected, which leads to fewer updates of efferent neurons. The plot shows a definite relation between network activity and connection density: the sparser a network is, the more spikes are emitted during inference. The opposite correlation is valid for the total number of SOPs. Due to the sparser connectivity, the emitted spikes are received by a smaller number of neurons. The fewer SOPs per spike introduced by a higher sparsity is thus more than enough to compensate for the larger number of total spikes.

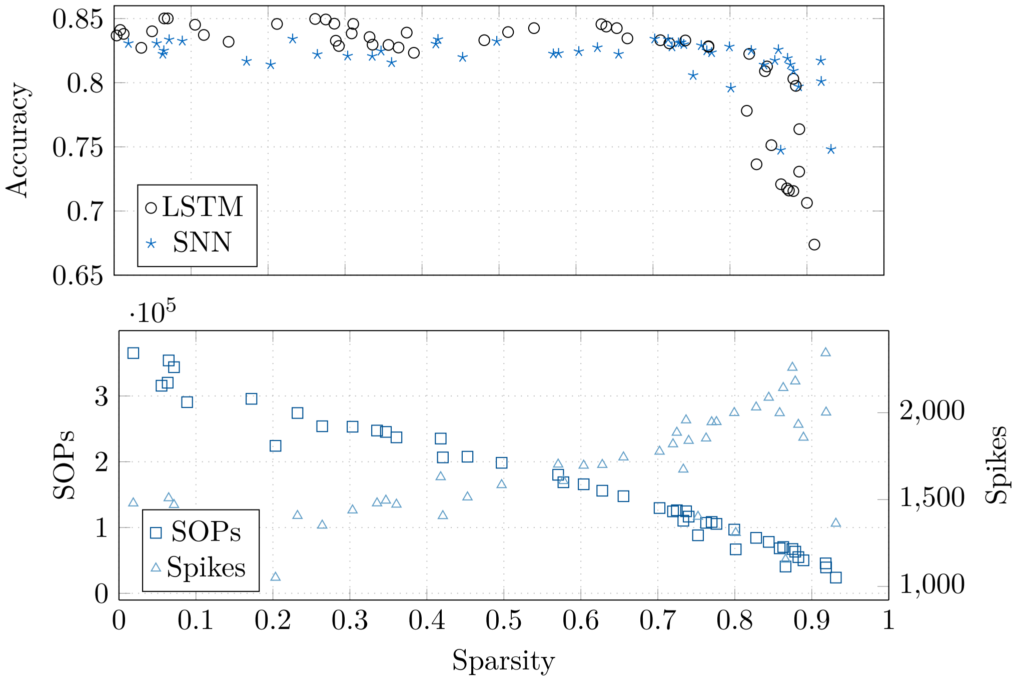


Figure 5.10: Network sparsity sweep. The classification performance versus sparsity of LSTMs and recurrently connected SNNs on RF-encoded input data is shown in the upper plot. The lower plot shows the network activity of the SNNs for the same sparsities. The number of spikes corresponds to the number of spikes, which have been emitted by the RF encoding and by the neurons of the hidden layer. The number of events corresponds to the number of SOPs, which have been triggered by the spikes. The number of SOPs decreases with increasing sparsity since less neurons have to be updated per emitted spike.

Regularization

To restrict the growing network activity in sparse networks, spike activity regularization is used. We, therefore, add the L1-norm for the spike activity of the hidden layer to the training loss as introduced in Section 4.2.2. The activity regularization factor β scales the loss contribution and thus controls its influence during the optimization.

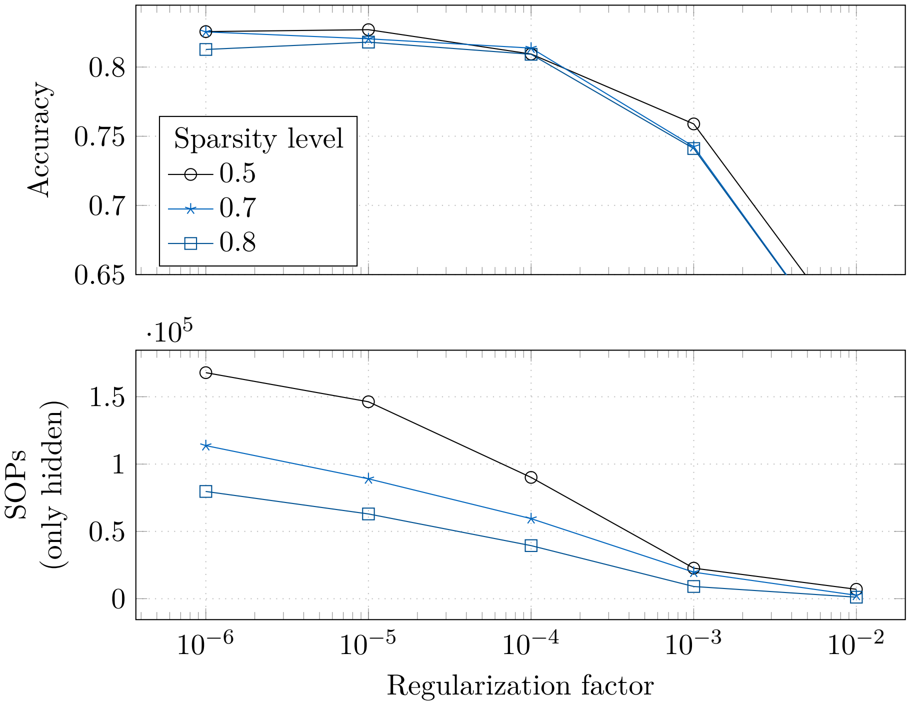


Figure 5.11: Network activity regularization sweep. The classification performance versus the regularization factor of the activity regularization is shown in the upper plot. The sweep is performed for different sparsity configurations. The lower plot shows the number of SOPs for the different regularization factors. Small regularization factors already influence the network activity while only marginally influencing the classification accuracy. At too large factors, the networks start to collapse, because the optimization favors decreasing the network activity over the reduction of the training error.

The results in Figure 5.11 show that with a growing activity regularization factor, far fewer spikes are emitted during inference. For low factors, this is even possible without sacrificing classification performance. It is thus recommendable to include a cautiously weighted activity penalization during the training phase. If the chosen factor is too large, the reached classification accuracy starts to lower, introducing the tunable tradeoff between accuracy and network activity.

This relation holds independently of the chosen target sparsity of the network. The differences in accuracy and spike activity remain in place: a sparse network will likely reach a lower classification performance while exchanging more spikes during inference.

Accordingly, the tradeoff between network density, classification accuracy, and spike activity is still present.

The training showed to be unstable in numerous instances because pruning and regularization are essentially working against each other. Therefore, the introduction of a schedule, which adapts the regularization factor during the training process, is beneficial. With the schedule, the regularization factor can be increased as soon as the pruning of the network is completed. This reduces the opposing effects of the two mechanisms. The instability often results in deadlocks of the network, in which no continuous connection between the network's input and output is given anymore. When this happens, the error gradients cannot be propagated through the network anymore, and the state of the network cannot be altered. This behavior heavily depends on the random initialization of the network weights. The corresponding phenomenon during pruning, in which all connections of a layer are pruned, is called layer-collapse [186]. So far, the state for which the network results in a deadlock cannot be predicted.

The regularization of large absolute values of connection weights (weight decay) did not alter the network performance. Due to the inherent dependencies between the optimization variables, weights, time constants, and threshold voltages, the regularization of one of the variables leads to the increase of another. Only if the specifications of the target (neuromorphic) hardware are available, a useful optimization can be achieved.

5.5 Discussion of the Results

5.5.1 Comparison to ANNs and the State of the Art

In this chapter, we demonstrated insights and relations of the sparse activity of SNNs on the example of speech recognition. Most evaluations are based on RF encoded features and simple SNNs consisting of a single population of recurrently connected neurons.

The accuracies reported in related publications vary slightly depending on the subset and version of the benchmark task being used. However, most works report values above 95%, which is far more than we achieved in our experiments. Reasons for that are the use of larger and more complex network architectures, more elaborate training techniques, and more expensive preprocessing. We intentionally used small and simple network structures to motivate the use of SNNs in restricted environments. Additionally, we dispensed with techniques like data augmentation [221] or lavish normalization approaches during feature generation [220]. Our goal was to identify relations and feasibility studies for using SNNs in real-world applications instead of attempting to beat existing benchmark records.

For this reason, we introduced baseline models based on ANNs to compare the spiking network approaches with. These ANNs were subject to the same training schedules and input data and therefore provided a more realistic comparative measurement. These networks, however, outperform the evaluated spiking network models on every type of input features, as shown in table 5.6. It is, therefore, reasonable to assume that SNN are less suitable for this application or even inferior to ANNs in general.

Our results show, and [172] come to a similar conclusion, that SNNs with two-dimensional convolutional layers can reach classification accuracies, which are close to those of ANNs. In both cases, the network architectures were similar and Mel-based input features were used. However, the commonalities of both approaches, their convolutional architecture, and the Mel-based input feature generation contradict our conceptions of efficient, event-based computation.

As shown in the detailed descriptions of the network architectures in tables 5.4 and 5.5, plain convolutional networks with two-dimensional convolutions result in huge networks, albeit having only a small number of tunable parameters. The weight sharing property causes the number of parallel synaptic connections to soar. Reducing the number of parameters to restrict the network size, on the other hand, limits the ability of the network to generalize well. Additionally, by using solely two-dimensional convolutions, the temporal characteristics of the spiking neurons are not really used. Thus, the inherent potential is not leveraged.

As discussed in [52], resonating neurons may be a more energy-efficient alternative to the digital processing chain used in modern speech recognition systems. Using these neurons, analog-to-digital converters, digital filters, fast Fourier transform blocks, and the MFCC feature generation can be omitted. We demonstrated that these neurons generate feature-rich spike trains that the following network structures can analyze. The set of hyperparameters such as the number of resonating neurons, the choice of their resonance frequencies, or the threshold adaption characteristic allows space for improvement and the adaption to other applications. However, the energy efficiency of this method using specialized electrical circuits remains to be proven.

5.5.2 Energy and Complexity Considerations

Our experiments show the contending relation between the network activity, the network complexity, and the reached classification accuracy. Especially using the pruning and regularization techniques, networks can be adapted to fit the targeted hardware optimally. The complexity of the network can thereby be influenced by the number of neurons, parameters, and synaptic connections. The architectures, which were evaluated in this section, are listed in table 5.7. Depending on the neuromorphic realization, many solutions arise, which all comprise specific trade-offs.

In Section 4.3, we introduced Davidson and Furber's approximation of the energy consumption of ANNs and SNNs in purely digital implementations [23]. The authors state that with equivalent architectures, each spiking neuron can emit a maximum of 1.72 spikes during the inference of the same example to undercut the energy consumed by the ANN. The approximation is not entirely suited to compare networks with different architectures, but it might still be valid for a ballpark figure. The scaling effects by parallel execution and vectorization, application of non-linearities, and the propagation of spikes are not considered in the comparison.

The evaluated LSTM executes nearly $5 \cdot 10^6$ MAC operations during the 100 time steps of the inference of one sample (see table 5.4). The recurrent SNN on the other hand totals $\approx 0.33 \cdot 10^6$ SOPs (see table 5.7 and table A.1 for layer-wise activity numbers). Even without the merits of simpler calculations, the sheer number of executions already highlights the potentially much higher efficiency of the SNN. At the same time, models

Table 5.7: Complexities of evaluated SNN architectures. All networks comprise 50k initial trainable parameters. The sparse recurrently connected SNN has the same number of initial parameters, however, most of them are pruned during training. The networks implementing convolution operations have significantly more synapses due to the weight sharing properties of convolution layers.

Model	**Neurons**	**Synapses**	**SOPs**
DNN	412	50k	363,440
RNN	212	50k	335,868
RNN (80% sparse)	212	10k	105,254
1D-CNN	709	52k	198,346
1D-RCNN	708	69k	256,287
2D-CNN	3,540	1M	4,460,040

given in other works, which use a comparable number of operations, provide worse performances [8]. An even higher potential reduction in connection complexity and network activity is given when sparse networks are used. Table 5.7 lists one exemplary sparse RNN, which is 80% sparse, resulting in only 10^4 remaining connections between the neurons with only minor losses in the classification accuracy.

A further important difference arises when we compare the number of synaptic or MAC operations, which are induced only by the input layer. On average, the spike train generated by the RF neuron layer comprises 260 spikes for one spoken word. In comparison, the inputs based on MFCC and log-Mel features deliver 40 distinct values in each time step, resulting in 4000 values throughout a single word utterance. Additionally, to generate the log-Mel features, the speech signal first has to be transformed into the frequency domain using algorithms like the FFT. This requires thousands of additional operations. The large discrepancy between the required numbers of operations alone motivates the use of event-based computation schemes. As for RF neurons, however, we do not know whether the actual embedded implementations advance these benefits yet. It has to compete with the highly-optimized implementations of digital signal processing modules like filters and FFT blocks.

6 Radar-Based Hand Gesture Recognition

In chapter 5, we found that given a suitable spike representation of the input, a recurrently connected SNN can reach good performances while requiring only few resources compared to convolutional structures. In this chapter, we demonstrate this on a second application: hand gesture recognition[1]. Radar-based hand gesture recognition is a well-suited example application for spatio-temporal sequence classification.

6.1 Motivation (based on [55])

In many fields, human-computer interaction evolves towards dynamic interactions that are more intuitive for humans. One possible natural method for humans is using hand gestures, which are sensed and recognized by machines. The applications for this type of interaction are various: smart home, car entertainment, mobile phones, robot control, or even interactive display panels in smart cities.

The gestures are commonly detected using cameras [223], [224]. However, multiple other approaches have been proposed. Human attached sensors such as an armband to detect muscle activity showed successful classification of 6 different classes [225]. Gesture recognition can also be extended to full-body movements, as shown by [226], where an ANN classifies the micro-Doppler signatures of sonar sensors.

Radar-based gesture recognition provides many advantages over other solutions, like the independence of lighting, atmospheric conditions, and the inherent privacy. Radar systems can detect the range, velocity, and angle of arrival of nearby targets independent of environmental conditions. Therefore, a variety of classification algorithms based on ANNs have been proposed [227]–[232].

For all applications, a high classification accuracy as well as a low-energy consumption are of major importance. Especially for mobile devices, a low-energy consumption of the sensor itself and the attached signal processing is crucial for a long battery life. Therefore, SNNs are examined as potential solutions for this task.

6.2 Background

In the following, we provide the background for radar-based gesture recognition. Therefore, a brief introduction of the measurement principle of radar is given. Subsequently, we present an overview of related works within this topic.

[1] This approach has been previously published in [55].

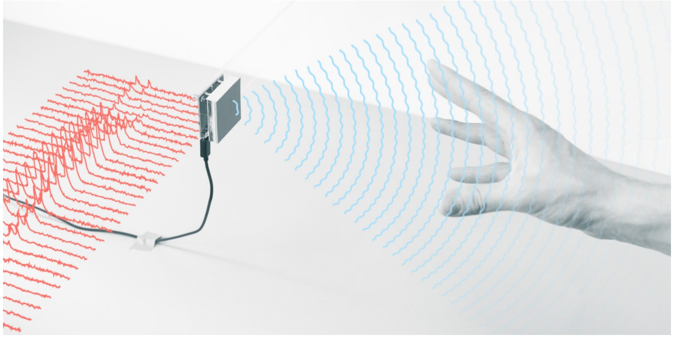


Figure 6.1: Radar-based hand gesture recognition. (Figure from [231])

6.2.1 Measurement Principle

Radar (*Ra*dio *D*etection *a*nd *R*anging) is a technology using electromagnetic waves in millimeter-wave range to measure the relative velocity and distance between sensor and target. The spatial position of the target can be further defined using multi-antenna designs. The basic principle is to measure the reflection of a previously emitted wave. Depending on the specific method, the time between emission and reception of the measurement signal or its frequency is measured. Radar is used in large-scale applications like weather forecasting and land surveying, aircraft and ship surveillance, but also in smaller fields like automotive applications and the usage in handheld devices. The benefits of this technology are its scalability, low volume production costs, reasonable package size, and robustness to environmental conditions like dust, water, and different lighting conditions.

The most simple measurement principle is the pulse-Doppler radar, which uses the time of flight measurement of the transmitted signal and the frequency shift by the reflection on the target due to the Doppler effect. In modern small-scale radar applications, often Frequency Modulated Continuous Wave (FMCW) or chirp signals are used because of the best possible utilization of the signal power, used bandwidth, and measurement time [233]. Here, the frequency of the continuously transmitted signal is varied to measure the target's distance and velocity. The transmitted signal of the FMCW radar is

$$x_{\text{transmit}}(t) = \cos\left(2\pi f_c t + \pi \frac{B}{T_c} t^2\right).$$

The frequency of the signal is thereby linearly increased during the chirp duration T_c starting from the base frequency f_c by the sweep bandwidth B, as shown in Figure 6.2.

The instantaneous frequency of the transmitted and one reflected signal is depicted in Figure 6.2. The waveform of the received signal

$$y_{\text{receive}}(t) = \alpha \cos\left(2\pi f_c(t - t_d) + \pi \frac{B}{T_c}(t - t_d)^2\right)$$

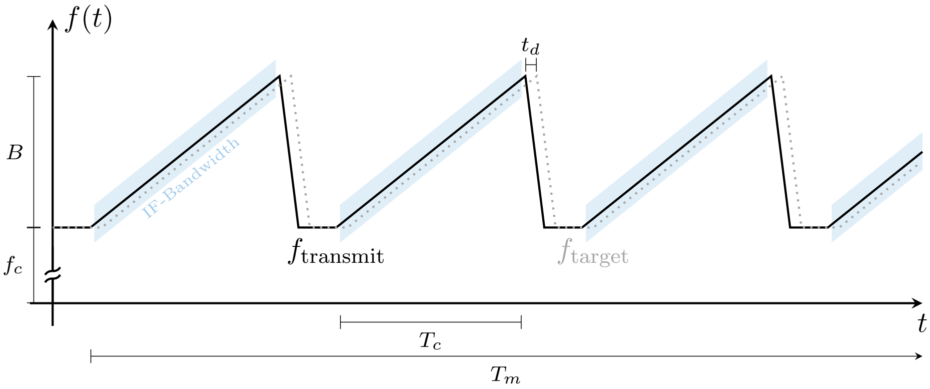


Figure 6.2: Schematic chirp sequence of an FMCW radar system. The transmit signal's frequency is modulated over time. Starting at the carrier frequency f_c, the frequency is gradually increased by up to B during the chirp interval T_c. The received signal is temporally shifted by t_d but otherwise corresponds to the transmitted signal. This temporal shift is proportional to the target's distance and can be obtained by determining the difference frequency between send and received signal. Over the course of a measurement interval T_m, the phase change of the received signal can be determined to calculate the relative velocity between sender and target.

is scaled by α due to transmission path losses and delayed by t_d due to the traveling time of the signal. After mixing the transmitted and the received signal and filtering the high-frequency part, the Intermediate Frequency (IF) signal remains as

$$y_{\text{IF}}(t) = \cos\left(\phi_0 - 2\pi \frac{B}{T_c} t_d t\right).$$

Since the chirp duration is very short, a frequency shift during the sweep due to Doppler effect is small and can be neglected. With that, a linear correspondence between the IF signal's frequency and the range of the measured object is given. Applying a first dimension FFT on the IF signal exposes the received frequencies and with that distances to the detected objects. The values of the discrete spectrum represent fixed distances, which are therefore called range bins. To determine the relative velocity between the measured objects and the sensor, the second dimension FFT can be applied to evaluate the phase change of the IF signal between multiple chirps. The two-dimensional matrix as a result of the two Fourier transforms is called Range-Doppler Matrix (RDM). The matrix contains the absolute values of the second Fourier transform and thus includes the corresponding velocity values for each distance bin.

6.2.2 Related Work (based on [55])

Most gesture recognition systems use classical signal processing algorithms to generate RDMs and use subsequent neural networks to classify the sensed gesture. Often, the networks are separated into two parts: spatial feature generation and temporal sequence recognition. The former part is achieved using convolutional layer structures to extract meaningful features from the incoming range-Doppler images. The latter part then analyzes sequences of extracted features to recognize the actual movement of the hand and fingers. These temporal dependencies are evaluated using either temporal convolutions [230] or LSTM layers [227], [229], [232]. Alternative CNN-based solutions omit the second Fourier transform to generate the RDMs. Instead, they evaluate the temporal development of the distance metric using multiple convolutional layers [234], or they only use the temporal development of the velocity profile [228]. A comprehensive review of current approaches is given by Ahmed, Kallu, Ahmed, and Cho [235].

Recently, four works have been published, which solve the gesture recognition task using SNNs, too [173], [236]–[238]. The simultaneous publications underline the significance and the topicality of the research.

The authors of [236] have shown that SNN-based radar processing performs well on the recognition of whole-body gestures. There, the authors use convolutional structures to extract features from the spectral input data. Additionally, they use the bio-inspired STDP learning rule to adapt the network weights based on the relative timing between spikes. However, they use a comparatively large CNNs to classify large body movements in a small dataset.

The authors of the approaches introduced in [237] and [173] base their evaluations of the hand gesture recognition task on the dataset provided by [229], thus enabling a meaningful comparison between the different SNN-based and classical approaches.

The network proposed in [237] is a hybrid structure consisting of a spike-based Liquid State Machine (LSM) and a conventional Support Vector Machine (SVM). The LSM is a reservoir of excitatory and inhibitory neurons, which are randomly connected and will not be adapted during the learning process. Learning takes place in the readout layer, which maps the activity of the reservoir onto a classification output. The authors implement three different readout classifiers for this purpose, namely logistic regression, random forest, and a SVM. However, they show that the SVM classifier performs best. The input of the network are binarized RDMs using a fixed threshold, comparable to the methods introduced in Section 3.2.3. The authors show that a network with a LSM consisting of 153 neurons already gives a high classification accuracy. From this point, the accuracy starts to saturate when further increasing the number of neurons being part of the reservoir. Most performance metrics are reported for a network with 460 neurons. The connections between the input pixels and the reservoir and the connections within the reservoir are sparse. However, two details within the work's evaluation part are questionable and lead to non-realistic performance: (1) The authors use a random 50/50 split to divide the available data into training and test data. With that, gestures performed by a single person are present in both sets. This is highly unrealistic since most users use their devices without being part of the product's development process. The more realistic Leave-One-Subject-Out (LOSO) cross-validation, using all but one subject for training and using the remaining subject for testing, is a more reasonable approach and is used only once. (2) The authors distinguish between normalized and variable sequence length, for which only the variable sequence length is a viable choice.

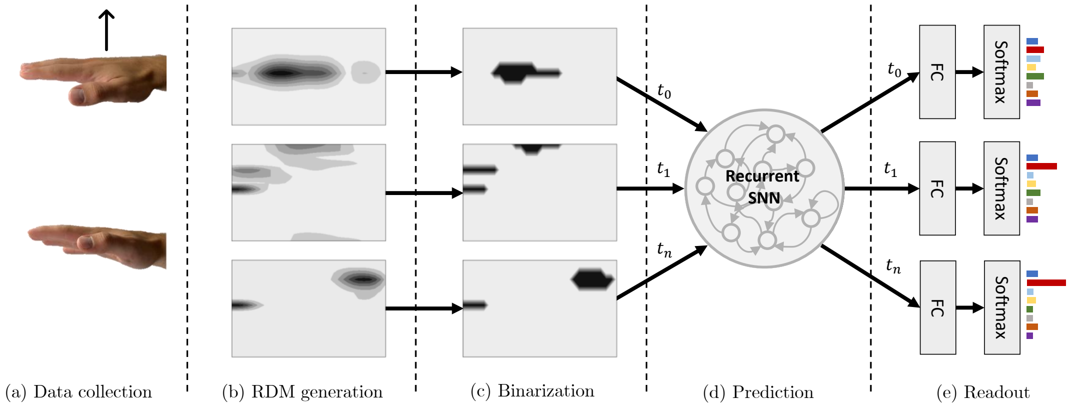


Figure 6.3: Network architecture for the hand gesture recognition task. The gesture data is collected using radar sensors (a). Subsequently, the image-like RDMs are generated using classical signal processing approaches (b). The RDMs are binarized using different techniques (c) to provide spike representations for the recurrently connected SNN to be recognized (d). Non-spiking integrator neurons, one for each class, indicate the probability of a detected gesture (e). (Figure previously published in [55])

Concerning the normalized length, each sequence is normalized to a fixed length, thus compressing or stretching the sequence in the time domain.

The network proposed in [173] consists of layers of ALIF neurons, which extend the LIF neuron model by an adaptive threshold. The recurrent connections within the layers and the feedforward connections between the layers are trained using pseudo-gradient-based BPTT. The authors propose the Multi-Gaussian surrogate gradient, which includes negative slopes as inspired by the Exponential Linear Unit (ELU) activation function. The network consists of two hidden layers with 512 neurons each and a non-spiking output layer. The first layer thereby does not include any recurrent connections. The encoding of the input RDMs is not specified further.

6.3 Setup (based on [55])

The structure of the evaluated architecture is shown in Figure 6.3. The raw data, provided by two different datasets, is transformed into RDMs using standard radar signal processing techniques. The resulting image-like maps are encoded into binary events. These events are then fed into the SNN to classify the presented gesture. More detailed descriptions of the individual steps are given in the following.

6.3.1 Datasets

The datasets used in the experiments consist of 11 different gestures, which are specifically designed to assess the performance of gesture recognition systems [229]. The gestures comprise small finger movements (*pinch pinky*, *pinch index finger*), larger movements of

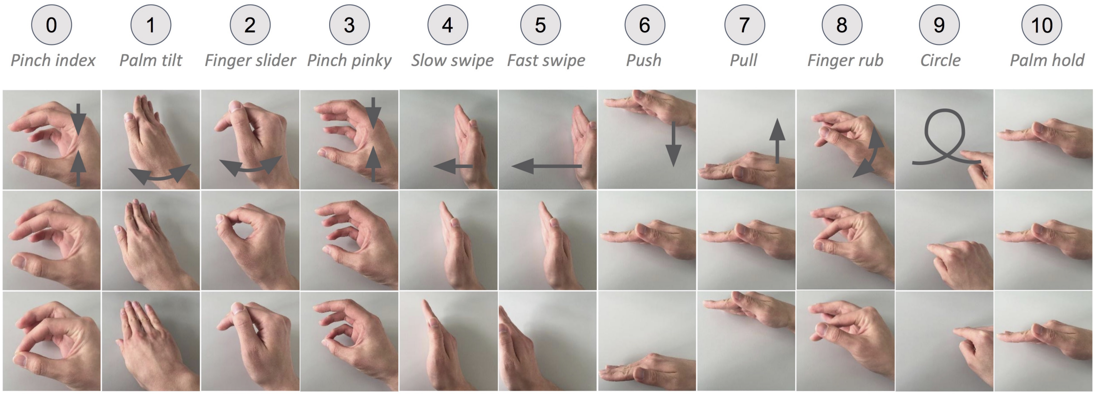


Figure 6.4: Hand gestures contained in the Interacting with Soli dataset. The arrows in the pictures of the first row indicate the movements of the fingers or the hand to perform the gestures. The second and third row show the start and finish positions of each respective gesture. (Figure from [229])

the whole hand (*push*, *pull*), and gestures with different speeds of movement (*slow swipe* and *fast swipe*). Illustrations of the gestures are depicted in Figure 6.4.

The initial version of the dataset [229] uses Google's project Soli sensor [231]. A second independent version uses Acconeer's A1 RADAR sensors to record the same set of base gestures and an additional *no hand* gesture [230]. Although both datasets consist of the same set of gestures, the classification accuracies reached during the evaluation cannot be compared directly. The two sensors produce quite different data streams due to their different designs. Additionally, the datasets differ significantly in the size of the available training data. The relevant properties of the datasets are summarized in table 6.1. We apply the proposed networks to both datasets to achieve the best comparability of the specific algorithms.

6.3.2 Preprocessing and Encoding

For both datasets, we use zero padding to reach a constant number of frames for each gesture for batch processing. Therefore, empty RDMs are added at the end of each recording. The approaches in [229], [237] use temporal interpolation to achieve constant sequence lengths. However, temporal relationships are altered as this stretches or compresses the sequence.

The TinyRadarNN dataset is available in a raw format, leaving the freedom to choose the parameters to generate the RDMs. The chirps in the dataset provide information about nearly 500 range bins with a resolution of below a millimeter. To reduce the data rate and to prevent overfitting, each chirp is decimated using average pooling with a kernel size of four. Subsequently, the Fourier transform is applied to generate the velocity axis of the RDMs. The window size of the Fourier transform is chosen to be 16 chirps to avoid smearing on the range axis due to too long evaluation time windows. To reduce

Table 6.1: Dataset parameters from Interacting with Soli and TinyRadarNN. The Interacting with Soli dataset was recorded using a Soli sensor with four input channels. The data is available as in the form of RDM sequences. Thus, always 32 chirps are grouped together in one RDM. The TinyRadarNN dataset was recorded using two separate A1 RADAR sensors. The data is available as frequency-transformed raw data of the individual chirps.

Parameter	**Interacting with Soli** [229]	**TinyRadarNN** [230]
N_{channels}	4	2
Chirp frequency	$32 \cdot 40$ Hz	160 Hz
Range bins	32	492
Velocity bins	32	32
Recording length	≈ 1 s	≤ 3 s
N_{persons}	10	26
Recordings per gesture	$10 \cdot 25$ (25 recordings per person)	$26 \cdot 35$
Total recordings	2750	10010

the size of the input data further and to increase the generalization, an additional max pooling is performed on the range dimension of the generated RDMs.

The used radar systems have multiple receive channels; thus, multiple RDMs are generated and can be used to determine the angle of the perceived targets additionally. The maps are either fed directly into the network by converting the values of the RDMs directly to the input currents of the spiking neurons or encoded into spikes to match the information exchange format of the binary activated networks (see Section 3.2.3). Different approaches are used concurrently to evaluate the influences of the different binarization techniques: mean binarization, α-quantile binarization, and three-level binarization (ternarization).

In all cases, the binarization is performed for each input channel individually to prevent the dominance of a single channel due to possibly different characteristics of the channels. The resulting channel-wise binary encoded RDMs are stacked and form a 4-dimensional tensor of the format [time, range, velocity, channel]. Accordingly, the encoding scheme converts the amplitude values in the RDMs along the range and Doppler axis $v_{r,d}$ for each time step and channel into a binary form $z_{\text{input},r,d}$.

6.3.3 Evaluation Metrics

In [229], the authors differentiate between the per-sequence and the per-frame accuracy. In the former accuracy metric, the information of the completed gesture is present at the time of evaluation and classification. In the latter case, only the instantaneous RDM is presented without the specific information of the start and the end of the sequence. Since the definition of a frame is ambiguous [230], we report the per-sequence accuracy only.

The reported accuracies are based on the multi-user LOSO cross-validation tests. There, the networks are trained using the data of all but one subject and are evaluated on the unseen data. This is repeated for every subject, and the resulting accuracy values are averaged, hence the cross-validation. This is the most realistic scenario, as it is implausible that each user's gesture data is also part of the training set.

6.3.4 Network Architectures

In this experimental setup, we evaluate whether the findings of chapter 5 are applicable to other applications. Therefore, recurrently connected SNNs are used because of their architectural simplicity and their convincing performance in the speech recognition task. The resulting classification accuracies are compared with those of other works [229], [230], [237].

In this application, the input consists of sequences of two-dimensional data frames with multiple channels each. Because the recurrently connected SNN does not extract spatial information like a CNN would do, the input is flattened. With that, the input RDM sequences are reshaped into a sequence of feature vectors. The recurrently connected population of LIF neurons in the hidden layer is fully connected to the input feature vector at every time step. Similar to the speech recognition application in chapter 5, the output consists of non-spiking integrators, which are evaluated at the end of each sequence.

The SNNs are simulated in discrete time steps. The network is updated with each presentation of a new input frame without intermediate update steps. The network is thus updated as often as there are frames in the sequence.

6.4 Evaluation

6.4.1 Classification Performance

The classification performances reached during the LOSO cross-validation tests are shown in table 6.2. The performances of the networks with different input encoding variants are comparable with those of other approaches given in the literature. The results show that in this application, the binarization of the input does not reduce the performance of the binary activated SNNs. In fact, using the binarized input RDM sequences reached even higher accuracies than the raw inputs. This supports the hypothesis that the notion of the nearest range and velocity bins holds sufficient information. The individual amplitude values of each bin are thus not needed.

The confusion matrices shown in Figure 6.5 illustrate the links between the false classifications of one exemplary cross-validation test for each dataset. Especially, small finger movements – *pinch index* (0) and *pinch pinky* (3) – are easily mistaken. Also, a steady hand (10) is often confused with gestures involving finger movements, as in both cases, most of the hand does not move at all. The distinction between a fast and slow swipe gesture (4 and 5) is also often misclassified, suggesting a too coarse time resolution or too vague data in the training set due to different interpretations of *fast* and *slow* by the recorded persons. Most gestures, however, can be recognized with low to no error.

Table 6.2: Classification performance of different approaches for the radar gesture recognition task. Wang et al. and Scherer et al. are also the provider of the respective datasets. Both solve the recognition task using ANNs. Tsang et al. use a hybrid network consisting of both spiking and non-spiking neurons.

	Interacting with Soli	**TinyRadarNN**
Wang et al. (2016) [229]	88.27	-
Tsang et al. (2021) [237]	91.40	-
Scherer et al. (2021) [230]	-	78.85
Current injection (raw)	86.24	79.15
Mean binarization	88.20	80.31
α=0.10 ternarization	88.17	79.33
α=0.10 max binarization	86.71	79.02
α=0.05 max binarization	87.40	76.43

6.4.2 Ablation Study

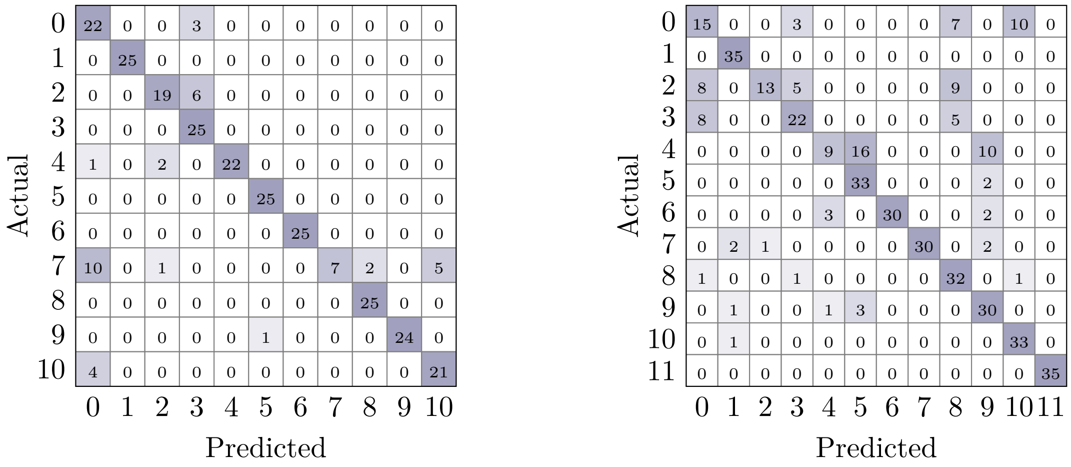
Because the applied methods and obtained results are similar for both datasets, we limit the following evaluations on the TinyRadarNN dataset due to its larger number of samples. To reduce the computational overhead, 5-fold cross-validation is used instead of LOSO cross-validation. Thus, the test set contains the gestures of five individuals each, instead of only one in the case of LOSO. This does not change the qualitative results of the experiments, as the train and test groups are still large, and we are interested in the scaling properties of individual design aspects. However, the quantitative accuracy measures cannot directly be compared to those reported in table 6.2.

Network Size

Similar to the speech recognition task's evaluation, the necessary network size is first determined. Figure 6.6 shows the performances reached during the evaluation of the gestures of one data set. Starting at 10^5 trainable variables, the accuracy begins to saturate. We, therefore, use networks with 10^5, which corresponds to 100 neurons in the hidden layer, which are fully connected to the input and recurrently connected within the population. During Section 6.4.2 we will see that, similar to the speech recognition task, only a fraction of the dense connections are necessary to reach the same performances. Thus, the number of neurons determines the reachable accuracy.

Input Encoding (based on [55])

The influence of the encoding scheme on the classification performance is evaluated in the next set of experiments. As shown in table 6.2, the use of all five encoding schemes leads to performances that are comparable with those reached by related works. The mean binarization encoding scheme leads to the best results on both datasets. Figure 6.7 shows the exemplary evaluation of the finger slider gesture. The spike activity patterns for the



	0	1	2	3	4	5	6	7	8	9	10
0	22	0	0	3	0	0	0	0	0	0	0
1	0	25	0	0	0	0	0	0	0	0	0
2	0	0	19	6	0	0	0	0	0	0	0
3	0	0	0	25	0	0	0	0	0	0	0
4	1	0	2	0	22	0	0	0	0	0	0
5	0	0	0	0	0	25	0	0	0	0	0
6	0	0	0	0	0	0	25	0	0	0	0
7	10	0	1	0	0	0	0	7	2	0	5
8	0	0	0	0	0	0	0	0	25	0	0
9	0	0	0	0	0	1	0	0	0	24	0
10	4	0	0	0	0	0	0	0	0	0	21

(a) Interacting with Soli

	0	1	2	3	4	5	6	7	8	9	10	11
0	15	0	0	3	0	0	0	0	7	0	10	0
1	0	35	0	0	0	0	0	0	0	0	0	0
2	8	0	13	5	0	0	0	0	9	0	0	0
3	8	0	0	22	0	0	0	0	5	0	0	0
4	0	0	0	0	9	16	0	0	0	10	0	0
5	0	0	0	0	0	33	0	0	0	2	0	0
6	0	0	0	0	3	0	30	0	0	2	0	0
7	0	2	1	0	0	0	0	30	0	2	0	0
8	1	0	0	1	0	0	0	0	32	0	1	0
9	0	1	0	0	1	3	0	0	0	30	0	0
10	0	1	0	0	0	0	0	0	0	0	33	0
11	0	0	0	0	0	0	0	0	0	0	0	35

(b) TinyRadarNN

Figure 6.5: Confusion matrices for the two radar gesture datasets. In both cases, recurrently connected SNN with 10^5 trainable variables are used. The input RDMs are encoded using mean binarization. The gestures comprise in ascending order: *pinch index, palm tilt, finger slider, pinch pinky, slow swipe, fast swipe, push, pull, finger rub, circle, palm hold, no hand.* The Interacting with Soli dataset in (a) does not include the *no hand* gesture.

different encoding schemes, as well as the output of the hidden and output layers, are depicted. The high activity in the middle segments of the encoded gestures corresponds to the static hand in front of the sensor. The activity outside this area is likely to be part of the finger movement within the distance-velocity coordinate system of the encoded RDMs.

T-Distributed Stochastic Neighbor Embedding (TSNE) visualization plots [239] are handy tools to graphically analyze nonlinear relationships within multidimensional data, where the low-dimensional representation projects the neighboring probability of the higher dimensional data clusters. In our case, we use it to get an understanding of the separability of the raw RDMs as well as the spike train representations of the input binarization methods and the activity of the hidden layer. Figure 6.8 shows the clustering of all classes with color indications, where each dot corresponds to a gesture in the test set of the TinyRadarNN dataset.

The first finding here is that the TSNE clustering algorithm is, in fact, able to separate the classes based on the spike trains of the inputs and the hidden layer. The algorithm itself separates the data points iteratively in an unsupervised manner. The colors representing the different classes are added later for a better visual distinction. The visualizations show that the different gestures performed by a single person can be separated using the algorithm. Though, a perfect separability of the raw input data is not necessarily given. The binarization encoding does not affect the clustering much, thus suggesting that no relevant information is lost. Some clusters are hard to distinguish in both the raw and the binarized format. Note that the distance between data points in

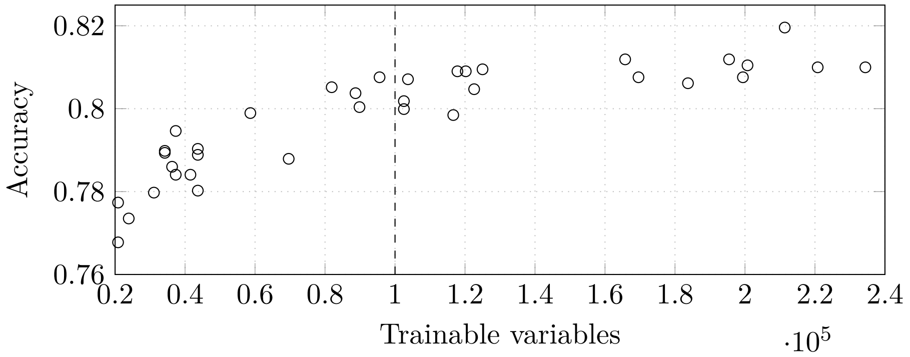


Figure 6.6: Network size sweep. Classification performance versus network size of recurrent SNNs on the TinyRadarNN dataset. The stroked vertical line marks the number of 100,000 trainable variables. After that point the classification accuracy starts to saturate when further increasing the network size.

Table 6.3: Synaptic operations in densely connected networks using different binarization approaches. The large disparity of the SOPs in the input layer are caused by the different sparseness levels of the encodings. When feeding the raw stream into the neurons, each neuron in the input layer has to be updated in every time step. Whereas in the sparse encoding schemes, only a subset of the inputs need to be updated.

Encoding	**SOPs Input**	**SOPs Hidden**
Raw	2,400,000	44,920
Mean bin.	356,600	48,320
α=0.10 tern.	147,800	47,280
α=0.10 max bin.	147,800	49,900
α=0.05 max bin.	79,800	49,000

TSNE is only a representation of clusters; it is not suited to make quantitative statements. The spike activity of the hidden layer shows a clear separation between most of the clusters, highlighting the capabilities of the recurrently connected neuron population to extract temporal patterns from the input signal. The two blue shaded clusters, which are not separable in all TSNE plots, correspond to the gestures *pinch index* and *pinch pinky*. As already seen in the confusion matrix (indices 7 and 8), the distinction between those two gestures is, in most cases, not possible for the network.

Besides the differences in the reached classification accuracy, the different input encoding schemes also result in vastly different numbers of synaptic events. Table 6.3 shows the average number of SOPs induced by the input and the spikes emitted by the hidden layer for the different evaluated encoding schemes. Qualitatively, these differences are also visible in the exemplary visualization in Figure 6.7.

In “raw” encoding, each pixel of the input RDMs has to be multiplied by its associated weight and to be accumulated by the neurons in the hidden layer. For all frames within

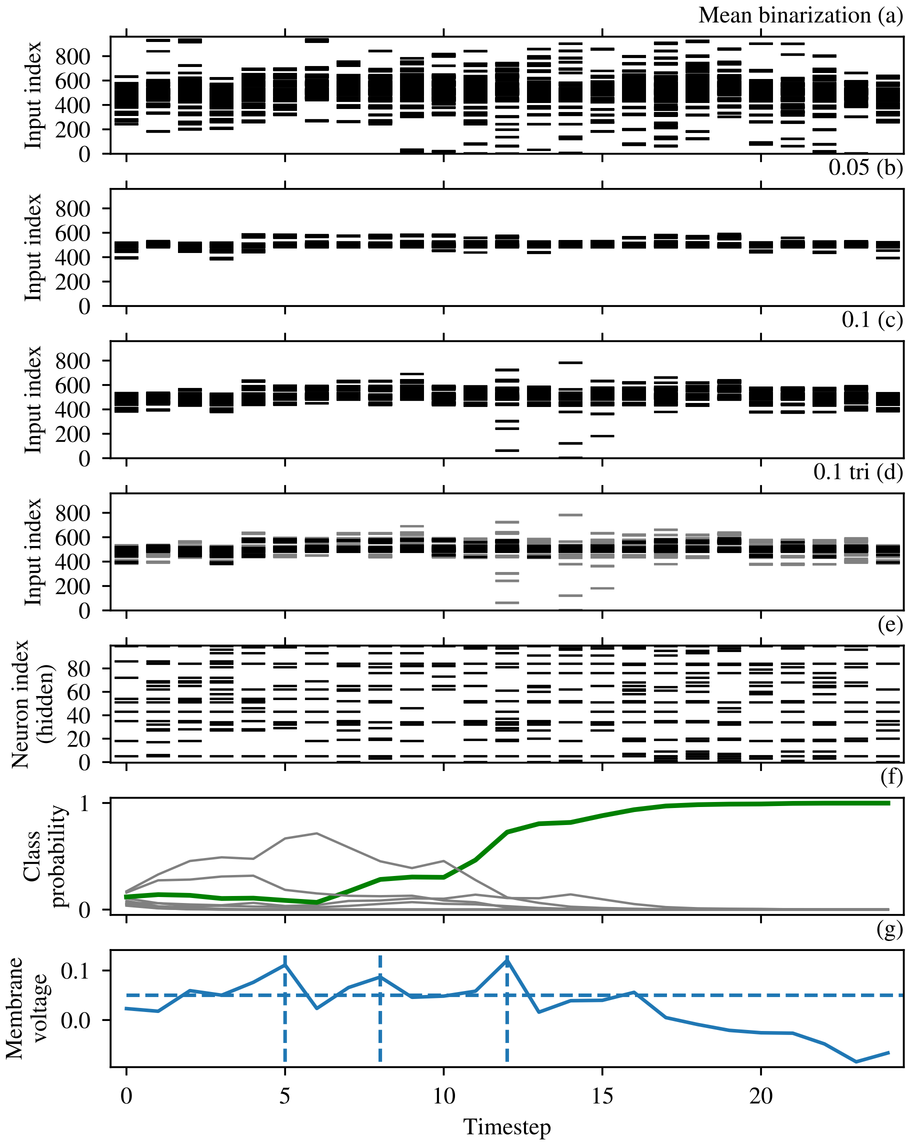
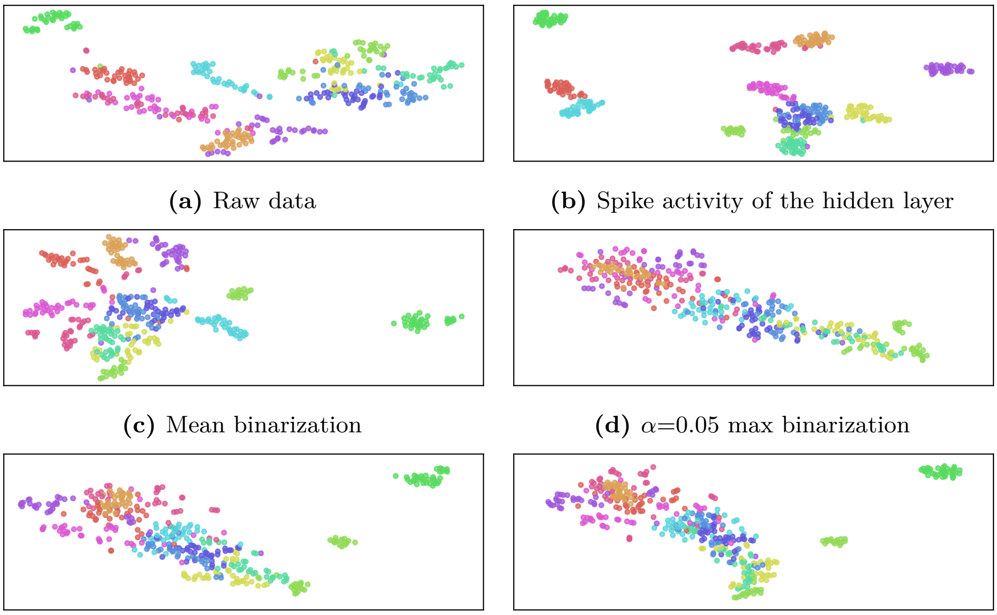


Figure 6.7: Exemplary evaluation of the SNN's inference of a gesture. (a-d) show the different binarization approaches on the RDM sequence of the gesture *finger slider* from the Interacting with Soli dataset. The approaches show notable differences in the average number of spikes, which are active to encode each frame. The activity of the hidden layer, which was fed with the spike train resulting from the mean binarization in (a), is shown in (e). (f) depicts the course of the normalized membrane voltages of the integrating output neurons. The voltage course of the neuron, which corresponds to the correct class, is highlighted in green. The plot at the bottom (g) shows the evolution of the membrane voltage of one arbitrarily chosen neuron from the hidden layer in (e). The stroked horizontal line visualizes the threshold voltage. The stroked vertical lines illustrate the spike events.



(a) Raw data

(b) Spike activity of the hidden layer

(c) Mean binarization

(d) α=0.05 max binarization

(e) α=0.10 max binarization

(f) 3-level binarization

Figure 6.8: TSNE visualization plots of the gesture data. (a) depicts the clustering of the unaltered RDM sequences. The clustering of the spike activity of the hidden layer is shown in (b). This hidden layer is fed with RDM sequences using mean binarization. (c-f) show the TSNE visualization for the binarized RDM sequences. In all cases, most information is preserved during the binarization, enabling the SNN to extract the relevant patterns for the classification. The binarization techniques in (d-f), however lead to the merging of some clusters, which is also notable in the confusion matrices in Figure 6.5.

one sequence, this sums up to 2.4 million MAC operations. Obviously, many of these operations do not necessarily contribute to the inferred classification, which will be shown during pruning (see Section 6.4.2). The binarized inputs result in much fewer operations additionally to being solely add operations. However, the encoding of the input does, of course, also introduce additional overhead, which we do not consider here. The least SOPs are induced by the α=0.05 quantile binarization. The number of operations is even smaller than 0.05× the number of operations of the raw input stream. This is because there is not even enough activity to fill the available spike contingent in some frames.

The SOPs induced by the hidden layer express themselves in the sum of the operations by the recurrent connections and the connections to the output layer. Interestingly, the number of operations is the lowest for the raw input type. In the next section, we will see that this relation stays the same during the pruning of the networks.

Pruning

In this experiment, the iterative pruning method introduced in Section 4.2.3 is used again. As shown in Figure 6.9, we train and evaluate networks with different target sparsity levels for the different encoding schemes. The results of this experiment support the findings of the relationship between accuracy, sparsity, and network activity.

In the upper plot of Figure 6.9, the relation between the network sparsity and the reached classification accuracy for the different encoding schemes is shown. For values up to 90% sparsity, the reached performances are constant and resemble the results reported in table 6.2. However, because we use an 80/20 split in this experiment instead of LOSO cross-validation, the numbers are not quite the same. Networks that are sparser than 90% result in increasingly worse accuracy because, with higher sparsity levels, important connections are pruned, too.

The relations for the number of spikes and SOPs are depicted separately by the hidden layer and input. The two middle plots show the SOPs and spike emissions of the hidden layer, respectively. Similar to the results of the speech recognition application in Section 5.4, the number of spikes rises with increasing connection sparsity. The number of synaptic events decreases simultaneously. Again, this is due to the outweighing influence of the fewer number of afferent neurons, for which the spikes induce SOPs. Above sparsity levels of 95%, there is a sudden decline of emitted spikes, which matches the region where the overall reached accuracy breaks down. Accordingly, the network is not able to work properly anymore at these sparsity levels. The choice of the encoding scheme slightly influences the activity of the hidden population. The injection of the raw RDM values into the layer leads to the lowest activity in the hidden layer. On the other side, the quantile-based binarization schemes lead to more spikes being exchanged. The 3-level and mean binarization schemes lie in between.

However, the differences in the induced spike activity in the hidden layer are small against those induced by the input layer itself. The SOPs induced by the input layer are shown in the lower plot of Figure 6.9. The different binarized inputs are sparse in space and time by design, thus triggering much less SOPs. The injection of the raw RDMs, in comparison, triggers operations for every input pixel at every frame. The introduced connection sparsity, therefore, influences the raw RDM input most.

Regularization

The adaption of the scaling factor of the activity regularization loss during the speech recognition experiment in Section 5.4 yielded a degradation of the reached classification accuracy, followed by a sudden drop as soon as the impact of the regularization term surpassed that of the training loss. At the same time, with a growing regularization scaling factor, the synaptic events within the hidden layer decreased. The evaluation of the gesture recognition task results in similar behavior.

Figure 6.10 shows the reached classification accuracy and the number of synaptic events of the hidden layer for the different encoding schemes of the RDMs at a range of regularization factors. The results are shown for a target sparsity of 70%; however,

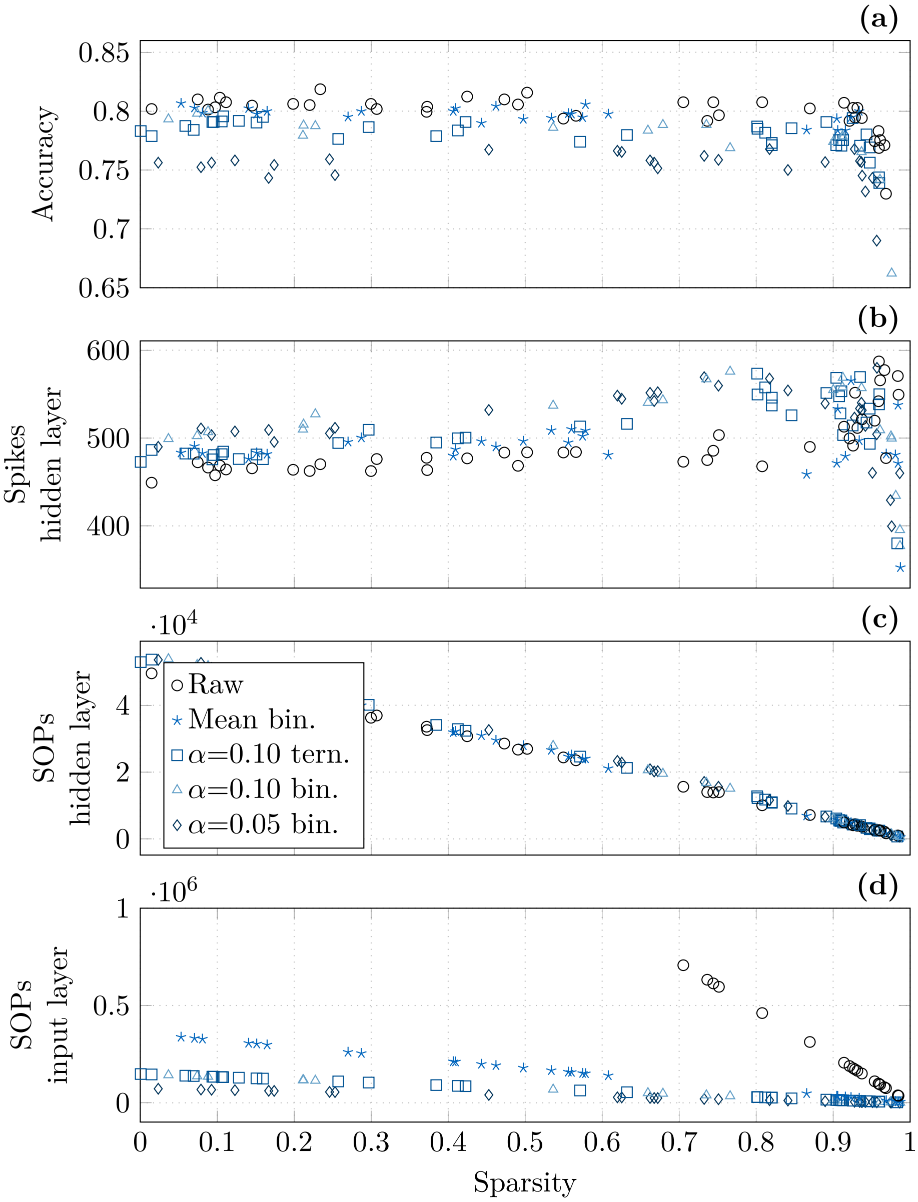


Figure 6.9: Network sparsity sweep. The top-most plot (a) shows the classification performance over the connection sparsity of recurrently connected SNNs on the TinyRadarNN dataset. The performance of networks using unaltered RDM sequences as well as those using one of the four binarization configurations to binarize the input are depicted. (b) shows the number of emitted spikes in the hidden layer. The number of SOPs shown in (c), however, is not influenced by the chosen input format because the increasing connection sparsity outweighs all other effects. The number of SOPs induced by the input layer (d) shows the largest differences between the encoding schemes. Due to the sparser activity of the binarized schemes, the number of induced SOPs is significantly lower and decreases further the sparser the network gets.

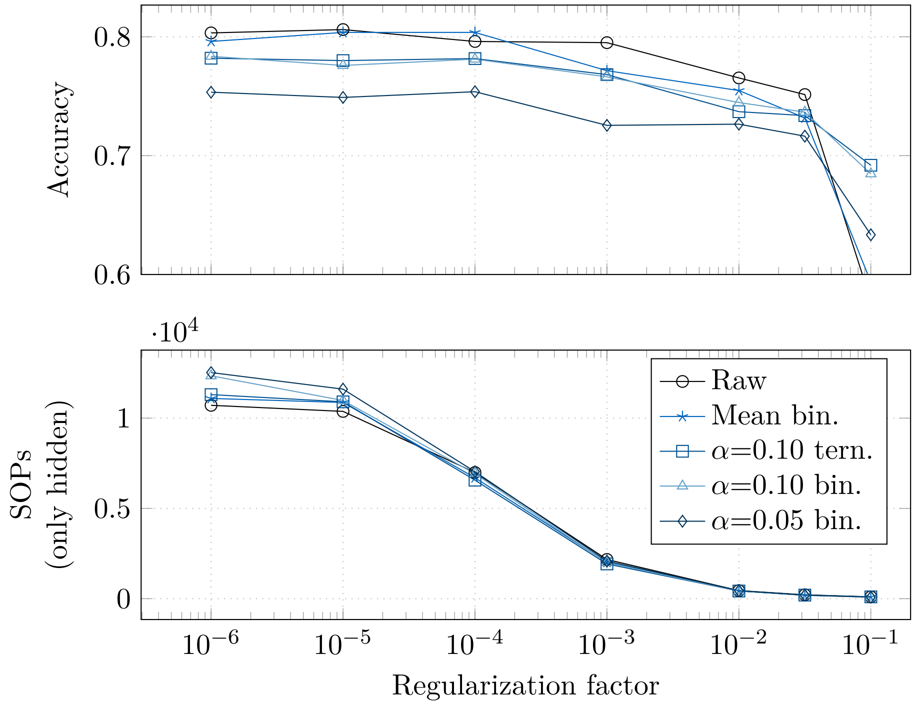


Figure 6.10: Network activity regularization sweep. The classification performance versus the regularization factor of the activity regularization is shown in the upper plot. All networks are sparsely connected with a connection sparsity of 70%. Using the different input encoding schemes, different accuracies are reached. However, with increasing regularization factor, the general arrangement among the different schemes stays constant. The number of SOPs in the hidden layer decreases significantly with increasing regularization factor. There are no noteworthy differences in the network activity between the encoding schemes.

the qualitative results are similar for all sparsity levels below the turning point shown in Figure 6.9. The arrangement of the accuracy levels for the different encoding schemes meets the performances reported for the LOSO cross-validation shown in table 6.2. The reached accuracies degrade slowly with an increasing regularization factor. At a factor of 0.1, the performance drops significantly, suggesting that the optimization goal of minimizing the spike activity now tops the minimization of the classification error. The performance drop is particularly given for the encoding schemes with no binarization and the mean binarization scheme.

The course of the number of synaptic events of the hidden layer is shown in the bottom plot of Figure 6.10. As already shown in Figure 6.9, the different encoding schemes only slightly differ in the number of induced synaptic events of the hidden layer. The majority of the synaptic events of the whole network is given by the input itself, which is two orders of magnitude larger. However, this is not affected by the activity regularization of the hidden layer. The low activity with which the networks can maintain a high accuracy is remarkable: 200 to 500 synaptic events, which correspond to 10 to 20 spikes being emitted by the population of the hidden layer, are utilized at regularization factors of 0.01 to 0.03.

6.5 Discussion of the Results

This chapter showed the successful application of SNNs to classify hand gestures in radar data sequences. The networks showed comparable performances to ANN-based solutions on two different datasets (see table 6.2). In some cases, the SNN-based approach was able to surpass the accuracies reported in the literature slightly. Accordingly, this approach shows to be a viable solution to this simple sequence classification problem.

The most notable differences between the approaches are their computational complexities. Even in the worst configuration, the SNN was able to solve the recognition task using less than $2.5 \cdot 10^6$ synaptic operations per sequence on the TinyRadarNN dataset. In comparison, [230] reports $20 \cdot 10^6$ MAC operations for each single frame for a convolutional neural network-based solution to the same dataset with a slightly worse classification performance. For the whole sequence, this totals $500 \cdot 10^6$ MAC operations. In our experiments using the TinyRadarNN dataset, we drastically reduced the resolution of the range measurements to reduce the overall size of the input and, with that, the number of necessary operations. As our results show, this reduced resolution still provides enough information to reach the same accuracy levels. In order to include this in the comparison, the number of operations within the network in [230] has to be adapted by the factor of 16. Since most of the MAC operations are to be assigned to the convolutions in the first layer, this is a reasonable assumption. However, even with this factor, the SNN-based solutions need far fewer operations to solve the problem. Additionally, as noted in Section 4.3, the main operation involved in the processing of SNNs is the add operation. ANNs, in contrast, depend on more costly multiplications. In the best case, using 3-level binarization or mean binarization in conjunction with regularization and pruning to 90% sparsity, the number of SOPs decreases to $25 \cdot 10^3$ while maintaining the same level of accuracy. The choice of the encoding scheme in the experiments mostly influenced the number of synaptic events induced by the input. The influence on the SOPs in the hidden layer is negligible.

As we showed in table 6.3 and Figure 6.9, the binarization of the input RDMs drastically reduced the number of operations required in the input layer. Additionally, increasing the connection sparsity by pruning ineffective connections led to a further reduction of required operations. Of course, pruning and skipping ineffective operations with zeros is not exclusive to SNNs, but their sparse event-based communication scheme vastly leverages these techniques.

The good performance of all approaches heavily relies on the digital preprocessing of the radar data. To generate the RDMs, a multitude of FFTs has to be calculated. The complexity of the classical radix-2 algorithm [240] to calculate the FFT scales with $N/2 \log_2 N$, resulting in $2N \log_2 N$ real MAC operations, ignoring further additions and control logic. Modern algorithms undercut this number by several percent [241], but the order of magnitude stays the same. For the downscaled TinyRadarNN dataset, this results in nearly 10^6 operations to calculate the RDMs of one gesture sequence. This number is huge compared to the number of operations required for the optimized SNNs.

An alternative approach without calculating the RDMs is the direct analysis of the IF signal of the radar sensor in the time domain. The main challenge, in this case, is encoding the signal into a spike representation. The use of the RF neuron-based encoding similar to the speech recognition application is not possible due to the loss of the phase information as shown in Section 3.4. The direct mapping of the signal course using TC methods is not practical due to the high signal frequencies in the order of several MHz and the rich variability in amplitude, frequency, and phase. To be able to capture these details would require a vast number of spikes in relatively small time intervals. Frame-based encoding methods in which each sample is translated into one temporal event likewise lead to large numbers of synaptic events to be processed. However, the tradeoff can be justified if operation-intense calculations like the Fourier transform can be replaced. Approaches like a spike-based Fourier transform [242] might render this a possibility.

7 Summary and Conclusion

Spiking Neural Networks (SNNs) offer many potential advantages over classical neural network approaches. Especially the evaluation of data sequences in low-power environments using specialized neuromorphic hardware can be a promising field of application as it combines multiple key features of the spike-based communication scheme and biologically inspired neuron models. In this work, approaches and the evaluation of specific applications were presented, which support this hypothesis.

In the first chapter, we motivated this work and proposed our research questions. Following on from the first chapter, we have provided a summary of the background for the following approaches in chapter 2. The methodological approaches were introduced afterwards; chapter 3 discussed the encoding of signals into spikes, and chapter 4 presented methods to train and optimize neural networks based on spiking neurons. Finally, the methods were evaluated in two exemplary applications, speech recognition (chapter 5) and gesture recognition (chapter 6).

The following sections give conclusions of the individual chapters in more detail.

7.1 Background

Chapter 2 provided the background and introduced related research on which this work is based. First, we introduced the Hodgkin-Huxley neuron model, which lays the foundations for every neuronal model. The authors Hodgkin and Huxley received the Nobel Prize in Medicine in 1963 for their fundamental contributions. We then derived the Leaky Integrate-and-Fire (LIF) and Resonate-and-Fire (RF) neuron models, which are used in this work. Subsequently, we gave an overview of biologically inspired learning algorithms to train SNNs. The learning rule used in this work, gradient descent using pseudo gradients, was presented afterwards. Many of the hypotheses why SNNs can outperform Artificial Neural Networks (ANNs) are based on the utilization of neuromorphic hardware. The most prominent chips were summarized in Section 2.3. We showed that there are large general purpose accelerators, smaller architectures, which often showcase specific technological or architectural advances, and Field-Programmable Gate Array (FPGA)-based solutions, which offer to accelerate SNNs on generally available hardware. To conclude the background chapter, a review of encoding schemes for the use in SNNs was given.

7.2 Signal Encoding

In chapter 3, we discussed the translation of data sequences into spikes. This translation is necessary to leverage the spike-based communication scheme of the SNNs. Therefor, the encoding schemes introduced in the previous chapter were evaluated with respect to the targeted applications. We identified two possible modes to convert the data sequences into spikes: frame-based approaches and streaming approaches.

Frame-based approaches convert sub-sequences of the signal with fixed lengths into spikes, which are evaluated at once. These approaches are suited for applications with multidimensional inputs with low update frequencies. When the input data streams are preprocessed using classical Fourier transform-based signal processing approaches, frame-based encodings are most suited.

Streaming approaches convert the signal continuously. They are useful to encode fast changing one-dimensional signals. Due to the sinusoidal characteristics of the analyzed signals, we proposed the use of RF neurons to convert the analog input signal into spike trains. The data stream is analyzed continuously and is, therefore, part of the streaming approaches. We showed that using the resonating neurons as input encoders, a frequency-selective spike response can be generated. To perform the encoding, the signal is directly applied to the input of the neuron. Because of its properties of a resonator, the two coupled membrane capacitors start to resonate when the matching frequency is present within the signal. When the resonant oscillation reaches a threshold level, a spike is emitted. The limitation of this encoding is that only the real amplitude spectrum of the signal can be represented. The amplitude of the spectral component is expressed in the timing of the emitted spikes. The phase of the spectral component also influences the precise timing of the emitted spike. This phase induced time shift, however, is small compared to the fluctuations introduced by amplitude noise. Thus, the information about the signal phase gets lost.

7.3 Network Architecture and Training

The methods involving the architectures and the training of the SNNs were presented in chapter 4. First, we introduced the neuron models and connectivity schemes, which were used in our simulations. These definitions were followed by methods that improve the SNNs during training. Specifically, we covered optimization variables, regularization techniques, and the reduction of the networks' complexity by pruning connections with low influences on the network output.

The last section within chapter 4 discussed the implications of realizing SNNs in neuromorphic hardware. Therefor, we first analyzed the effect of architectural choices on the hardware realizations. Next, we considered the relation of the number of trainable variables, neurons, and synapses for different layer types. We closed the chapter with a reflection on the energy consumption of SNNs in comparison with ANNs.

7.4 Applications

Chapters 5 and 6 contain the evaluations of the methods proposed in chapters 3 and 4. The methods were evaluated using two different sequence classification applications: speech recognition (chapter 5) and radar-based hand gesture recognition (chapter 6).

7.4.1 Speech Recognition

The speech recognition application comprised the detection and classification of keywords in short audio snippets. We proposed the usage of RF neurons to encode the audio stream into frequency-selective spike trains. This approach was compared to classical approaches using Fourier transform-based preprocessing. The different preprocessing methods were used to feed neural networks based on classical and spiking neurons.

For both neuron types, networks using convolutional layers together with temporal memories reached the best performances. They reached the highest scores when the input was present as a log-Mel spectrogram. Generally, the ANNs performed slightly better than the SNNs. Both neuron types were also able to reach good classification performances using RF neurons as input encoders. This demonstrates the overall feasibility of the approach. However, the performances were still lower than those of the configurations that used input sequences of raw or binarized Fourier spectra.

The first experiments were followed by ablation studies, which aimed at getting a more profound understanding of the network behaviors when scaling certain parameters. We, therefore, compared Long Short-Term Memorys (LSTMs) with recurrently connected SNNs on RF-encoded input features. These two networks performed best on the new type of encoding. We showed the relation between network sparsity, the reached classification accuracy, and the network activity. When increasing the targeted sparsity, which is obtained by pruning during the training phase, the reached accuracy stays nearly constant. At the same time, the network activity in the form of emitted spikes grows. However, with the increased connection sparsity, each emitted spike has to be considered at fewer downstream neurons. In sum, this leads to a declining number of Synaptic Operations (SOPs). We further showed that regularizing the spike activity of a neuron can further reduce the overall network activity.

We concluded the chapter with a discussion of the results. We reflected on the examined network architectures and the found relations as well as their impact on the complexity and energy consumption in neuromorphic realizations. Especially convolutional layers – with all their advantages as generalizing feature detectors – result in humongous network realizations. We showed that using two-dimensional convolutions, 10 to 50 times more synaptic events are triggered than with feedforward or recurrent connectivity schemes. The best overall performance could be reached with a recurrently connected population of sparsely connected neurons. With that, the number of neurons, synapses, and SOPs was minimal while suffering only minor accuracy losses.

7.4.2 Hand Gesture Recognition

In the second application, we used recurrently connected SNNs to classify hand gestures based on distance-velocity measurement sequences of radar sensors. The sequences of frames were encoded using different approaches.

We showed that our SNN-based implementations match or slightly surpass the accuracies reported in the literature. Similar to the previous chapter, the presentation of the initial results were followed by ablation studies. The best results were achieved using mean-binarization to encode the input frames. This proved the assumption that the actual amplitude measurements are less important, but only the notion of the largest measurement entries holds enough information. Additionally, we showed that the selection of the input encoding scheme can directly influence the network activity in the form of necessary SOPs. The experiments involving pruning and regularization lead to similar results as in the ablation studies of the speech recognition experiments: networks can be pruned during training to sparsity levels of 80% without accuracy losses. We could show again that with higher sparsity levels, the overall spike activity rises, but due to less downstream neurons, the resulting number of SOPs drops. Additionally, gentle spike activity regularization should always be used to encourage sparse temporal spike activity without harming performance.

In the subsequent discussion of the experiments' results, we focused on the number of Multiply-Accumulate (MAC) operations involved in the execution of the proposed SNNs in digital neuromorphic hardware and the ANNs given in the literature. We showed that our SNN-based approaches need at least one order of magnitude fewer operations while still reaching the same classification accuracy. Additionally, we considered the number of MAC operations used for the preprocessing of the radar signals. Because the preprocessing of the radar signals requires the computation of multiple two-dimensional Fast Fourier Transforms (FFTs), the number of operations involved in this step is likely to surpass the operations needed for the actual network inference. This, in turn, motivates the development of alternative preprocessing methods, which we will discuss in chapter 8.

7.4.3 Similarities and Differences

In both applications, we were able to solve the sequence classification problems with high accuracy. While matching and surpassing the performance of other works in the gesture recognition example, the performance during the speech recognition task came short compared with related works. The question arises why this is the case.

The processing of radar data based on Range-Doppler Matrices (RDMs) seems to be a perfect fit for SNNs. In this context, once the RDMs are calculated, the amplitude information is of less relevance. We assumed that instead of the precise amplitude of each bin within the RDMs, only the relative information is important in our application. It is sufficient to know whether an entry belongs to the highest valued bins of one frame. The assumption is valid for this unique application because we only consider a single target that is close to the sensor.

Additionally, the structure of the input data is simple. Movements within the RDMs are always centered around the central line corresponding to zero velocity. Thus, the network must detect the existence and sequences of binary events in the input to infer the correct movement. The small, recurrently connected SNN achieves exactly that. Convolutional layer structures do not add any benefit because the influence of translationally invariant features is small compared with the binary activity patterns at a larger scale.

Speech recognition, on the other hand, is heavily reliant on the accurate representation of the audio spectrum. The binarization of the spectra, thus, leads to decreased classification performances. Additionally, features are translationally invariant due to the various pitches and pronunciations of the speakers. Because of that, convolutional layers on the raw spectra lead to very high accuracies. With the use of RF neurons to encode the speech signal, we aimed to eliminate the dependence on amplitude information from the first stage on. This type of feature encoding resulted in inferior performances compared with the classical techniques, which utilize Fourier transforms and further digital processing steps. However, it opens a new set of possibilities to analyze audio signals, with the potential to eliminate the need for digital preprocessing.

The common features of both approaches are the same scaling effects in performance, sparsity, and activity. In both applications, sparsifying the connectivity matrices by pruning resulted in higher spike rates and overall lower number of neuron updates due to fewer synaptic input events. The reached classification accuracy stayed constant in both cases, followed by a sudden drop in performance at sparsity rates of about 80 to 95%. The regularization of the networks' activities during training led to a notable reduction of the required operations if tuned thoroughly.

7.5 Limitations

In the first chapter, we motivated our research with the overall question:

> *Are neural networks based on spiking neurons viable and advantageous alternatives to common ANNs in real-world applications?*

This high-level question still cannot be answered conclusively. However, we provided evidences that SNNs are able to outperform ANNs in certain applications. Additionally, SNNs show to have favorable scaling properties in small-scale applications.

Sparsifying the input to the networks is one of the major options to ensure a highly energy-efficient execution of the SNNs. The event-based processing scheme enables the networks to only contribute to the dynamic power consumption when spikes need to be processed. In our applications, we achieved this by using RF-based encoding and binarization. However, this kind of sparse input conversion might not be applicable to many other applications. As soon as rate-based encoding schemes are utilized to represent a richer set of values, many of the SNNs' advantages disappear, and ANN-based implementations might be favorable again.

Much of the pending success of SNNs is dependent on the development of future neuromorphic hardware realizations. The architecture, configurability, and availability of the accelerators will show how well solutions based on SNNs will perform compared to classical networks.

The stability of the training process using pseudo gradient-based Backpropagation Through Time (BPTT) was comparatively low in our experiments. Depending on the random initialization of the networks, vastly different results were achieved. Especially while using pruning and regularization methods, networks often result in untrainable configurations. This behavior is far more pronounced than with ANNs due to the “all-or-nothing” characteristic of the spike events.

8 Outlook

In the coming decades, more and more devices, applications, and processes will be enhanced by the utilization of data-driven algorithms in the form of neural networks. No matter whether highly automated vehicles, smart assistants and wearables, the connected industry, or cloud services, neural networks will be powering large portions of the world's technologies. Therefore, our overall motivation and goals for this work stay valid for future research: reducing the energy consumption and increasing the capabilities of these networks by exploring new methods.

In this work, we investigated neural networks based on biologically inspired spiking neurons. With their event-based communication scheme and their inherent temporal affinity, SNNs are capable alternatives to ANNs, which go well with the above-mentioned goals. To continue these thoughts, further research should address the topics of improving the overall performance of SNNs and providing methods for the comparison of approaches based on classical algorithms, ANNs, and SNNs.

For the analysis of sinusoidal signal streams, we proposed the use of RF neurons to translate the time-domain signals into feature-rich spike trains. We demonstrated the viability of this approach. The approach has the potential to provide alternatives to classical approaches involving the calculation of Fourier transforms. In this work, we showed the RF neurons' application in the area of speech recognition. We additionally demonstrated the detection of interference in automotive radar applications in a related article [53]. Next to further theoretical analyses and improvements, hardware realizations should be developed to prove the hypothetical advantages in the power consumption and deliver metrics for the comparison with classical approaches.

Further research should also focus on advancing the training process of SNNs. Many of the methods developed for ANNs can be adapted for the use for SNNs. However, the event-based processing scheme also necessitates the development of specialized approaches. The instability of the pseudo gradient-based training, the initialization of the networks, or the temporal credit assignment in long sequences, for example, needs further addressing.

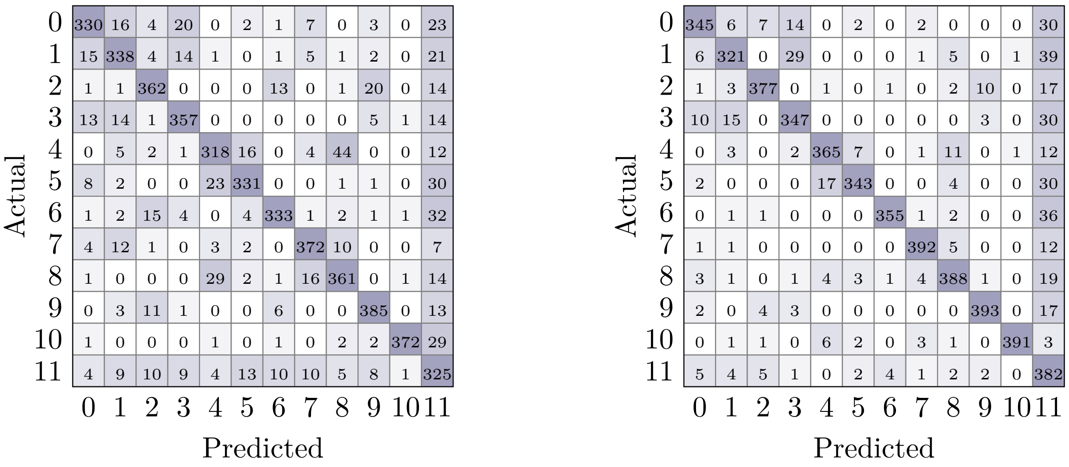
A direct comparison between ANNs and SNNs is often not trivially possible. For ANNs, there exist efforts to provide a standardized benchmark suite [243]. The benchmark assesses the performance of standardized network architectures and tasks of different domains when executed on different hardware. The evaluated performance indicators are the inference latency, execution speed, and, in some cases, power consumption. With this benchmark being developed to compare different hardware configurations executing the networks, others can be designed to compare different architectures or training methods. For an elaborate comparison of SNNs, for example, the benchmarks should contain the achieved performance, metrics reflecting the network's complexity, a comprehensive breakdown of the energy consumption, and the way how the network was executed, being it a software-based simulation or realization in hardware.

In this work, we focused on the use of SNNs in low-power applications. However, there are more research directions on the topic of biologically inspired neural networks, which can positively influence each other. Research directed towards understanding biological nervous systems, for example, can provide new insights into the complex interactions of large networks, which can be transferred to technical applications. Application-oriented research, in turn, can provide evidence for phenomenons in vivo, which could not be explained before. In any case, the author looks forward to future developments in this exciting field.

Appendix

Table A.1: Layer-wise activity of the evaluated densely connected SNNs to solve the speech recognition task. All networks comprise 50,000 trainable parameters.

Model	**Spikes**	**SOPs**	**ΣSOPs**
DNN	1505 + 870	52,000 + 301,000 + 10,440	363,440
RNN	1339	52,000 + 283,868	335,868
1D-CNN	1385 + 818	34,320 + 96,950 + 67,076	198,346
1D-RCNN	1747 + 569	34,320 + 176,447 + 45,520	256,287
2D-CNN	4627 + 3935 + 1146	131,040 + 2,332,008 + 1,983,240 + 13,752	4,460,040



Actual \ Predicted	0	1	2	3	4	5	6	7	8	9	10	11
0	330	16	4	20	0	2	1	7	0	3	0	23
1	15	338	4	14	1	0	1	5	1	2	0	21
2	1	1	362	0	0	0	13	0	1	20	0	14
3	13	14	1	357	0	0	0	0	0	5	1	14
4	0	5	2	1	318	16	0	4	44	0	0	12
5	8	2	0	0	23	331	0	0	1	1	0	30
6	1	2	15	4	0	4	333	1	2	1	1	32
7	4	12	1	0	3	2	0	372	10	0	0	7
8	1	0	0	0	29	2	1	16	361	0	1	14
9	0	3	11	1	0	0	6	0	0	385	0	13
10	1	0	0	0	1	0	1	0	2	2	372	29
11	4	9	10	9	4	13	10	10	5	8	1	325

(a) ANN - LSTM on RF inputs

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9	10	11
0	345	6	7	14	0	2	0	2	0	0	0	30
1	6	321	0	29	0	0	0	1	5	0	1	39
2	1	3	377	0	1	0	1	0	2	10	0	17
3	10	15	0	347	0	0	0	0	0	3	0	30
4	0	3	0	2	365	7	0	1	11	0	1	12
5	2	0	0	0	17	343	0	0	4	0	0	30
6	0	1	1	0	0	0	355	1	2	0	0	36
7	1	1	0	0	0	0	0	392	5	0	0	12
8	3	1	0	1	4	3	1	4	388	1	0	19
9	2	0	4	3	0	0	0	0	0	393	0	17
10	0	1	1	0	6	2	0	3	1	0	391	3
11	5	4	5	1	0	2	4	1	2	2	0	382

(b) SNN - 2CNN on log-Mel spectogram

Figure A.1: Confusion matrix for a recurrent ANN and a convolutional SNN to solve the speech recognition task. Both networks comprise 50,000 parameters. The ANN uses on log-Mel-encoded inputs. The SNN is fed with RF-generated spike trains.

Bibliography

[1] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*. The MIT Press, 2016, vol. 1, 800 pp.

[2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4. edition. Hoboken: Pearson, 2020, 1152 pp.

[3] E. R. Kandel, J. D. Koester, S. H. Mack, and S. A. Siegelbaum, *Principles of Neural Science*, 6th ed. New York: McGraw-Hill Education, 2021, 1696 pp.

[4] W. B. Levy and V. G. Calvert, "Computation in the human cerebral cortex uses less than 0.2 watts yet this great expense is optimal when considering communication costs", *BioRxiv*, 2020.

[5] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI", *Communications of the ACM*, vol. 63, no. 12, pp. 54–63, 2020.

[6] E. Gibney, "Google AI algorithm masters ancient game of Go", *Nature News*, vol. 529, no. 7587, p. 445, 2016.

[7] D. Silver, A. Huang, C. J. Maddison, *et al.*, "Mastering the game of Go with deep neural networks and tree search", *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[8] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello Edge: Keyword Spotting on Microcontrollers", 2017. arXiv: 1711.07128.

[9] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal Brain Damage", in *Advances in Neural Information Processing Systems*, 1990, pp. 598–605.

[10] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both Weights and Connections for Efficient Neural Network", *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[11] J. Frankle and M. Carbin, "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks", 2018. arXiv: 1803.03635.

[12] A. Lavin and S. Gray, "Fast Algorithms for Convolutional Neural Networks", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4013–4021.

[13] A. G. Howard, M. Zhu, B. Chen, *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", 2017. arXiv: 1704.04861.

[14] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.

[15] A. Shafiee, A. Nag, N. Muralimanohar, *et al.*, “ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars”, *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[16] W. Maass, “Networks of Spiking Neurons: The Third Generation of Neural Network Models”, *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[17] J. Schmidhuber, “Deep Learning in Neural Networks: An Overview”, *Neural networks*, vol. 61, pp. 85–117, 2015.

[18] C.-S. Poon and K. Zhou, “Neuromorphic silicon neurons and large-scale neural networks: Challenges and opportunities”, *Frontiers in neuroscience*, vol. 5, p. 108, 2011.

[19] 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.

[20] 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.

[21] M. Davies, N. Srinivasa, T. Lin, *et al.*, “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning”, *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[22] P. Blouw and C. Eliasmith, “Event-Driven Signal Processing with Neuromorphic Computing Systems”, in *2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2020, pp. 8534–8538.

[23] S. Davidson and S. B. Furber, “Comparison of Artificial and Spiking Neural Networks on Digital Hardware”, *Frontiers in Neuroscience*, vol. 15, 2021.

[24] C. Farabet, R. Paz, J. Pérez-Carrasco, *et al.*, “Comparison between frame-constrained fix-pixel-value and frame-free spiking-dynamic-pixel convnets for visual processing”, *Frontiers in neuroscience*, vol. 6, p. 32, 2012.

[25] D. Neil, M. Pfeiffer, and S.-C. Liu, “Learning to be Efficient: Algorithms for Training Low-Latency, Low-Compute Deep Spiking Neural Networks”, in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 2016, pp. 293–298.

[26] M. Pfeiffer and T. Pfeil, “Deep Learning With Spiking Neurons: Opportunities and Challenges”, *Frontiers in neuroscience*, vol. 12, 2018.

[27] E. M. Izhikevich, “Which Model to Use for Cortical Spiking Neurons?”, *IEEE transactions on neural networks*, vol. 15, no. 5, pp. 1063–1070, 2004.

[28] J. L. Lobo, I. Laña, J. Del Ser, M. N. Bilbao, and N. Kasabov, “Evolving Spiking Neural Networks for Online Learning over Drifting Data Streams”, *Neural Networks*, vol. 108, pp. 1–19, 2018.

[29] M. Nolte, M. W. Reimann, J. G. King, H. Markram, and E. B. Muller, “Cortical reliability amid noise and chaos”, *Nature communications*, vol. 10, no. 1, pp. 1–15, 2019.

[30] J. M. Cruz-Albrecht, M. W. Yung, and N. Srinivasa, “Energy-Efficient Neuron, Synapse and STDP Integrated Circuits”, *IEEE transactions on biomedical circuits and systems*, vol. 6, no. 3, pp. 246–256, 2012.

[31] A. Tavanaei, T. Masquelier, and A. Maida, “Representation Learning using Event-based STDP”, *Neural Networks*, vol. 105, pp. 294–303, 2018.

[32] A. R. Young, M. E. Dean, J. S. Plank, and G. S. Rose, “A Review of Spiking Neuromorphic Hardware Communication Systems”, *IEEE Access*, vol. 7, pp. 135 606–135 620, 2019.

[33] E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado, and G. Cauwenberghs, “Event-Driven Contrastive Divergence for Spiking Neuromorphic Systems”, *Frontiers in neuroscience*, vol. 7, p. 272, 2014.

[34] E. Mueller, J. Hansjakob, D. Auge, and A. Knoll, “Minimizing Inference Time: Optimization Methods for Converted Deep Spiking Neural Networks”, in *2021 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2021, pp. 1–8.

[35] J. Wu, E. Yılmaz, M. Zhang, H. Li, and K. C. Tan, “Deep Spiking Neural Networks for Large Vocabulary Automatic Speech Recognition”, *Frontiers in Neuroscience*, vol. 14, p. 199, 2020.

[36] S. Thorpe, D. Fize, and C. Marlot, “Speed of Processing in the Human Visual System”, *nature*, vol. 381, no. 6582, p. 520, 1996.

[37] R. S. Johansson and I. Birznieks, “First Spikes in Ensembles of Human Tactile Afferents Code Complex Spatial Fingertip Events”, *Nature neuroscience*, vol. 7, no. 2, p. 170, 2004.

[38] T. Gollisch and M. Meister, “Rapid Neural Coding in the Retina with Relative Spike Latencies”, *Science*, vol. 319, no. 5866, pp. 1108–1111, 2008.

[39] J. J. Hopfield, “Pattern Recognition Computation Using Action Potential Timing for Stimulus Representation”, *Nature*, vol. 376, no. 6535, p. 33, 1995.

[40] C. Kayser, M. A. Montemurro, N. K. Logothetis, and S. Panzeri, “Spike-Phase Coding Boosts and Stabilizes Information Carried by Spatial and Temporal Spike Patterns”, *Neuron*, vol. 61, no. 4, pp. 597–608, 2009.

[41] C. M. Gray, P. König, A. K. Engel, and W. Singer, “Oscillatory Responses in Cat Visual Cortex Exhibit Inter-Columnar Synchronization Which Reflects Global Stimulus Properties”, *Nature*, vol. 338, no. 6213, p. 334, 1989.

[42] S. Thorpe and J. Gautrais, “Rank Order Coding”, in *Computational Neuroscience*, Springer, 1998, pp. 113–118.

[43] J. Gautrais and S. Thorpe, “Rate Coding versus Temporal Order Coding: A Theoretical Approach”, *Biosystems*, vol. 48, no. 1-3, pp. 57–65, 1998.

[44] M. Li and J. Z. Tsien, “Neural Code-Neural Self-Information Theory on How Cell-Assembly Code Rises from Spike Time and Neuronal Variability”, *Frontiers in Cellular Neuroscience*, vol. 11, p. 236, 2017.

[45] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. S. Maida, "Deep Learning in Spiking Neural Networks", *Neural Networks*, vol. 111, pp. 47–63, 2019.

[46] S. M. Bohte, "Error-backpropagation in Networks of Fractionally Predictive Spiking Neurons", in *International Conference on Artificial Neural Networks*, Springer, 2011, pp. 60–68.

[47] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Long Short-Term Memory and Learning-to-Learn in Networks of Spiking Neurons", in *Advances in Neural Information Processing Systems*, 2018, pp. 787–797.

[48] F. Zenke and S. Ganguli, "SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks", *Neural computation*, vol. 30, no. 6, pp. 1514–1541, 2018.

[49] 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 Process. Mag.*, vol. 36, no. 6, pp. 51–63, 2019.

[50] D. Auge, J. Hille, E. Mueller, and A. Knoll, "A Survey of Encoding Techniques for Signal Processing in Spiking Neural Networks", *Neural Processing Letters*, 2021.

[51] D. Auge and E. Mueller, "Resonate-and-Fire Neurons as Frequency Selective Input Encoders for Spiking Neural Networks", *TUM (Technical Report)*, 2020.

[52] D. Auge, J. Hille, F. Kreutz, E. Mueller, and A. Knoll, "End-to-end Spiking Neural Network for Speech Recognition Using Resonating Input Neurons", in *30th International Conference on Artificial Neural Networks (ICANN)*, 2021.

[53] J. Hille, D. Auge, C. Grassmann, and A. Knoll, "Resonate-and-Fire Neurons for Radar Interference Detection", in *International Conference on Neuromorphic Systems 2022*, Association for Computing Machinery, 2022.

[54] D. Auge, P. Wenner, and E. Mueller, "Hand Gesture Recognition using Hierarchical Temporal Memory on Radar Sequence Data", in *Bernstein Conference 2020*, 2020.

[55] D. Auge, J. Hille, E. Mueller, and A. Knoll, "Hand Gesture Recognition in Range-Doppler Images Using Binary Activated Spiking Neural Networks", in *IEEE International Conference on Automatic Face and Gesture Recognition 2021*, 2021.

[56] *KI-ASIC — Mikroelektronikforschung*, `https://www.elektronikforschung.de/projekte/ki-asic`, Accessed: 2021-12-08.

[57] B. Vogginger, F. Kreutz, J. Lopez-Randulfe, *et al.*, "Automotive Radar Processing with Spiking Neural Networks: Concepts and Challenges", *Frontiers in Neuroscience*, p. 414,

[58] E. Mueller, V. Studenyak, D. Auge, and A. Knoll, "Spiking Transformer Networks: A Rate Coded Approach for Processing Sequential Data", in *International Conference on Systems and Informatics (ICSAI)*, 2021.

[59] E. Mueller, D. Auge, S. Klimaschka, and A. Knoll, "Neural Oscillations for Energy-Efficient Hardware Implementation of Sparsely Activated Deep Spiking Neural Networks", 2022.

[60] J. Hille, D. Auge, C. Grassmann, and A. Knoll, “FMCW radar2radar Interference Detection with a Recurrent Neural Network”, in *2022 IEEE Radar Conference (RadarConf22)*, IEEE, 2022, pp. 1–6.

[61] S. Nair, S. Shafaei, D. Auge, and A. C. Knoll, “An Evaluation of” Crash Prediction Networks”(CPN) for Autonomous Driving Scenarios in CARLA Simulator.”, in *SafeAI@ AAAI*, 2021.

[62] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity.* Cambridge university press, 2002.

[63] 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.

[64] E. M. Izhikevich, “Resonate-and-fire neurons”, *Neural networks*, vol. 14, no. 6-7, pp. 883–894, 2001.

[65] B. Hutcheon and Y. Yarom, “Resonance, oscillation and the intrinsic frequency preferences of neurons”, *Trends in Neurosciences*, vol. 23, no. 5, pp. 216–222, 2000.

[66] M. J. E. Richardson, N. Brunel, and V. Hakim, “From Subthreshold to Firing-Rate Resonance”, *Journal of Neurophysiology*, vol. 89, no. 5, pp. 2538–2554, 2003.

[67] I. Lampl and Y. Yarom, “Subthreshold oscillations and resonant behavior: Two manifestations of the same mechanism”, *Neuroscience*, vol. 78, no. 2, pp. 325–341, 1997.

[68] H. G. Rotstein, “Frequency Preference Response to Oscillatory Inputs in Two-dimensional Neural Models: A Geometric Approach to Subthreshold Amplitude and Phase Resonance”, *The Journal of Mathematical Neuroscience*, vol. 4, no. 1, p. 11, 2014.

[69] M. St-Hilaire and A. Longtin, “Comparison of Coding Capabilities of Type I and Type II Neurons”, *Journal of Computational Neuroscience*, vol. 16, no. 3, pp. 299–313, 2004.

[70] T. Asai, Y. Kanazawa, and Y. Amemiya, “A Subthreshold MOS Neuron Circuit Based on the Volterra System”, *IEEE transactions on neural networks*, vol. 14, no. 5, pp. 1308–1312, 2003.

[71] K. Nakada, T. Asai, and H. Hayashi, “A Silicon Resonate-and-Fire Neuron Based on the Volterra System”, in *Int. Symp. on Nonlinear Theory and Its Applications*, 2005, pp. 82–85.

[72] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory.* Psychology Press, 2005.

[73] S. Löwel and W. Singer, “Selection of Intrinsic Horizontal Connections in the Visual Cortex by Correlated Neuronal Activity”, *Science*, vol. 255, no. 5041, pp. 209–212, 1992.

[74] T. Masquelier and S. J. Thorpe, “Unsupervised Learning of Visual Features through Spike Timing Dependent Plasticity”, *PLoS computational biology*, vol. 3, no. 2, e31, 2007.

[75] T. Masquelier, R. Guyonneau, and S. J. Thorpe, “Spike Timing Dependent Plasticity Finds the Start of Repeating Patterns in Continuous Spike Trains”, *PloS one*, vol. 3, no. 1, e1377, 2008.

[76] T. Masquelier and S. R. Kheradpisheh, “Optimal Localist and Distributed Coding of Spatiotemporal Spike Patterns Through STDP and Coincidence Detection”, *Frontiers in Computational Neuroscience*, vol. 12, 2018.

[77] P. U. Diehl and M. Cook, “Unsupervised Learning of Digit Recognition Using Spike-Timing-Dependent Plasticity”, *Frontiers in computational neuroscience*, vol. 9, p. 99, 2015.

[78] 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.

[79] S. Fusi, M. Annunziato, D. Badoni, A. Salamon, and D. J. Amit, “Spike-driven synaptic plasticity: Theory, simulation, VLSI implementation”, *Neural computation*, vol. 12, no. 10, pp. 2227–2258, 2000.

[80] N. Kasabov, K. Dhoble, N. Nuntalid, and G. Indiveri, “Dynamic evolving spiking neural networks for on-line spatio-and spectro-temporal pattern recognition”, *Neural Networks*, vol. 41, pp. 188–201, 2013.

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

[82] M. Mozafari, S. R. Kheradpisheh, T. Masquelier, A. Nowzari-Dalini, and M. Ganjtabesh, “First-Spike-Based Visual Categorization Using Reward-Modulated STDP”, *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 12, pp. 6178–6190, 2018.

[83] 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.

[84] R. V. Florian, “The Chronotron: A Neuron That Learns to Fire Temporally Precise Spike Patterns”, *PloS one*, vol. 7, no. 8, e40233, 2012.

[85] A. Mohemmed, S. Schliebs, S. Matsuda, and N. Kasabov, “SPAN: Spike Pattern Association Neuron for Learning Spatio-Temporal Spike Patterns”, *International journal of neural systems*, vol. 22, no. 04, p. 1 250 012, 2012.

[86] B. Widrow and M. E. Hoff, “Adaptive Switching Circuits”, Stanford Univ Ca Stanford Electronics Labs, 1960.

[87] J. D. Victor and K. P. Purpura, “Nature and Precision of Temporal Coding in Visual Cortex: A Metric-Space Analysis”, *Journal of neurophysiology*, vol. 76, no. 2, pp. 1310–1326, 1996.

[88] J. H. Lee, T. Delbruck, and M. Pfeiffer, “Training Deep Spiking Neural Networks Using Backpropagation”, *Frontiers in Neuroscience*, vol. 10, 2016.

[89] T. Nakano, M. Otsuka, J. Yoshimoto, and K. Doya, “A Spiking Neural Network Model of Model-Free Reinforcement Learning with High-Dimensional Sensory Input and Perceptual Ambiguity”, *PLOS ONE*, vol. 10, no. 3, e0115620, 2015.

[90] B. Sallans and G. E. Hinton, “Reinforcement Learning with Factored States and Actions”, *Journal of Machine Learning Research*, vol. 5, pp. 1063–1088, Aug 2004.

[91] M. Otsuka, J. Yoshimoto, and K. Doya, “Free-energy-based Reinforcement Learning in a Partially Observable Environment”, in *ESANN*, 2010.

[92] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, “Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification”, *Frontiers in neuroscience*, vol. 11, p. 682, 2017.

[93] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, “Fast-Classifying, High-Accuracy Spiking Deep Networks through Weight and Threshold Balancing”, in *Neural Networks (IJCNN), 2015 International Joint Conference On*, IEEE, 2015, pp. 1–8.

[94] A. Kasiński and F. Ponulak, “Comparison Of Supervised Learning Methods For Spike Time Coding in Spiking Neural Networks”, *International Journal of Applied Mathematics and Computer Science*, vol. 16, pp. 101–113, 2006.

[95] M. Minsky, “Steps toward Artificial Intelligence”, *Proceedings of the IRE*, vol. 49, no. 1, pp. 8–30, 1961.

[96] P. J. Werbos, “Backpropagation Through Time: What It Does and How to Do It”, *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[97] Y. Bengio, P. Simard, and P. Frasconi, “Learning Long-Term Dependencies with Gradient Descent is Difficult”, *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[98] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks”, in *International Conference on Machine Learning*, PMLR, 2013, pp. 1310–1318.

[99] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory”, *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[100] K. Cho, B. Van Merriënboer, C. Gulcehre, *et al.*, “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”, 2014. arXiv: 1406.1078.

[101] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, *et al.*, “A million spiking-neuron integrated circuit with a scalable communication network and interface”, *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[102] C. Mayr, S. Hoeppner, and S. Furber, "SpiNNaker 2: A 10 Million Core Processor System for Brain Simulation and Machine Learning", 2019. arXiv: `1911.02385`.

[103] *Human Brain Project*, `https://www.humanbrainproject.eu/`, Accessed: 2021-12-08.

[104] *The BRAIN Initiative*, `https://braininitiative.nih.gov/`, Accessed: 2021-12-08.

[105] J. Stuijt, M. Sifalakis, A. Yousefzadeh, and F. Corradi, " Brain: An Event-Driven and Fully Synthesizable Architecture for Spiking Neural Networks", *Frontiers in Neuroscience*, vol. 0, 2021.

[106] Y. Kuang, X. Cui, Y. Zhong, *et al.*, "A 28-nm 0.34-pJ/SOP Spike-Based Neuromorphic Processor for Efficient Artificial Neural Network Implementations", in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2021, pp. 1–5.

[107] S. Furber, "Large-scale neuromorphic computing systems", *Journal of neural engineering*, vol. 13, no. 5, p. 051 001, 2016.

[108] C. D. Schuman, T. E. Potok, R. M. Patton, *et al.*, "A Survey of Neuromorphic Computing and Neural Networks in Hardware", 2017. arXiv: `1705.06963`.

[109] C. S. Thakur, J. L. Molin, G. Cauwenberghs, *et al.*, "Large-Scale Neuromorphic Spiking Array Processors: A Quest to Mimic the Brain", *Frontiers in neuroscience*, vol. 12, p. 891, 2018.

[110] B. Rajendran, A. Sebastian, M. Schmuker, N. Srinivasa, and E. Eleftheriou, "Low-Power Neuromorphic Hardware for Signal Processing Applications: A Review of Architectural and System-Level Design Approaches", *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 97–110, 2019.

[111] M. Bouvier, A. Valentian, T. Mesquida, *et al.*, "Spiking Neural Networks Hardware Implementations and Challenges: A Survey", *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 15, no. 2, pp. 1–35, 2019.

[112] U. Rueckert, "Update on Brain-Inspired Systems", in *NANO-CHIPS 2030*, Springer, 2020, pp. 387–403.

[113] J. L. Lobo, J. Del Ser, A. Bifet, and N. Kasabov, "Spiking Neural Networks and Online Learning: An Overview and Perspectives", *Neural Networks*, vol. 121, pp. 88–100, 2020.

[114] N. T. Carnevale and M. L. Hines, *The NEURON Book*. Cambridge University Press, 2006.

[115] J. M. Bower and D. Beeman, *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System*. Springer Science & Business Media, 2012.

[116] M. Stimberg, R. Brette, and D. F. Goodman, "Brian 2, an intuitive and efficient neural simulator", *eLife*, vol. 8, F. K. Skinner, R. L. Calabrese, F. K. Skinner, F. Zeldenrust, and R. C. Gerkin, Eds., e47314, 2019.

[117] M.-O. Gewaltig and M. Diesmann, "NEST (NEural Simulation Tool)", *Scholarpedia*, vol. 2, no. 4, p. 1430, 2007.

[118] R. A. Tikidji-Hamburyan, V. Narayana, Z. Bozkus, and T. A. El-Ghazawi, "Software for Brain Network Simulations: A Comparative Study", *Frontiers in neuroinformatics*, vol. 11, p. 46, 2017.

[119] E. Falotico, L. Vannucci, A. Ambrosano, *et al.*, "Connecting Artificial Brains to Robots in a Comprehensive Simulation Framework: The Neurorobotics Platform", *Frontiers in Neurorobotics*, vol. 11, 2017.

[120] A. P. Davison, D. Brüderle, J. M. Eppler, *et al.*, "PyNN: A common interface for neuronal network simulators", *Frontiers in Neuroinformatics*, vol. 2, 2009.

[121] 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 2016, Savannah, GA, USA, November 2-4, 2016*, K. Keeton and T. Roscoe, Eds., USENIX Association, 2016, pp. 265–283.

[122] A. Paszke, S. Gross, F. Massa, *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library", in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 8024–8035.

[123] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner, "A Wafer-Scale Neuromorphic Hardware System for Large-Scale Neural Modeling", in *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2010, pp. 1947–1950.

[124] R. Brette and W. Gerstner, "Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity", *Journal of neurophysiology*, vol. 94, no. 5, pp. 3637–3642, 2005.

[125] S. A. Aamir, P. Müller, A. Hartel, J. Schemmel, and K. Meier, "A highly tunable 65-nm CMOS LIF neuron for a large scale neuromorphic system", in *ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference*, IEEE, 2016, pp. 71–74.

[126] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker Project", *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[127] B. V. Benjamin, P. Gao, E. McQuinn, *et al.*, "Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations", *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[128] A. Neckar, S. Fok, B. V. Benjamin, *et al.*, "Braindrop: A Mixed-Signal Neuromorphic Architecture With a Dynamical Systems-Based Programming Model", *Proceedings of the IEEE*, vol. 107, no. 1, pp. 144–164, 2018.

[129] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)", *IEEE transactions on biomedical circuits and systems*, vol. 12, no. 1, pp. 106–122, 2017.

[130] J. Shen, D. Ma, Z. Gu, *et al.*, "Darwin: A neuromorphic hardware co-processor based on spiking neural networks", *Science China Information Sciences*, vol. 59, no. 2, pp. 1–5, 2016.

[131] C. Frenkel, M. Lefebvre, J.-D. Legat, and D. Bol, "A 0.086-mm ^{2}12.7-pJ/SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS", *IEEE transactions on biomedical circuits and systems*, vol. 13, no. 1, pp. 145–158, 2018.

[132] N. Qiao, H. Mostafa, F. Corradi, *et al.*, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses", *Frontiers in neuroscience*, vol. 9, p. 141, 2015.

[133] F. N. Buhler, P. Brown, J. Li, T. Chen, Z. Zhang, and M. P. Flynn, "A 3.43 TOPS/W 48.9 pJ/pixel 50.1 nJ/classification 512 analog neuron sparse coding neural network with on-chip learning and classification in 40nm CMOS", in *2017 Symposium on VLSI Circuits*, IEEE, 2017, pp. C30–C31.

[134] G. K. Chen, R. Kumar, H. E. Sumbul, P. C. Knag, and R. K. Krishnamurthy, "A 4096-neuron 1M-synapse 3.8-pJ/SOP spiking neural network with on-chip STDP learning and sparse weights in 10-nm FinFET CMOS", *IEEE Journal of Solid-State Circuits*, vol. 54, no. 4, pp. 992–1002, 2018.

[135] S.-G. Cho, E. Beigné, and Z. Zhang, "A 2048-neuron spiking neural network accelerator with neuro-inspired pruning and asynchronous network on chip in 40nm CMOS", in *2019 IEEE Custom Integrated Circuits Conference (CICC)*, IEEE, 2019, pp. 1–4.

[136] J. Park, J. Lee, and D. Jeon, "7.6 A 65nm 236.5 nJ/classification neuromorphic processor with 7.5% energy overhead on-chip learning using direct spike-only feedback", in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, IEEE, 2019, pp. 140–142.

[137] J. Pei, L. Deng, S. Song, *et al.*, "Towards artificial general intelligence with hybrid Tianjic chip architecture", *Nature*, vol. 572, no. 7767, pp. 106–111, 2019.

[138] Y. Kuang, X. Cui, Y. Zhong, *et al.*, "A 64K-neuron 64M-1b-synapse 2.64 pJ/SOP Neuromorphic Chip with All Memory on Chip for Spike-based Models in 65nm CMOS", *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2021.

[139] Y. Zhong, X. Cui, Y. Kuang, K. Liu, Y. Wang, and R. Huang, "A Spike-event-based Neuromorphic Processor with Enhanced On-chip STDP Learning in 28nm CMOS", in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2021, pp. 1–5.

[140] C. Eliasmith and C. H. Anderson, *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems.* MIT press, 2004.

[141] G. Orchard, E. P. Frady, D. B. D. Rubin, *et al.*, “Efficient Neuromorphic Signal Processing with Loihi 2”, in *2021 IEEE Workshop on Signal Processing Systems (SiPS)*, 2021, pp. 254–259.

[142] A. Cassidy, A. G. Andreou, and J. Georgiou, “Design of a one million neuron single FPGA neuromorphic system for real-time multimodal scene analysis”, in *2011 45Th Annual Conference on Information Sciences and Systems*, IEEE, 2011, pp. 1–6.

[143] Q. Wang, Y. Li, B. Shao, S. Dey, and P. Li, “Energy efficient parallel neuromorphic architectures with approximate arithmetic on FPGA”, *Neurocomputing*, vol. 221, pp. 146–158, 2017.

[144] D. Pani, P. Meloni, G. Tuveri, F. Palumbo, P. Massobrio, and L. Raffo, “An FPGA platform for real-time simulation of spiking neuronal networks”, *Frontiers in neuroscience*, vol. 11, p. 90, 2017.

[145] H. Mostafa, B. U. Pedroni, S. Sheik, and G. Cauwenberghs, “Fast classification using sparsely active spiking networks”, in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2017, pp. 1–4.

[146] R. M. Wang, C. S. Thakur, and A. van Schaik, “An FPGA-based massively parallel neuromorphic cortex simulator”, *Frontiers in neuroscience*, vol. 12, p. 213, 2018.

[147] E. D. Adrian and Y. Zotterman, “The Impulses Produced by Sensory Nerve-Endings. Part 2. The Response of a Single End-Organ.”, *The Journal of Physiology*, vol. 61, no. 2, 1926.

[148] S. J. Thorpe, “Spike Arrival Times: A Highly Efficient Coding Scheme for Neural Networks”, *Parallel processing in neural systems*, pp. 91–94, 1990.

[149] T. J. Gawne, T. W. Kjaer, and B. J. Richmond, “Latency: Another Potential Code for Feature Binding in Striate Cortex”, *Journal of neurophysiology*, vol. 76, no. 2, pp. 1356–1360, 1996.

[150] M. A. Montemurro, M. J. Rasch, Y. Murayama, N. K. Logothetis, and S. Panzeri, “Phase-of-Firing Coding of Natural Visual Stimuli in Primary Visual Cortex”, *Current biology*, vol. 18, no. 5, pp. 375–380, 2008.

[151] G. Portelli, J. M. Barrett, G. Hilgen, *et al.*, “Rank Order Coding: A Retinal Information Decoding Strategy Revealed by Large-Scale Multielectrode Array Retinal Recordings”, *Eneuro*, vol. 3, no. 3, 2016.

[152] R. Galambos and H. Davis, “The Response of Single Auditory-Nerve Fibers to Acoustic Stimulation”, *Journal of neurophysiology*, vol. 6, no. 1, pp. 39–57, 1943.

[153] T. W. Margrie and A. T. Schaefer, “Theta Oscillation Coupled Spike Latencies Yield Computational Vigour in a Mammalian Sensory System”, *The Journal of physiology*, vol. 546, no. 2, pp. 363–374, 2003.

[154] N. M. Abraham, H. Spors, A. Carleton, T. W. Margrie, T. Kuner, and A. T. Schaefer, “Maintaining Accuracy at the Expense of Speed: Stimulus Similarity Defines Odor Discrimination Time in Mice”, *Neuron*, vol. 44, no. 5, pp. 865–876, 2004.

[155] P. Lichtsteiner, C. Posch, and T. Delbruck, “A 128×128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor”, *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008.

[156] S. Denève and C. K. Machens, “Efficient Codes and Balanced Networks”, *Nature neuroscience*, vol. 19, no. 3, p. 375, 2016.

[157] E. D. Adrian and Y. Zotterman, “The Impulses Produced by Sensory Nerve Endings: Part 3. Impulses Set up by Touch and Pressure”, *The Journal of physiology*, vol. 61, no. 4, pp. 465–483, 1926.

[158] M. Zhang, Z. Gu, N. Zheng, D. Ma, and G. Pan, “Efficient Spiking Neural Networks With Logarithmic Temporal Coding”, *IEEE Access*, vol. 8, pp. 98 156–98 167, 2020.

[159] H. Hamanaka, H. Torikai, and T. Saito, “Quantized Spiking Neuron with A/D Conversion Functions”, *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, no. 10, pp. 1049–1053, 2006.

[160] N. Kasabov, N. M. Scott, E. Tu, *et al.*, “Evolving Spatio-Temporal Data Machines Based on the NeuCube Neuromorphic Framework: Design Methodology and Selected Applications”, *Neural Networks*, Special Issue on ”Neural Network Learning in Big Data”, vol. 78, pp. 1–14, 2016.

[161] B. Petro, N. Kasabov, and R. M. Kiss, “Selection and Optimization of Temporal Spike Encoding Methods for Spiking Neural Networks”, *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 2, pp. 358–370, 2020.

[162] F. Ponulak and A. Kasinski, “Introduction to spiking neural networks: Information processing, learning and applications.”, *Acta neurobiologiae experimentalis*, vol. 71, no. 4, pp. 409–433, 2011.

[163] B. A. Olshausen and D. J. Field, “Sparse Coding of Sensory Inputs”, *Current opinion in neurobiology*, vol. 14, no. 4, pp. 481–487, 2004.

[164] S. Ahmad and L. Scheinkman, “How Can We Be So Dense? The Benefits of Using Highly Sparse Representations”, 2019. arXiv: `1903.11257`.

[165] J. Hawkins and S. Ahmad, “Why Neurons Have Thousands of Synapses, a Theory of Sequence Memory in Neocortex”, *Frontiers in Neural Circuits*, vol. 10, 2016.

[166] B. Schrauwen and J. Van Campenhout, “BSA, a Fast and Accurate Spike Train Encoding Scheme”, in *Proceedings of the International Joint Conference on Neural Networks*, vol. 4, IEEE Piscataway, NJ, 2003, pp. 2825–2830.

[167] M. Hough, H. De Garis, M. Korkin, F. Gers, and N. E. Nawa, “SPIKER: Analog Waveform to Digital Spiketrain Conversion in ATR’s Artificial Brain (Cam-Brain) Project”, in *International Conference on Robotics and Artificial Life*, Citeseer, 1999.

[168] N. Sengupta, N. Scott, and N. Kasabov, “Framework for Knowledge Driven Optimisation Based Data Encoding for Brain Data Modelling Using Spiking Neural Network Architecture”, in *Proceedings of the Fifth International Conference on Fuzzy and Neuro Computing (FANCCO - 2015)*, V. Ravi, B. K. Panigrahi, S. Das, and P. N. Suganthan, Eds., ser. Advances in Intelligent Systems and Computing, Springer International Publishing, 2015, pp. 109–118.

[169] N. Nuntalid, K. Dhoble, and N. Kasabov, “EEG Classification with BSA Spike Encoding Algorithm and Evolving Probabilistic Spiking Neural Network”, in *Neural Information Processing*, B.-L. Lu, L. Zhang, and J. Kwok, Eds., vol. 7062, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 451–460.

[170] Y. LeCun, “The MNIST Database of Handwritten Digits”, *http://yann. lecun. com/exdb/mnist/*, 1998.

[171] T. Delbrück and C. A. Mead, “Analog VLSI adaptive logarithmic wide-dynamic-range photoreceptor”, in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 4, 1994, pp. 339–342.

[172] T. Pellegrini, R. Zimmer, and T. Masquelier, “Low-activity supervised convolutional spiking neural networks applied to speech commands recognition”, in *2021 IEEE Spoken Language Technology Workshop (SLT)*, IEEE, 2021, pp. 97–103.

[173] 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, 10 2021.

[174] 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.

[175] B. Yin, F. Corradi, and S. M. Bohté, “Effective and Efficient Computation with Multiple-timescale Spiking Recurrent Neural Networks”, in *International Conference on Neuromorphic Systems 2020*, ser. ICONS 2020, New York, NY, USA: Association for Computing Machinery, 2020, pp. 1–8.

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

[177] M. Lukoševičius and H. Jaeger, “Reservoir computing approaches to recurrent neural network training”, *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[178] M. Bear, B. Connors, and M. A. Paradiso, *Neuroscience: Exploring the Brain, Enhanced Edition*, 004 Edition. Burlington: JONES & BARTLETT PUB INC, 2020, 975 pp.

[179] S. K. Esser, P. A. Merolla, J. V. Arthur, *et al.*, “Convolutional Networks for Fast, Energy-Efficient Neuromorphic Computing”, *Proceedings of the National Academy of Sciences*, vol. 113, no. 41, pp. 11 441–11 446, 2016. arXiv: 1603.08270.

[180] E. Yılmaz, O. B. Gevrek, J. Wu, Y. Chen, X. Meng, and H. Li, "Deep convolutional spiking neural networks for keyword spotting", *Proc. Interspeech 2020*, pp. 2557–2561, 2020.

[181] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

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

[183] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[184] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect", in *International Conference on Machine Learning*, PMLR, 2013, pp. 1058–1066.

[185] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, "Predicting parameters in deep learning", in *Advances in Neural Information Processing Systems*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., vol. 26, Curran Associates, Inc., 2013.

[186] H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow", in *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc., 2020, pp. 6377–6389.

[187] G. Bellec, D. Kappel, W. Maass, and R. Legenstein, "Deep rewiring: Training very sparse deep networks", 2017. arXiv: `1711.05136`.

[188] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression", 2017. arXiv: `1710.01878`.

[189] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks.", *J. Mach. Learn. Res.*, vol. 22, no. 241, pp. 1–124, 2021.

[190] Q. Yu, H. Tang, K. C. Tan, and H. Yu, "A Brain-Inspired Spiking Neural Network Model with Temporal Encoding and Learning", *Neurocomputing*, vol. 138, pp. 3–13, 2014.

[191] C. Banbury, C. Zhou, I. Fedorov, *et al.*, "MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers", *Proceedings of Machine Learning and Systems*, vol. 3, pp. 517–532, 2021.

[192] S. Furui, *Digital Speech Processing: Synthesis, and Recognition.* CRC Press, 2018.

[193] D. Purves, Ed., *Neuroscience*, Sixth edition. New York: Oxford University Press, 2018, 1 p.

[194] X. Wang, T. Lu, D. Bendor, and E. Bartlett, "Neural coding of temporal information in auditory thalamus and cortex", *Neuroscience*, From Cochlea to Cortex: Recent Advances in Auditory Neuroscience, vol. 154, no. 1, pp. 294–303, 2008.

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

[196] A. Kern, C. Heid, W.-H. Steeb, N. Stoop, and R. Stoop, "Biophysical parameters modification could overcome essential hearing gaps", *PLoS computational biology*, vol. 4, no. 8, e1000161, 2008.

[197] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences", *IEEE transactions on acoustics, speech, and signal processing*, vol. 28, no. 4, pp. 357–366, 1980.

[198] B. Logan, "Mel frequency cepstral coefficients for music modeling", in *In International Symposium on Music Information Retrieval*, Citeseer, 2000.

[199] T. Ganchev, N. Fakotakis, and G. Kokkinakis, "Comparative evaluation of various MFCC implementations on the speaker verification task", in *Proceedings of the SPECOM*, vol. 1, 2005, pp. 191–194.

[200] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks", in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2013, pp. 6645–6649.

[201] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, and G. Penn, "Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition", in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2012, pp. 4277–4280.

[202] K. Kumatani, S. Panchapagesan, M. Wu, *et al.*, "Direct modeling of raw audio with DNNS for wake word detection", in *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2017, pp. 252–257.

[203] T. N. Sainath, R. J. Weiss, K. W. Wilson, *et al.*, "Multichannel signal processing with deep neural networks for automatic speech recognition", *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 5, pp. 965–979, 2017.

[204] J. Lee, J. Park, K. L. Kim, and J. Nam, "Sample-level deep convolutional neural networks for music auto-tagging using raw waveforms", 2017. arXiv: `1703.01789`.

[205] T. Kim, J. Lee, and J. Nam, "Comparison and analysis of samplecnn architectures for audio classification", *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 2, pp. 285–297, 2019.

[206] S. Sheik, M. Coath, G. Indiveri, S. L. Denham, T. Wennekers, and E. Chicca, "Emergent auditory feature tuning in a real-time neuromorphic VLSI system", *Frontiers in neuroscience*, vol. 6, p. 17, 2012.

[207] V. Chan, S.-C. Liu, and A. van Schaik, “AER EAR: A matched silicon cochlea pair with address event representation interface”, *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 1, pp. 48–59, 2007.

[208] S. Liu, A. van Schaik, B. A. Mincti, and T. Delbruck, “Event-Based 64-Channel Binaural Silicon Cochlea with Q Enhancement Mechanisms”, in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, 2010, pp. 2027–2030.

[209] B. U. Pedroni, S. Sheik, H. Mostafa, S. Paul, C. Augustine, and G. Cauwenberghs, “Small-footprint spiking neural networks for power-efficient keyword spotting”, in *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, IEEE, 2018, pp. 1–4.

[210] P. Blouw, X. Choo, E. Hunsberger, and C. Eliasmith, “Benchmarking keyword spotting efficiency on neuromorphic hardware”, in *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*, 2019, pp. 1–8.

[211] P. Warden, “Speech commands: A dataset for limited-vocabulary speech recognition”, 2018. arXiv: 1804.03209.

[212] J. J. Hopfield and C. D. Brody, “What is a moment?“Cortical” sensory integration over a brief interval”, *Proceedings of the National Academy of Sciences*, vol. 97, no. 25, pp. 13 919–13 924, 2000.

[213] D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout, “Isolated word recognition with the liquid state machine: A case study”, *Information Processing Letters*, vol. 95, no. 6, pp. 521–528, 2005.

[214] S. Loiselle, J. Rouat, D. Pressnitzer, and S. Thorpe, “Exploration of Rank Order Coding with Spiking Neural Networks for Speech Recognition”, in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 4, IEEE, 2005, pp. 2076–2080.

[215] J. J. Wade, L. J. McDaid, J. A. Santos, and H. M. Sayers, “SWAT: A spiking neural network training algorithm for classification problems”, *IEEE Transactions on Neural Networks*, vol. 21, no. 11, pp. 1817–1830, 2010.

[216] S. G. Wysoski, L. Benuskova, and N. Kasabov, “Evolving spiking neural networks for audiovisual information processing”, *Neural Networks*, vol. 23, no. 7, pp. 819–835, 2010.

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

[218] A. Tavanaei and A. S. Maida, “A spiking network that learns to extract spike signatures from speech signals”, *Neurocomputing*, vol. 240, pp. 191–199, 2017.

[219] A. Tavanaei and A. S. Maida, “Training a Hidden markov model with a Bayesian spiking neural network”, *Journal of Signal Processing Systems*, vol. 90, no. 2, pp. 211–220, 2018.

[220] G. Bellec, F. Scherr, A. Subramoney, *et al.*, “A solution to the learning dilemma for recurrent networks of spiking neurons”, *Nature Communications*, vol. 11, no. 1, p. 3625, 1 2020.

[221] B. Kim, S. Chang, J. Lee, and D. Sung, “Broadcasted Residual Learning for Efficient Keyword Spotting”, 2021. arXiv: 2106.04140.

[222] A. Berg, M. O’Connor, and M. T. Cruz, “Keyword Transformer: A Self-Attention Model for Keyword Spotting”, 2021. arXiv: 2104.00769.

[223] G. Rogez, J. S. Supancic, and D. Ramanan, “Understanding everyday hands in action from rgb-d images”, in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 3889–3897.

[224] C. Zimmermann and T. Brox, “Learning to estimate 3d hand pose from single rgb images”, in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 4903–4911.

[225] M. E. Benalcázar, C. Motoche, J. A. Zea, *et al.*, “Real-time hand gesture recognition using the Myo armband and muscle activity detection”, in *2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM)*, IEEE, 2017, pp. 1–6.

[226] J. Craley, T. S. Murray, D. R. Mendat, and A. G. Andreou, “Action recognition using micro-Doppler signatures and a recurrent neural network”, in *2017 51st Annual Conference on Information Sciences and Systems (CISS)*, IEEE, 2017, pp. 1–5.

[227] Z. Zhang, Z. Tian, and M. Zhou, “Latern: Dynamic continuous hand gesture recognition using FMCW radar sensor”, *IEEE Sensors Journal*, vol. 18, no. 8, pp. 3278–3289, 2018.

[228] B. Dekker, S. Jacobs, A. S. Kossen, M. C. Kruithof, A. G. Huizing, and M. Geurts, “Gesture recognition with a low power FMCW radar and a deep convolutional neural network”, in *2017 European Radar Conference (EURAD)*, IEEE, 2017, pp. 163–166.

[229] S. Wang, J. Song, J. Lien, I. Poupyrev, and O. Hilliges, “Interacting with soli: Exploring fine-grained dynamic gesture recognition in the radio-frequency spectrum”, in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, 2016, pp. 851–860.

[230] M. Scherer, M. Magno, J. Erb, P. Mayer, M. Eggimann, and L. Benini, “TinyRadarNN: Combining spatial and temporal convolutional neural networks for embedded gesture recognition with short range radars”, *IEEE Internet of Things Journal*, 2021.

[231] J. Lien, N. Gillian, M. E. Karagozler, *et al.*, “Soli: Ubiquitous gesture sensing with millimeter wave radar”, *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, pp. 1–19, 2016.

[232] E. Hayashi, J. Lien, N. Gillian, *et al.*, “RadarNet: Efficient Gesture Recognition Technique Utilizing a Miniature Radar Sensor”, in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–14.

[233] H. Winner, S. Hakuli, F. Lotz, and C. Singer, *Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten Und Systeme Für Aktive Sicherheit Und Komfort (3., Überarbeitete Und Ergänzte Auflage Aufl.)* Wiesbaden: Springer Vieweg, 2015.

[234] S. Skaria, A. Al-Hourani, M. Lech, and R. J. Evans, “Hand-gesture recognition using two-antenna Doppler radar with deep convolutional neural networks”, *IEEE Sensors Journal*, vol. 19, no. 8, pp. 3041–3048, 2019.

[235] S. Ahmed, K. D. Kallu, S. Ahmed, and S. H. Cho, “Hand gestures recognition using radar sensors for human-computer-interaction: A review”, *Remote Sensing*, vol. 13, no. 3, p. 527, 2021.

[236] D. Banerjee, S. Rani, A. M. George, *et al.*, “Application of spiking neural networks for action recognition from radar data”, in *2020 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2020, pp. 1–10.

[237] I. J. Tsang, F. Corradi, M. Sifalakis, W. Van Leekwijck, and S. Latré, “Radar-Based Hand Gesture Recognition Using Spiking Neural Networks”, *Electronics*, vol. 10, no. 12, p. 1405, 12 2021.

[238] F. Kreutz, P. Gerhards, B. Vogginger, K. Knobloch, and C. G. Mayr, “Applied Spiking Neural Networks for Radar-based Gesture Recognition”, in *2021 7th International Conference on Event-Based Control, Communication, and Signal Processing (EBCCSP)*, IEEE, 2021, pp. 1–4.

[239] L. Van der Maaten and G. Hinton, “Visualizing data using t-SNE.”, *Journal of machine learning research*, vol. 9, no. 11, 2008.

[240] J. W. Cooley and J. W. Tukey, “An algorithm for the machine calculation of complex Fourier series”, *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[241] S. G. Johnson and M. Frigo, “A modified split-radix FFT with fewer arithmetic operations”, *IEEE Transactions on Signal Processing*, vol. 55, no. 1, pp. 111–119, 2006.

[242] J. López-Randulfe, T. Duswald, Z. Bing, and A. Knoll, “Spiking Neural Network for Fourier Transform and Object Detection for Automotive Radar”, *Frontiers in Neurorobotics*, vol. 15, 2021.

[243] V. J. Reddi, C. Cheng, D. Kanter, *et al.*, “Mlperf inference benchmark”, in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2020, pp. 446–459.