Efficient and Accurate Spiking Neural Networks

Bojian Yin

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Yin, B

Efficient and Accurate Spiking Neural Networks by Bojian Yin.
Eindhoven: Technische Universiteit Eindhoven, 2022. Proefschrift.

Cover design by DALL·E 2

A catalogue record is available from the Eindhoven University of Technology and Centrum Wiskunde & Informatica Library

ISBN 978-90-386-5629-8

Keywords: Spiking Neural Network, Efficient Learning

The work in this thesis has been sponsored by Imec.

Efficient and Accurate Spiking Neural Networks

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op woensdag 14 december 2022 om 11:00 uur

door

Bojian Yin

geboren te China

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

Voorzitter:	
Prof.dr.ir. P.G.M. (Peter) Baltus	
Promotoren:	
Prof. dr. S.M. Bohté	Centrum Wiskunde & Informatica Universiteit van Amsterdam Rijksuniversiteit Groningen
Prof.dr. H. Corporaal	
Co-promotor:	
dr. F. Corradi	
Promotiecommissieleden:	
Prof.dr.ir. A. de Vries	
Prof.dr. E. Chicca	Rijksuniversiteit Groningen
dr.ing. F. Zenke	University of Basel
Prof. dr. M.A.J. van Gerven	Radboud University Nijmegen
dr.ir. Y.B. van de Burgt	

Het onderzoek of ontwerp dat in deze thesis wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Contents

Chapter 1

Introduction

Research in artificial intelligence has made tremendous strides in recent years based on developments in the capacities and architectures of neural networks, from simple handwritten number recognition [1] to nuclear reactor control [2], and from multilayer perceptrons (MLPs) to over thousand-layers transformers [3]. This construction of larger and deeper neural networks has been driven by the increasing complexity of the problems to which they have been applied. Increasing the size of neural networks increases their number of parameters, which consequently increases the computational and energy requirements for model training and inference [4]. As a result, large neural networks almost invariably must be trained and deployed on supercomputers or in the cloud. However, such in-cloud training of neural networks based on large amounts of personal data has privacy risks and thus may prohibit the provision of personalised network services. A potential solution to this problem is to develop more energy-efficient, robust, and hardware-friendly neural network models that can function with low-latency on increasingly popular wearable smart devices, as this would facilitate more private use of large amounts of personal information [5, 6].

We look to the human brain, the most powerful learning machine in the world, for inspiration. The brain is the culmination of millions of years and is the physical basis of most biological intelligence. It is composed of approximately 100 billion neurons [7] that use electrical pulses called spikes, or action potentials, to communicate sparsely and robustly via trillions of junctions called synapses (Figure 1.3). These neurons and synapses form a large-scale

entangled, multilayer network structure that consists of increasingly abstract features stacked on top of each other, which realises increasingly complicated functions [8, 9].

In the brain, information is continuously routed between network layers, and spikes flow in complex circuits, with information in the nervous system typically encoded by the frequency and timing of these spikes [10]. At a microscopic level, neurons accumulate action potentials and "fire" spikes of electrical signals. These spikes are initiated near the neuron cell body and transmitted along extensions of its body, called axons, which connect to the next neuron (Figure 1.3). Once a spike reaches the end of an axon, neurotransmitters (small molecules) are released to transmit the message to the connected neuron via a junction called a synapse. Thus, neurons encode neurotransmitter signals into spikes to transmit to their connected neurons via synapses [11]. This process is termed synaptic transmission and occurs only when input spikes are received at the synapse. The resulting in event-based approach allows low-precision spikes to efficiently and robustly transmit accurate information throughout the nervous system.

The efficient communication mechanism of the biological nervous system also means it has an extraordinary ability to learn new skills and adapt to new environments. In particular, our brain learns new knowledge and skills by tuning the weight and properties of the connections between neurons in its network (i.e. the strength of synapses). Moreover, during learning, the time difference between pre- and post-synaptic spikes can be used to adjust the weight of a synaptic connection [12]. It follows that by utilising sparse temporal local pulses to dynamically modulate the strength of a connection in a sparse neural network [13], it should be possible to solve contextual or abstract problems in a event-based manner that consumes ultra-low levels of power, like our brain [14].

Brain-inspired neural networks, especially spiking neural networks (SNNs), simulate the activity of neural networks in the brain by abstracting the activity patterns of biological nervous systems [15, 16]. SNNs use mathematical models of simplified biological neurons to construct multilayer neural networks that ideally approximate or exceed the high accuracy of artificial networks with the high energy-efficiency of biological networks. The neurons in classical artificial neural networks (ANNs) are bio-inspired but abstract from the brain at a more coarse level, with each artificial neuron principally modeling the joint activity in a pool of biological spiking neurons [17] (see Fig 1.1). The binary and sparse nature of communication between spiking neurons in principle enables more energy efficient computation compared to ANNs. In ANNs, the contribution

from one neuron to another requires a Multiply-Accumulate (MAC) for every timestep, multiplying each input activation with the respective weight before adding to the internal sum. In contrast, for a spiking neuron a transmitted spike requires only a cheap Accumulate (AC) at the target neuron, adding the weight to the potential, and where spike inputs may be quite sparse. As calculating MACs is much more energetically expensive compared to ACs (e.g., 31x on 45nm CMOS [18]), the relative efficiency of SNNs is determined by the number of connections times activity sparsity and the spiking neuron model complexity. Thus, SNNs have the potential to be incorporated into edge devices to equip them with energy-efficient artificial intelligence [19, 20]. However, the performance of current SNNs has remained lacking compared to artificial neural networks (ANNs).

This thesis aims to contribute to the effective training of large-scale, energy-efficient, and accurate spiking neural networks that are applicable to complex tasks. As such, it considers the training of SNNs from the perspective of current well-established deep learning techniques, with a focus on advanced learning algorithms.

1.1 Background

Artificial neural networks attempt to understand the underlying computations that occur in the dense network of interconnected neurons making up the nervous system, and apply these knowledge to solve complex real-world problems. Originally, in 1943, McCulloch and Pitts took inspiration from the human nervous system to establish the fundamental concept of artificial neural networks (ANNs) [17]. An ANN consists of a network of connected neural units typically organized into layers: an input layer, one or more hidden layers, and an output layer. Each neural unit or artificial neuron is a node in a network; it is connected to another node with an associated weight. The earliest ANNs were perceptrons and were composed of single-layer networks comprising of simple binary neurons [21, 22] (see Figure 1.1).

A perceptron uses a weighted sum to simulate the information integration function of a neuron and emits a binary output when its input surpasses a certain threshold. Subsequently, the ability of perceptrons to solve complex problems was discovered, which resulted in an intense period of research. This was however abruptly halted when 1969, Minsky and Paper shown that a single-layer perceptron could not even solve the simple “XOR” problem [24], and similar non-linearly sparable classification problems by extension.

In more recent years, the development of mathematical tools has facilitated a resurgence in the use of ANNs and artificial intelligence. In 1885, the Stone–Weierstrass theorem was developed [25]; this theorem shows that a polynomial function can uniformly approximate any continuous function defined on a closed interval. More than a hundred years later, the universal approximation theorem was devised [23]; this theorem extends the Stone–Weierstrass theo-

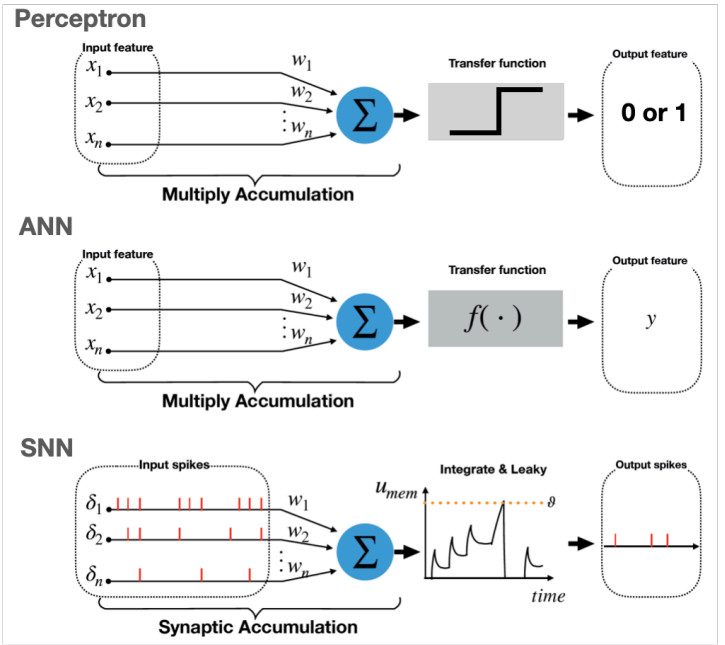


Figure 1.1: Neural Network Architectures: **Top**: A classical perceptron [21, 22] neural unit computes a weighted sum over input activations and then computes an output activation from this sum using a step function. Time is modelled as an iterated recomputation of the network graph. **Middle**: A classical artificial neural [23] unit computes a weighted sum over input activations and then computes an output activation from this sum using a non-linear transfer function $f()$. **Bottom**: Spiking neurons [15] receive spikes that are weighted and added to their internal state (membrane potential), which further develops over time following a pattern that can be described by differential equations. When the membrane potential of spiking neurons crosses a threshold, they emit a spike and their potential is then reset.

rem by proving that an ANN can approach any continuous functions whose input and output are both in Euclidean space. In parallel, Hinton, Rumelhart, and Williams [26] developed the error backpropagation algorithm for training ANNs. The core task of this algorithm (which is detailed in the next section) is to perfectly backpropagate gradients through an ANN, which it achieves by applying a continuous and differentiable function instead of the threshold-based discrete activation function used in Perceptrons (Figure 1.1). In the 21st century, advancements in information technology and microelectronics have led to the construction of high-performance parallel computing devices for training ANNs, which has ushered in another intense period of research.

Several typical ANNs are introduced in this section.

1.1.1 Feedforward ANNs

ANNs are composed of a set of artificial neurons (nodes), inspired by a simplified biological neuron model, that compute a nonlinear weighted sum of input vectors (see Figure 1.1). They generally have the following form:

$$h_j = f(W_x^j x + b_j), j \in 1,2,...,m, \tag{1.1}$$

where h_j is the activation value of an output neuron j, $x = \{x_i | i = 1,2,...n, x_i \in \mathbb{R}^n\}$ is the input vector, $f : \mathbb{R} \to \mathbb{R}$ is a piece-wise differentiable nonlinear activation function such as $ReLU(x) = \max(0, x)$, $W_x^j \in \mathbb{R}^{1\times n}$ is the connection strength (synaptic weight) controlling the contribution of connections between input neurons to the output neuron j, and b_j is a scalar bias.

ANNs that serve as feedforward networks are structured as multiple connected layers of nonlinear neurons. When an input x is fed into a typical ANN with L layers, the ANN generates a prediction $\hat{y}$ by performing the forward propagation culation:

$$\hat{y} = W_L...f(W_l...f(W_2 f(w_1 x + b_1) + b_2)... + b_l)... + b_L, \tag{1.2}$$

where, if only using the linear activation function f, $\hat{y}$ can only approximate the linear features in the data. However, ANNs are designed to learn the implicit and nonlinear relationship between input data and desired outputs, given that they use nonlinear activation functions, $f(\cdot)$. The greater the nonlinearity of an ANN, the more complex decisions it can make [23]. Therefore, increasing the number of layers in ANNs enables them to generate more powerful representations of nonlinear and high-dimensional features [27]. However, overfitting

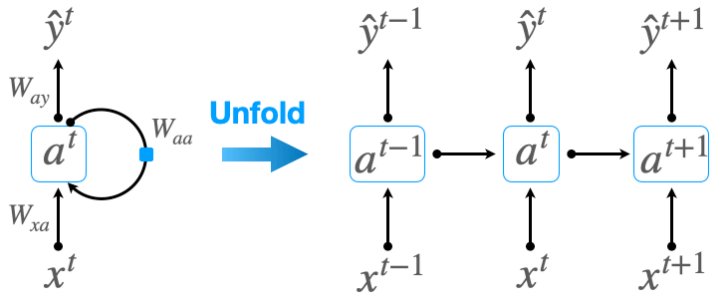


Figure 1.2: Recurrent Neural Networks. Compressed (left) and unfolded (right) basic recurrent neural network. When a recurrent neural network received an input x^t, the instantaneous activation a^t and the prediction $\hat{y}_t$ are updated following Eq 1.3 and 1.4. The blue square is the recurrent weight W_{aa}

problems[1] can occur if an ANN has too many parameters, i.e., it is too deep or wide [28].

Moreover, although a feedforward network structure can generally learn the implicit spatial features in an input, none of their feature computations involving a time dimension. Therefore, feedforward ANNs cannot learn temporal characteristics in time series [2].

1.1.2 Recurrent Neural Networks (RNNs)

To overcome the above-mentioned disadvantages of feedforward ANNs, Recurrent Neural Networks (RNNs) are used to capture the temporal information hidden in time series. Thus, unlike feedforward networks, which read all of their input data simultaneously, RNNs read their input data sequentially. In addition, some RNNs can store historical data in hidden variables, which function as memory units. This enables these RNNs to memorise and summarise temporal information hidden in a given sequence of input data. Many advanced RNNs have been devised to solve problems using different-scale memories, such as the long short-term memory (LSTM) network [30] and the gated recurrent unit

[1] Overfitting is a modeling error in statistics that occurs when the model learns both data and noise in the training dataset to the extent that it poorly generalized on a new dataset.

[2] A Temporal Convolutional Network (TCN) [29] are a novel forward architecture that can be successfully applied on sequence modeling by compute over part or all of the sequential inputs.

(GRU) network [31]. Here, a vanilla RNN is taken as an example.

With RNNs, previous outputs can be used as inputs. Thus, for each timestep t, an input $x^t \in \{x^1, x^2, x^3, ..., x^T\}$ flows into the network, and the instantaneous activation a^t and the prediction $\hat{y}_t$ are updated as follows:

$$a^t = f_1(W_{aa}a^{t-1} + W_{xa}x^t + b_a), \tag{1.3}$$

$$\hat{y}^t = f_2(W_{ay}a^t + b_y), \tag{1.4}$$

where a^t are the hidden states that are initialised as zero; $W_{ax}, W_{aa}, W_{ay}, b_a, b_y$ are coefficients that are shared temporally, and f_1, f_2 are activation functions. The computation of an RNN at each time step t depends not only on the input x^t but also on the hidden network variables a^{t-1}. This function theoretically allows a network of a given size to cope with sequences of any length. Thus, for a sequence of length T, an RNN can be unrolled into an ANN with T hidden layers (Figure 1.2): at each moment, the RNN corresponds to a layer of this ANN, and all weights are shared across time.

RNNs are dynamical systems Eq 1.3 can be simplified as follows:

$$a^t = f(a^{t-1}, x^t), t \in \{0, 1, 2, ...T\}. \tag{1.5}$$

From a mathematical point of view, these iterative updates can be considered as Euler discretisations of the continuous transform [32], which suggests that RNNs can be interpreted as discrete or continuous dynamical systems akin to physical waves [33]. This interpretation serves as a scalable computational framework within which to implement the dynamics of biological neuron models, as discussed in Subsection 1.1.5.

Nevertheless, RNNs have some disadvantages. First, as network inputs are sequential, the hidden states of RNNs must be updated to allow inputs to be received. This slows the simulation and training speed of RNNs. Second, RNNs only have access to historical inputs, not future inputs, which limits their performance. Bidirectional RNN solved this issue by adding another hidden layer to propogate information in the backward direction and process them more dynamically. Third, as detailed in Subsection 1.1.5, RNNs experience gradient vanishing or explosion problems and thus struggle to learn long-range dependencies [30, 34, 35].

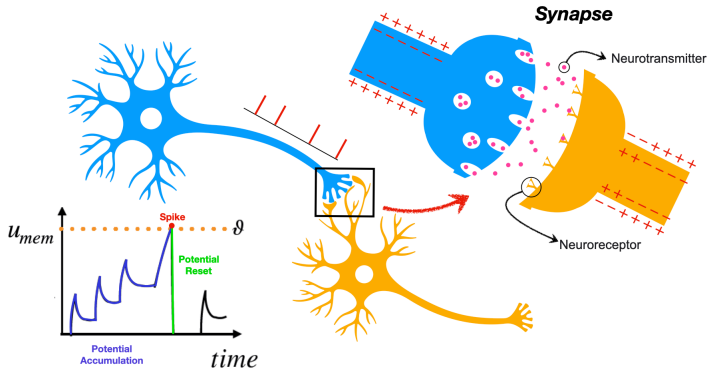


Figure 1.3: A membrane potential is the difference between the voltage inside and outside a membrane. It is converted into an action potential that is transmitted via the axon of a neuron to the dendrites of the next neuron via chemical synapses. An action potential also generates postsynaptic potentials that can propagate along the axon of a neuron by summing to generate other action potentials.

1.1.3 Spiking Neural Networks (SNNs)

The brain constantly processes the vast amount of information it acquires from the environment via the senses, and the high-performance neural networks constructed by biological neurons are far more complex than ANNs. In particular, neurons connect to each other via synapses and communicate with each other using electrical pulses known as spikes, which leads to a binary and event-based representation of information in the brain. In contrast, artificial neurons in ANNs/RNNs use floating-point numbers [3] to iteratively transmit information.

A biological neuron can be described by a highly complex nonlinear dynamic system that stores its state, for example, in the form of membrane potential (i.e., the difference between the voltage inside and outside of its cell membrane). When the membrane potential of a neuron reaches a specific threshold, it sends a spike to the neurons to which it is connected. When this spike reaches a synapse, it is converted via a series of biochemical reactions into a chemical signal that transfers the signal to the connected neuron. This process encodes in-

[3] A recent study [36] shows that the optimzied DNNs are also flexible in numerical precision during inference.

formation in the SNN as the firing frequency or time difference between spikes. A typical neuron in the human brain fires at frequencies of less than 1 Hz (one spike per second) to 10 Hz [37]. However, it can also fire at a frequency greater than 100 Hz when necessary [38]. In comparison, the population average firing rate in the rat cortex is 1 to 5 Hz in vivo [39]. Spikes thus serve as a sparse form of information representation, which is more robust and energy-efficient than other forms of information representation.

SNNs [15] are networks constructed using models of spiking neurons that mimic characteristics of biological neurons. These spiking neurons use abstract mathematical models of biological neurons rather than the simple nonlinear weighted sums used in ANNs. Synaptic strength (or weight) is the strength of the connection between neurons, which means that it functionally represents the contribution of a pre-synaptic neuron to a post-synaptic neuron. Each spiking neuron in an SNN is a modelled cell that converts a sequence of input spikes into an output spike, where the delayed response to the input represents the nonlinearity of a given bio-realistic neuron. Section 1.1.4 describes how the dynamics of spiking bio-realistic neurons contribute to various nonlinear functions. As SNNs encode the information in the time of binary spikes, they have the potential to exhibit an energy advantage over ANNs and RNNs in computation and communication tasks, as further discussed in Chapter 2.

1.1.4 Neural Dynamics

Neuronal function is the result of a complex dynamic system and is a source of natural intelligence. The sophisticated mechanisms embedded in the dynamics of neurons must be characterised and then incorporated into SNNs to enable their accuracy and energy efficiency to be improved [40]. This section briefly introduces various neuronal models and the components of neurons that affect their nonlinearity.

Neurons are highly complex

The physiological structure of the brain is the basis of human intelligence and enables humans to accomplish complex tasks. Although ANNs have already reached or surpassed the performance of humans in specialised tasks, the human brain consumes only ~20 W [14] of power to perform these specialised tasks, far less than the power consumed by (super)computer-based processing of ANNs. This high energy efficiency is attributable to the greater complexity of

biological nervous systems compared with ANNs. The complexity of biological nervous systems is the result of three key characteristics, as described below.

1. **The diversity of neurons.** The brain contains a wide variety of biological neurons, which differ in size, shape and nature. This further results in varying speed of response to input, sensitivity to noise, and computational complexity. These differences empower our brain to deconstruct complex information and handle diverse tasks, and thus, specific neurons are responsible for each type of neural activity.

2. **The high complexity of individual neurons.** An individual neuron is a cell and therefore is itself a complex system whose dynamic behaviour is affected by many factors, such as temperature and ageing. Recent studies [41] have shown that a large and deep temporal convolutional network with 5–8 layers is required to approximate the activity of a single pyramidal single neuron.

3. **The plasticity and complexity of synapses.** Neuroplasticity is the ability of a neuronal system to modify its structure or activity to modulate its actions in response to intrinsic or extrinsic stimuli. Similarly, a fundamental characteristic of neurons is their synaptic plasticity, i.e., their ability to alter their synaptic strength via complex activity-dependent biochemical mechanisms. Synapses in biological nervous systems are therefore essential to their dynamic functioning, and unlike the fixed connection weights in ANNs, the synaptic strength of biological nervous systems constantly evolves over time. Synaptic strength in this context is the magnitude of the postsynaptic potential generated by a neuron after a single synaptic event, as the strength of a synapse changes according to its activity history and other factors. The synaptic strength of two connected neurons is assumed to result in information storage and thus the creation of memory. In biological neural networks, synapses use their plasticity to build new neural pathways to learn new knowledge. The process of synaptic-strength evolution is closely related to neurological development, learning, and adaptation. The synaptic strength between some neurons can be increased by long-term high-frequency activity, which is called long-term potentiation, or it can be decreased by long-term low-frequency activity, which is called long-term depression.

The development of SNNs is based on the diversity of neurons and the internal dynamics by which they achieve information transmission.

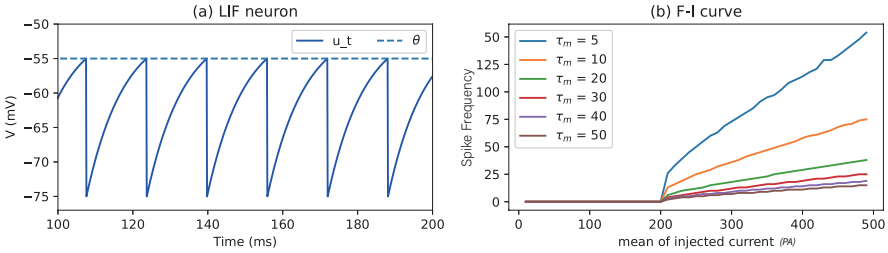


Figure 1.4: To demonstrate the activity of the leaky integrate-and-fire (LIF) neuron model, we define a LIF neuron with a threshold $\theta = -55\,mV$, a time constant $\tau_m = 10\,ms$, and a reset potential $u_r = -75\,mV$. (a) The voltage response of the LIF neuron when receiving a direct current of 300 pA. (b) The input–output transfer function of the LIF neuron to the input current (frequency–current curve) with various time constants τ_m.

A basic spiking neuron: the leaky integrate-and-fire (LIF) neuron model

Figure 1.3 shows that spiking neuron model's internal dynamics are conceptually governed by three functions: 1) differential equations that describe how the membrane potential evolves and the input information is integrated over time; 2) the mechanism that generates the outgoing spike, which converts continuous information into a discrete signal; and, 3) the function that resets the membrane potential after spike emission.

The LIF neuron model is the most simple spiking neuron model and serves as a coarse approximation of the complex dynamics of real neurons. Here, this model is used to elaborate the mechanisms of the above three characteristics of biological nervous systems.

(1) Potential updating As a spiking neuron is a dynamic system, its hidden state – the action potential u_t – evolves over time according to the following linear differential equation [42]:

$$\tau_m \frac{du}{dt} = -(u_t - u_r) + RI_{in}, \tag{1.6}$$

where I_{in} is the input current; u_t is the membrane potential; $R = 1$ is the the linear resistor,u_r is the reset membrane potential; and τ_m is the so-called time

constant of the membrane potential, which is the characteristic time of the decay function. Equation 1.6 illustrates how the membrane potential integrates successive inputs in the time dimension. The membrane potential is updated based on the previous states u_{t-1} and the current input I_{in}.

The time constant τ_m determines the response pattern of the neuron to the input and is the main parameter regulating its nonlinearity. Figure 1.4(b) shows how a smaller τ_m triggers a higher spike-firing frequency in response to the same fixed input current. Each time a neuron sends out a spike, it resets its memory. This means that the length of memory that a LIF neuron can maintain is inversely proportional to the value of τ_m.

The membrane voltage is updated only when an input pulse arrives, such that the neurons' potential updating is event-triggered. Thus, when these input events are sparse, SNNs can be more efficient than RNNs, as the latter, dynamically perform updating at each timestep.

(2) Spike generation A neuron emits an output spike $s_t = 1$ when its membrane potential u_t exceeds the threshold θ. The spike-generation process is defined as a discontinuous nonlinear function f_s:

$$s_t = f_s(u_t, \theta) = \begin{cases} 1, & \text{if } u_t \geq \theta \text{ and if } \dot{u}_t > 0 \\ 0, & \text{otherwise,} \end{cases} \tag{1.7}$$

where f_s thus converts a continuous signal u_t to discrete binary spike s_t. As SNNs are limited by the precision of a single spike, they can perform high-precision information transmission using either the spike train of a single neuron or a group of neurons [43]. This is the feature that potentially allows SNNs to communicate more efficiently and robustly than RNNs.

(3) Potential resetting Biological neurons not only have memory mechanisms but also have forgetting mechanisms. To mimic the forgetting function of a biological neuron, a neuron's hidden state u_t is reset to a certain level u_r – the reset potential – when the neuron triggers a spike ($s_t = 1$). This process can be described as follows:

$$u_t = u_t(1 - s_t) + u_r s_t. \tag{1.8}$$

This denotes a "hard reset" as it forcibly updates the membrane voltage to

the reset voltage. Analogously, a "soft reset" is defined as

$$u_t = u_t - s_t\theta. \tag{1.9}$$

In contrast to a hard reset, a soft reset preserves some memories from previous spikes.

Here, the resetting potential u_r is a constant. However, in some neurons, u_r is dynamic and history-dependent and thus controls the adaptation of the spike rate. This is explained in the next Subsection 1.1.4.

Neurons use a resetting mechanism to remove the traces of previous inputs from their voltage to prepare for saving upcoming memories. The collaboration between different neurons ensures that a network can learn different timescales of input dependencies [44].

To demonstrate the described dynamics, in Figure 1.4(a), the LIF neuron encodes a fixed input current into an output spike at a particular time or into a spike train with a specific firing rate (spikes/s). Figure 1.4(b) shows that the output frequency is nonlinearly proportional to the input intensity, with some delay. The intensity of an input value can thus be represented by neural activities such as the firing frequency or inter-spike-interval [42, 45].

Advanced neural models

In this Subsection, a biological neuron model is introduced as a complex dynamical system. The LIF neuron is a highly simplified neural model as it lacks many properties of biological neurons. Thus, several more complex and biologically plausible neuronal models are described here.

One of the main problems of LIF neurons is that their membrane potential is reset to a certain level after the neuron fires. This results in a LIF neuron not holding any memory of previous spikes. Consequently, a LIF neuron cannot retain long-term memory which is desirable in many applications with long-range dependencies. To circumvent this limitation, we introduce adaption – an important property of biological neurons – into a spiking neuron model.

Bio-plausible spiking neuron In biophysical terms, action potentials are generated by the opening and closing of ion channels in the cell membrane. This complex process cannot be described by a simple linear differential equation, such as that which describes the LIF neuron. A more accurate model of biological neurons was demonstrated by Hodgkin and Huxley in 1952, who measured squid axon currents and described their dynamics using differential equations.

The resulting Hodgkin–Huxley model represents the membrane potential as a function of the input current and activation of various ion channels:

$$C_m \frac{du}{dt} = -\sum_{k\in\{K,Na,l\}} I_k(t) + I_{in}(t) \tag{1.10}$$

$$= -[g_K(u_t - u^K) + g_{Na}(u_t - u^{Na}) + g_l(u_t - u^l)] + I(t) \tag{1.11}$$

$$= -[\bar{g}_K n_t^4(u_t - u^K) + \bar{g}_{Na} m_t^3 h_t(u_t - u^{Na}) + \bar{g}_l(u_t - u^l)] + I(t) \tag{1.12}$$

$$\frac{dp_i}{dt} = \alpha_n(u_t)(1 - p_i) - \beta_{p_i}(u_t), \tag{1.13}$$

where I_{in} is the input current; I_k is the activation of various channels; C_m is the membrane capacitance, g_K and g_K are the potassium and sodium conductances, respectively; u^K and u^{Na} are the reverse membrane potential of potassium and sodium, respectively; $\bar{g}_k$is the maximal value of the channel conductance; the parameters $\bar{g}_l$ and u^l are the leaky conductance and leak potential respectively; and $p_i \in \{n_t, m_t, h_t\}$ represents the probability of gate i being in the permissive state. Hodgkin and Huxley introduced three gating variables – h, m, and n – to mimic the probability of Na^+ and K^+ channels opening at a specific time. The joint cooperation of m and h controls the Na^+ channel, and the K^+ gate is governed by n. Equation 1.12 shows that the sodium conductance is governed by three n-type gates and one h-type gate. Similarly, the potassium conductance is modelled with four n-type gates. Although the Hodgkin–Huxley model accurately models the action potential of neurons using a general equation coupled with three parametric equations, it is computationally complex.

In 2003, the Izhikevich model was introduced. It is a simpler model that combines much of the biological plausibility of the Hodgkin–Huxley model with more of the computational efficiency of the LIF model, and is specified by the following two equations:

$$\tau_m \frac{du}{dt} = (u_t - u_r)(u - \theta) - \omega + I_{in} \tag{1.14}$$

$$\tau_\omega \frac{d\omega}{dt} = a(u_t - u_r) - \omega + b\tau_\omega \sum_{t^f} \delta(t - t^f), \tag{1.15}$$

where τ_ω is the time constant for adaption, and t^f is the previous firing time. As this models a spiking neuron, both membrane potential u_t and ω_t evolve over time. Then, when the neuron fires, the membrane potential u_t is reset to u_r $(u_t \geq \theta)$. The Dirac-delta $\delta(t - t^f) = \begin{cases} 1 & \text{if } t = t^f, \\ 0 & \text{otherwise} \end{cases}$ function implies

that the adaption current ω increases by an amount b during neuron firing. Subthreshold and spike-triggered adaption are entangled in adaptive spiking neurons. Subthreshold adaption results from coupling of the adaptive current ω and the potential u_t and is dominated by the parameter a, whereas spike-triggered adaption is activated when a neuron fires and is determined by the parameters a and b.

The adaptive LIF (ALIF) neuron model The adaptive nature of biological neurons is partly due to the dynamics of neurotransmitters and synapses. This means that the interval between output pulses varies, even with a fixed input current; in contrast, a fixed input into a LIF neuron results in spikes with a fixed rate. Theoretically, adaption is built on several successive spikes, which means that the contribution of previous spikes to the current spike must be considered. The ALIF neuron model incorporates a new, biologically inspired variable – the adaption current ω – into a neuron's state. Unlike the membrane voltage, ω is not reset when an ALIF neuron fires, which means that the historical information of spikes is integrated into the ALIF neuron's input by the coupling voltage and adaption current. This function allows ALIF neurons to retain longer memories than LIF neurons. More details will be described in Chapter2.

The ALIF neuron is defined by two differential equations:

$$\tau_m \frac{du}{dt} = -(u - u_r) - \sum_k \omega_k + I_{in}, \tag{1.16}$$

$$\tau_k \frac{d\omega_k}{dt} = a(u - u_r) - \omega_k + b_k \tau_k \sum_{t^f} \delta(t - t^f), \tag{1.17}$$

where τ_k is the time constant for adaption in the k channel.

ALIF neurons exploit historical spiking information by introducing adaptive currents. Although this adaption increases computational expenditure, it helps ALIF neurons to maintain a more durable memory than LIF neurons.

1.1.5 Training SNNs as RNNs

The behaviour of biological neurons is determined by a series of complex biochemical reactions that are almost impossible to simulate on a computer. The analysis of numerous experimental and observational data has enabled scientists to model the procedure of neurons sending out spikes as a dynamic system. Spiking neural networks are a type of ANN built using this simplified dynamic

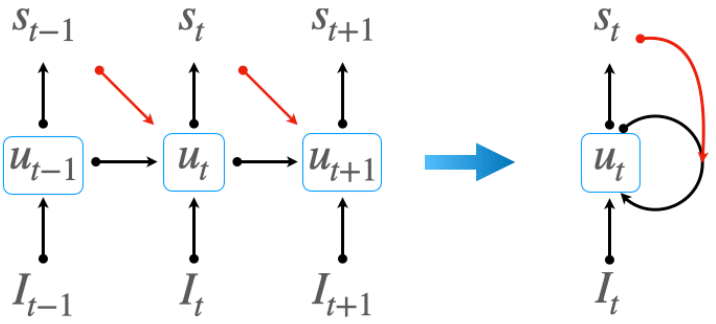


Figure 1.5: Computational graph of a LIF neuron. The red arrow represents the resetting function.

system, which has potential to efficiently solve complex problems. The performance of SNNs however has remained lacking compared to artificial neural networks (ANNs). The best way to train SNNs is therefore an open question. This section describes how to optimize the training of SNNs using modern deep-learning techniques.

Formulating SNNs as RNNs

To numerically simulate the membrane potential dynamics of a LIF neuron, we discretise the continuous dynamics of Eq 1.6 based on its Euler expansion, as follows:

$$\tau_m \frac{\Delta u}{\Delta t} = -(u_t - u_r) + I_{in,t}; u_{t+\Delta t} = u_t + \Delta u. \tag{1.18}$$

A more accurate approximation of the trajectory of Eq 1.6 can be obtained by running the simulation at a higher temporal resolution. The smaller the value of Δt, the more accurate the approximation, and the longer the simulation time. In most cases, to simplify and accelerate the computations, we set $\Delta t = 1$ms [4]. This discretised LIF neuron can then be reformulated as $u_{t+1} = f(u_t, I_{in})$, which is almost the same as the general expression of the recurrent neuron (Eq 1.5).

[4] We typically choose such a value as the time constant of most spiking neuron falls in the range 10~20 ms.

Therefore, we can simulate the discretised spiking neuron as a recurrent neuron with a complex circuit (Figure 1.5).

SNNs also have some other features that distinguish them from RNNs. For example, in SNNs, neurons communicate with each other using binary spikes instead of floating-point numbers. This underpins the lower energy consumption of SNNs compared with RNNs, Chapter 2 explains the specifics in detail. In addition, the computation of neurons in SNNs is event-based: in principle, each neuron in an SNN only needs to update its state when it receives or emits an impulse. Thus, if input is sufficiently sparse, an SNN consumes less power and computes faster than an RNN.

In general, SNNs and RNNs are computationally equivalent, in Figure 1.5, which enables existing training methods to be applied or modified to SNNs to develop new learning algorithms for SNNs. This approach is the conceptual framework of this thesis.

Learning algorithms: From animal learning to network training

The brain modulates the synaptic strength of connections between neurons during the learning process to learn features for classification and optimize decision-making for control purposes. Neurons form a multi-layered network structure in the brain that serves as the human cognitive system. The synaptic strength in this network significantly affects its activity and output, and the problem of measuring the contribution of the synapse to network performance is called the "credit assignment" problem and is complicated by the multilayer structure in the brain. Many details of learning and cognitive functions in the brain remain unknown [46, 47], which means that its efficient learning algorithms cannot yet be applied to train SNNs. In contrast, traditional ANNs have been fully developed with the aid of mathematical tools such as statistics and optimization theories.

In recent years, the Backpropagation (BP) algorithm [26] has been extensively applied for solving the credit assignment problem in the training of deep ANNs. Although it has been argued that BP algorithm is biologically unrealistic [48], it remains the most common algorithm used for the core training of deep neural networks and has been effective in solving a variety of problems. BP adjusts synaptic strength by computing the error signal and flows gradient through backward connections during training. The error gradient of the weight therefore represents the credit of the network connections. Learning algorithms can be classified based on error signal sources as supervised algorithms, unsupervised learning algorithms, and reinforcement learning algorithms.

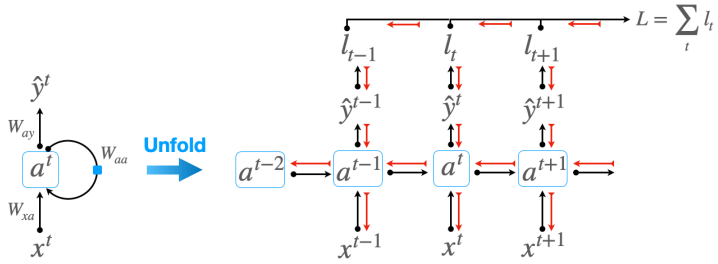


Figure 1.6: Back-propagation through time. The black and red arrows depict forward and backward propagation, respectively.

Supervised learning [49, 50] involves a network being trained under the supervision of labelled data to learn a specific relationship or structure between the input x and the target output y. The learning signal (error signal) arises as a feature of the difference between the well-labelled target y and the network predictions $\hat{y}$. Supervised learning includes classification and regression tasks; in the former, the network maps inputs to discrete category labels, whereas in the latter, the network projects inputs to continuous outputs. **Unsupervised learning** [50,51] differs from supervised learning by not involving signals with labels or desired outputs in the network training. Therefore, the network has to analyse and cluster the hidden information in the unlabelled data x. The resulting ability of unsupervised learning to discover similarities and differences in the data means it can be applied for exploratory learning. **Reinforcement learning** (RL) [52, 53] explores how to evolve an intelligent body to take actions in an environment that maximise the obtained cumulative reward. Unlike supervised learning, RL does not need to learn from any labelled input or output, nor does it need to be supervised by ideal actions. Instead, RL focuses more on learning from the exploration in the environment and using that knowledge to maximize rewards. The network can thus solve the control problem by combining BP with RL in a supervised learning-like approach.

Next, we explore the training of SNNs by focusing on the approaches used to train RNNs. RNNs and SNNs are designed to function as machines that learn the unknown temporal features in training data. Consequently, in network training, the credit assignment problem becomes a spatial and temporal credit assignment problem. Based on a weight-updating approach, SNN training algorithms can be classified as involving either offline learning or online learning, as de-

tailed below.

Offline learning: the back-propagation through time (BPTT) RNNs/SNNs operate as deep neural networks along a temporal expansion over a sequence. Therefore, the BPTT algorithm [54], which is a temporal variant of the BP algorithm, is used to optimize RNNs. The BPTT algorithm unfolds all of the input time-steps (each time-step has one input, one network copy, and one output) and then calculates and accumulates the error l_t at each time-step (Figure 1.6). It also uses gradient descent to adjust the network parameters in the direction of the gradient that reduces the error as effectively as $w_{new} = w_{old} - \frac{\partial L}{\partial w}$, which is the approach used by the BP algorithm in feedforward neural networks. However, in this context, the different gradients of the weights for each moment are summed. Finally, the averaged gradients over all timesteps are used to update the network weights.

In Figure 1.6, the forward path of a state update in an RNN is defined by Eq 1.3. The RNN has an instantaneous loss l_t at each time step, which is calculated based on the current prediction $\hat{y}_t$ and the target label y_t. After reading through the entire sequence, the gradient signal $\frac{\partial L}{\partial w}$ is used to update the parameters.

$$\frac{\partial L}{\partial w} = \frac{1}{T}\sum_{t=1}^{T}\frac{\partial l_t}{\partial w} \tag{1.19}$$

$$= \frac{1}{T}\sum_{t=1}^{T}\frac{\partial l_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial a_t}\frac{\partial a_t}{\partial w} \tag{1.20}$$

$$= \frac{1}{T}\sum_{t=1}^{T}\frac{\partial l_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial a_t}\sum_{i=1}^{t}\left(\prod_{j=i+1}^{t}\frac{\partial a_j}{\partial a_{j-1}}\right)\frac{\partial a_{j-1}}{\partial w} \tag{1.21}$$

The gradient calculation requires that both f_1 and f_2 (in Eq. 1.3) be nonlinear differentiable functions. BPTT is an easy understanding algorithm but its complexity is quadratic to sequence length T.

The training mode in which the network updates weights only after reading the entire sequence is called offline learning. Offline learning – in particular, the BPTT algorithm – is the most widely used and effective training method for RNNs. However, the BPTT algorithm updating mode requires all of the network states to be stored to update the network parameters, which is the main reason why the BPTT algorithm is regarded as biologically problematic [48]. As a

result, offline training is limited by the memory cost of training, which grows linearly with sequence length.

From a computational perspective, two aspects of Eq 1.21 lead to RNNs with low trainability, due to the well-known phenomenon of gradient explosion or vanishing [34, 55]:

$$\left\| \frac{\partial a_j}{\partial a_{j-1}} \right\|^2 < 1 \Rightarrow \text{Gradient Vanishing} \tag{1.22}$$

$$\left\| \frac{\partial a_j}{\partial a_{j-1}} \right\|^2 > 1 \Rightarrow \text{Gradient Explosion} \tag{1.23}$$

In Eq 1.22, the gradient vanishing problem is caused by the term $\frac{\partial L}{\partial w}$ decreasing to zero at an exponential rate, which makes it difficult for the network to learn long-range dependencies. In Eq 1.23, the gradient Explosion problem occurs because the terms $\frac{\partial L}{\partial w}$ exponentially increase to infinity and their values become "NaN."

The truncated BPTT algorithm [56] is a variant of the BPTT algorithm in which the gradient flow is truncated after a constant time period. This accelerates the training and reduces the memory requirements of the algorithm but does not fundamentally eliminate the limitation requiring perfect recall of prior memory. However, truncation only facilitates the short-term dependence in the data because the gradient estimate of the truncated BPTT is biased [57]. Therefore, it benefit the limited degree from the stochastic gradient theory's convergence guarantees.

Online learning Online learning requires algorithms that can update network parameters in real-time based on the current input. Compared with offline algorithms, these algorithms are closer to human brain-learning processes based on temporally local information. Like forward propagation, brain-like online learning algorithms require weight updates that are local in time and space, i.e., only the information of the most recent moments is used. Thus, after reading the entire sequence, a network can use accumulated gradients for weight updates. Alternatively, a network can perform updating only when the gradients are available. The real-time recurrent learning algorithm (RTRL) [58, 59] and the sparse n-step approximation algorithm [60] are online learning variants of the BPTT algorithm that are applied to the training of RNN to reduce memory and computational consumption. Theoretically, the BPTT algorithm and its variants provide the best and most complete gradient signal based on the perfect

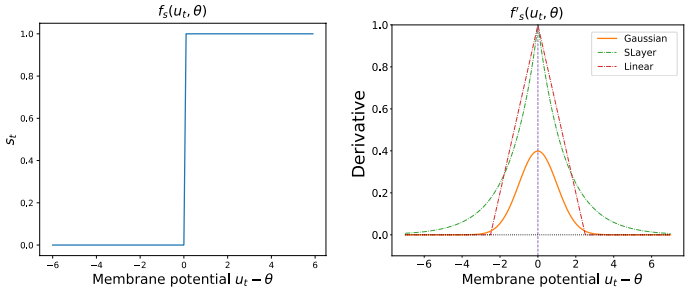


Figure 1.7: Illustration of different surrogate gradient functions $\hat{f}'_s$ as a function of the neuron's membrane potential u_t and threshold θ.

recall of history [54, 61]. RNNs are thus better trained by these offline learning methods than by online algorithms on sequential tasks.

Summary Overall, the quality of the learning signal, in addition to the gradient, determine the performance of an optimized neural network. This thesis therefore thoroughly investigates gradient-based supervised learning of spiking neural networks.

1.2 Challenges when training SNNs

The computational equivalence between SNNs and RNNs implies that SNNs could theoretically be trained with typical RNN learning algorithms. In general, all learning algorithms for RNNs are applicable to SNNs. However, the distinctiveness of SNNs results in the following challenges when training SNNs with RNN learning algorithms. These challenges further lead to SNNs being uncompetitive in performance compared to ANNs and difficult to apply to large-scale networks for complex tasks.

1.2.1 Challenge 1: Discontinuity of the spike generation function

The nonlinear spike generation function $f_s(u_t, \theta)$ is one of the primary sources of nonlinearity in SNNs. It is discontinuous and therefore non-differentiable

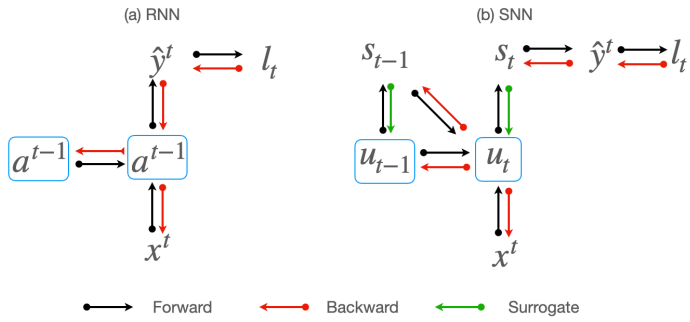


Figure 1.8: Gradient calculation for a spiking neuron

(Figure 1.7 a), which limits the training of SNNs possible by directly adopting the RNN learning algorithms. In 2000, Bohte et al. [62] first introduced surrogate gradients, a function of the neuron's membrane potential u_t and threshold θ, to approximate the derivative of the neural action function $\hat{f}'_s(u_t,\theta)$. Subsequently, SNNs have been developed that can use gradient-based training algorithms like RNNs [63,64]. Gradient-based learning algorithms, in particular the BPTT algorithm, are currently the most popular and effective training method for SNNs [65–68]. Various surrogate gradients [19,65] have presented to train SNNs more deeply and more accurately than other methods.

However, the surrogate gradient $f'_s(u_t,\theta)$ acts as an approximation to the unknown derivative of spike $\frac{\partial s_t}{\partial u_t}$ coming from the non-differentiable spike generation function. Therefore, the error $\|\frac{\partial s_t}{\partial u_t} - f'_s(u_t,\theta)\|^2$ between the surrogate gradient and the ideal gradient affects the performance of the trained networks and their activities. For example, in the BPTT algorithm framework, the approximation error accumulates continuously with the sequence length (Eq 1.21). This accumulation of the gradient error causes SNNs to be more vulnerable to gradient explosion or vanishing than RNNs.

In addition, most current surrogate gradients aim to obtain higher accuracy on benchmarks and do not consider their performance in the context of network sparsity.

1.2.2 Challenge 2: Cumbersome gradient flow

Challenge 1 comprises problems associated with non-differentiable spike-genera -tion functions during SNN training. Similarly, when training SNNs using the BPTT algorithm, there remains the problem of gradient computation due to complex computational loops.

The core of the BPTT algorithm is its learning of the association between network activities in the time dimension, which is denoted as the sum of the product terms, as noted in Eq 1.21:

$$\frac{\partial a_t}{\partial w} = \sum_{i=1}^{t} \left(\prod_{j=i+1}^{t} \frac{\partial a_j}{\partial a_{j-1}} \right) \frac{\partial a_{j-1}}{\partial w} \tag{1.24}$$

in Eq 1.21. SNNs have more complex computation circuits than RNNs in both feedforward- and feedback propagation. In SNNs, hidden states – the spike and membrane potentials – are entangled due to the reset mechanism. The membrane potential is governed by the previous membrane potential and spikes, such that the reset circuit leads to a more complicated gradient calculation in the time direction in SNNs than in RNNs (Figure 1.8). In the SNN, the core part of Eq 1.21 is transformed into

$$\frac{\partial u_t}{\partial w} = \sum_{i=1}^{t} \left[\prod_{j=i+1}^{t} \left(\frac{\partial u_t}{\partial u_{t-1}} + \frac{\partial u_t}{\partial s_{t-1}} \frac{\partial s_{t-1}}{\partial u_{t-1}} \right) \right] \frac{\partial u_{j-1}}{\partial w}; \tag{1.25}$$

In the BPTT algorithm framework, this gradient calculation amplifies the error of the surrogate gradient: the longer the sequence, the greater the effect of the gradient error. As a result, the network cannot effectively learn long-term dependencies in a long sequence.

In sum, the cumbersome gradient computation path increases the difficulty of back-propagating gradients in a deep model. It also obstructs the training of SNNs constructed from more biologically realistic neuronal models with more complex neuronal circuits.

1.2.3 Challenge 3: Memory

In offline learning algorithms, network parameter updates necessitate perfect recall of all of the hidden states during the feedforward process. As a result, the memory demand for network training increases with the sequence length. In

RNNs, only one hidden state must be stored (a_t), whereas spiking neurons generally need to save at least two states – the spike s_t and the membrane potential u_t. Each additional differential equation has at least one new hidden state. For example, the adaptive neuron needs to remember three states: spike s_t, membrane potential u_t, and adaption current ω_t. Thus, the training of SNNs with the BPTT algorithm requires more training memory than with other algorithms, which further limits the training of deep SNNs on long sequences, as further discussed in detail in Chapter 4.

1.2.4 Challenge 4: Continual running

In practice, a well-trained network is deployed on a device to run continuously and accomplish the intended task. In addition to the training problem, SNNs, like RNNs, exhibit accuracy degradation during continuous running [69]. In contrast, the brain can efficiently and continuously process the information flow from the body's sensors. SNNs (as a type of brain-inspired algorithm) and RNNs are also expected to maintain their performance over long testing periods. A model must therefore learn how to switch decisions during optimization on long sequences, so that a network's high performance is maintained during continuous running. Thus, we have evolved the classification problem into a decision-making problem that requires the network to make the correct decision at the correct moment. However, it is difficult for a network to learn effective decision transitions by being trained on short sequences.

In summary, the surrogate gradient-based SNN training in the BPTT algorithm framework suffers from the above-described limitations, which prohibit it from training highly accurate and deep SNN models on long sequences.

1.3 Contributions

SNNs inspired by biological nervous systems hold the potential of having lower competitive accuracy with ANNs/RNNs. Ideally, the effective learning algorithm of the brain would be used to optimize SNNs. However, our knowledge of the brain is currently not sufficient to allow us to similarly train SNNs. Still, given the computational equivalence between SNNs and RNNs, we can use recent advanced deep-learning techniques to perform this task. Nevertheless, there remains a lack of efficient training algorithms that can empower SNNs to obtain accuracies that match or exceed those of traditional ANNs.

In this thesis, we explore in detail how to solve the above-mentioned challenges (in Section 1.2) by exploiting our understanding of the brain and ANNs. This enables SNNs to perform more complex tasks with higher accuracy and lower power consumption.

- **Chapter 2** proposes a solution to *Challenge 1* that enables SNNs to achieve higher accuracy and theoretically demonstrates how that SNNs can be more energy efficient than RNNs.
- **Chapter 3** demonstrates how the truncation of the BPTT learning, T-BPTT algorithm, can be used to weaken the effect of gradient-related error, accelerate the training process, and reduce the memory requirement.
- **Chapter 4** shows how a novel training approach, FPTT, can be successfully solve SNNs training problems described in *Challenges 1,2 and 3* using novel Liquid Spiking Neurons, and presents the first end-to-end training of a large SNN-based object-detection model.
- **Chapter 5** introduces how the incorporation of local signal detection measures with brain-inspired decision circuits enables compact and high performance SRNNs to be applied to continual running scenarios and to solve *Challenge 4*.

Each chapter begins with a summary describing the core ideas of the cited studies and their role in this thesis. In the final chapter, interesting related ideas that have been explored as part of the work for this thesis are first discussed. And related research questions that may merit further investigation are detailed.

Chapter 2

Multi-timescales Spiking Recurrent Neural Networks

Significance: **SNNs are commonly considered as energy efficient networks, but a systematic analysis and comparison of network performance and energy consumption advantages is lacking. Here, we focus on solving *Challenge 1*. By introducing a new surrogate gradient for BPTT and parameterizing the time constants of neurons in SNNs, we demonstrate that SNNs can achieve performance that rivals or even exceeds RNNs on various tasks. We also thoroughly explored the theoretical power consumption advantage gained by SNNs due to their sparse communication where we show that SNNs can be one two three orders more energy efficient compared to equally performaning ANNs.**

This chapter is based on publications "Effective and efficient computation with multiple-timescale spiking recurrent neural networks" [20] **and "Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks" [19].**

Abstract: *Inspired by more detailed modeling of biological neurons, Spiking Neural Networks (SNNs) are investigated both as more biologically plausible and as more performant models of neural computation: the sparse and binary communication between spiking neurons potentially enables more powerful and energy-efficient neural networks. The performance of SNNs however has remained lacking compared to artificial neural networks (ANNs). Here, we demonstrate how an*

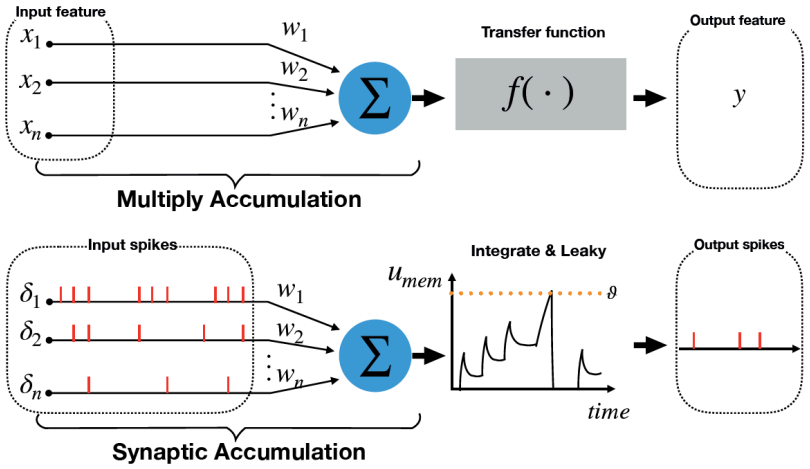


Figure 2.1: Top: a classical artificial neural unit computes a weighted sum over input activations and then computes an output activation from this sum using a non-linear transfer function $f()$. Time is modeled as iterated recomputation of the network graph. Bottom: Spiking neurons receive spikes that are weighted and added to the internal state (membrane potential) that further develops through time following differential equations. When the membrane potential crosses a threshold, a spike is emitted and the potential is reset.

activity-regularizing surrogate gradient combined with recurrent networks of tunable and adaptive spiking neurons yields state-of-the-art for SNNs on challenging benchmarks in the time domain, like speech and gesture recognition. This also exceeds the performance of standard classical recurrent neural networks (RNNs) and approaches that of the best modern ANNs. As these SNNs exhibit sparse spiking, we show that they are theoretically one to three orders of magnitude more computationally efficient compared to RNNs with similar performance. Together, this positions SNNs as an attractive solution for AI hardware implementations.

2.1 Introduction

The success of brain-inspired deep learning in AI is naturally focusing attention back onto those inspirations and abstractions from neuroscience [70]. One such example is the abstraction of the sparse, pulsed and event-based nature of com-

munication between biological neurons into neural units that communicate real values at every iteration or timestep of evaluation, taking the rate of firing of biological spiking neurons as an analog value (Figure 2.1). Spiking neurons, as more detailed neural abstractions, are theoretically more powerful compared to analog neural units [15] as they allow the relative timing of individual spikes to carry significant information. A real-world example in nature is the efficient sound localization in animals like Barn Owls using precise spike-timing [71]. The sparse and binary nature of communication similarly has the potential to drastically reduce energy consumption in specialized hardware, in the form of neuromorphic computing [72].

Since their introduction, numerous approaches to learning in spiking neural networks have been developed [62, 64, 73–75]. All such approaches define how input signals are transduced into sequences of spikes, and how output spike-trains are interpreted with respect to goals, learning rules, or loss functions. For supervised learning, approaches that calculate the gradient of the loss function with respect to the weights have to deal with the discontinuous nature of the spiking mechanism inside neurons. Local linearized approximations like SpikeProp [62] can be generalized to approximate "surrogate" gradients [65], or even calculated exactly in special cases [76]. The use of surrogate gradients in particular has recently resulted in rapidly improving performance on select benchmarks, closing the performance gap with conventional deep learning approaches for smaller image recognition tasks like CIFAR10 and (Fashion) MNIST, and demonstrating improved performance on temporal tasks like

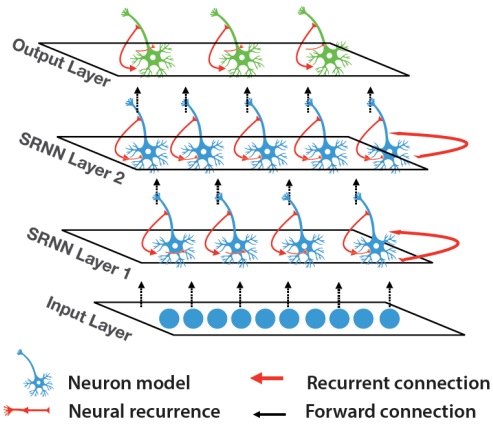


Figure 2.2: Example architecture of a Spiking Recurrent Neural Network: an input layer projects to a layer of recurrently connected spiking neurons. Recurrent layers then project to a read-out layer. Multiple recurrent layers can be connected in a feedforward fashion, here shown for two recurrent layers.

TIMIT speech recognition [77]. Still, spiking neural networks (SNNs) have struggled to demonstrate a clear advantage compared to classical artificial neural networks (ANNs) [78, 79].

Here, we introduce a novel approach to Spiking Recurrent Neural Networks (SRNNs) [20], networks that include recurrently connected layers of spiking neurons (Figure 2.2). We demonstrate how these networks can be trained to high performance on hard benchmarks, exceeding existing state-of-the-art in SNNs on all-but-one benchmark, and approaching or exceeding state-of-the-art in classical recurrent artificial neural networks. The high-performance in SRNNs is achieved by applying back-propagation-through-time (BPTT) [54] to spiking neurons using a novel Multi-Gaussian surrogate gradient and using adaptive spiking neurons where the internal time-constant parameters are co-trained with network weights. The Multi-Gaussian surrogate gradient is constructed to include negative slopes, similar to the gradient of the sigmoid-like dSilu activation function [80, 81]: we find that the Multi-Gaussian surrogate gradient consistently outperforms other existing surrogate gradients. Similarly, co-training the internal time-constants of adaptive spiking neurons proved always beneficial. We demonstrate that these ingredients jointly improve performance to a competitive level while maintaining sparse average network activity.

We demonstrate the superior performance of SRNNs for well-known benchmarks that have an inherent temporal dimension, like ECG wave-pattern classification, speech (Google Speech Commands, TIMIT), radar gesture recognition (SoLi), and classical hard benchmarks like sequential MNIST and its permuted variant. We find that the SRNNs need very little communication, with the average spiking neuron emitting a spike once every 3 to 30 timesteps, depending on the task. Calculating the theoretical energy cost of computation, we then show that in SRNNs, cheap Accumulate (AC) operations dominate over more expensive Multiply-Accumulate (MAC) operations. Based on relative MAC vs. AC energy cost [78, 79], we argue that these sparsely spiking SRNNs have an energy advantage ranging from one to three orders of magnitude over RNNs and ANNs with comparable accuracy, depending on network and task complexity.

2.2 Methods

In the SRNNs, the LIF spiking neuron is modeled as:

$$u_{t-1} = u_{t-1}(1 - S_{t-1}) + u_r S_{t-1} \tag{2.1}$$

$$u_t = u_{t-1}(1 - 1/\tau_m) + R_m I_t/\tau_m \tag{2.2}$$

$$S_t = f_s(u_t, \vartheta) \tag{2.3}$$

where $I_t = \sum_{t_i} w_i \delta(t_i) + I_{inj,t}$ is the input signal comprised of spikes at times t_i weighted by weight w_i and/or an injected current $I_{inj,t}$; u is the neuron's membrane potential which decays exponentially with time-constant τ_m, ϑ is the threshold, R_m is the membrane resistance (which we absorb in the synaptic weights). The function $f_s(u_t, \vartheta)$ models the spike-generation mechanism as function of the threshold ϑ, which is set to 1 when the neuron spikes and otherwise is 0 (where the approximating surrogate gradient is then $\hat{f}_s'(u_t, \vartheta)$). The value for the reset potential u_r was set to zero. The ALIF neuron is similarly modeled as :

$$u_t = \alpha u_{t-1} + (1 - \alpha) R_m I_t - \vartheta S_{t-1} \tag{2.4}$$

$$\eta_t = \rho \eta_{t-1} + (1 - \rho) S_{t-1} \tag{2.5}$$

$$\vartheta = b_0 + \beta \eta_t \tag{2.6}$$

$$S_t = \hat{f}_s(u_t, \vartheta), \tag{2.7}$$

where α, γ are parameters related to the temporal dynamics, $\alpha = \exp(-dt/\tau_m)$ and $\rho = \exp(-dt/\tau_{adp})$, ϑ is a dynamical threshold comprised of a fixed minimal threshold b_0 and an adaptive contribution $\beta\eta_t$; ρ expresses the single-timestep decay of the threshold with time-constant τ_{adp}. The parameter β is a constant that controls the size of adaptation of the threshold; we set β to 1.8 for adaptive neurons as default. Similarly, α expresses the single-timestep decay of the membrane potential with time-constant τ_m.

The SRNNs were trained using BPTT, various spiking neuron models with plastic time-constants and with various surrogate gradients. The standard validation sets were used where available to determine overfitting; for SHD we held out 5% of the training data and for (P)S-MNIST 10%. Apart from the SSC and SHD datasets, analog input values are encoded into spikes either using spikes generated by a level-crossing scheme (ECG) or by directly injecting a proportional current into the first spiking layer (S-MNIST, PS-MNIST, SoLi, TIMIT, GSC). To decode the output of the network, we used one of two methods: either spike-counting over the whole time-window, for the (P)S-MNIST

task, non-spiking LIF neurons (TIMIT, SHD, SoLi, and GSC), or spiking ALIF neurons (ECG). With spike-counting, classification is decoded from the sum of the output spikes as $\hat{y} = \text{softmax}(\sum_t S^t_{i,out})$ where $S^t_{i,out}$ is the spike of the output neuron i at time t. For either non-spiking LIF neurons and spiking ALIF neurons as outputs, a softmax classification is computed from the output neurons' membrane potential $u_{out,t}$ at each timestep as $\hat{y}_t = \text{softmax}(u_{out,t})$. For ECG, we used spiking ALIF neurons for outputs as they performed best, which we believe is related to the fact that this is the only task where classification switches within the sample - the spiking then functions effectively as resets. We use a standard BPTT approach [77] to minimize the cross-entropy (CE) or negative-log-likelihood (NLL) loss for each task using the Adam [82] optimizer, where we unroll all input timesteps from end to the start. The error-gradient is calculated and accumulated through all timesteps after which the weights are updated. BPTT for the spiking neurons is calculated retrogradely along with the self-recurrence circuits. As shown in Figure 2.5, given an input sequence $X = x_0, x_1, x_2, \ldots, x_T$, and a neuron with initial states $\{u_{h,0}, u_{o,0}, S_{h,0}, S_{o,0}\}$, we obtain for each timestep $t \in \{0, T\}$ the spiking neuron states $\{u_{h,t}, S_{h,t}, u_{o,t}, S_{o,t}\}$, where $S_{h,t}$ refers to a neuron firing-or-not in a hidden layer and $S_{o,t}$ to an output neuron (if spiking), and $u_{h,t}$ and $u_{o,t}$ denote hidden and output neurons' membrane potentials. We then obtain a classification $\hat{y}(t)$ either for each timestep or for the whole sequence $\hat{y}$ and an associated loss. In classification tasks with C classes, the prediction probability of class $c - \hat{y}_c$ is computed after having read the whole sequence, and then the loss of the network is calculated as $\mathscr{L} = \sum_{c=1}^{C} y_c \log \hat{y}_c$, where y_c is the target probability of class c. In streaming tasks (ECG, SoLi), the total loss is computed as the sum of the loss at each timestep – $\mathscr{L} = \sum_{t=1}^{T} \mathscr{L}_t$. For the BPTT-derived gradient, we compute $\frac{\partial \mathscr{L}}{\partial z} = \hat{y} - y$, and for recurrent weights W_{h2o}, we compute $\frac{\partial \mathscr{L}}{\partial W_{h2o}} = \frac{\partial \mathscr{L}}{\partial z} \sum_{t'}^{T} \frac{\partial S_{o,t'}}{\partial W_{h2o}}$, where each term can be computed at each timestep t' as

$$\frac{\partial S_{o,t'}}{\partial W_{h2o}} = \frac{\partial S_{o,t'}}{\partial u_{o,t'}} \frac{\partial u_{o,t'}}{\partial W_{h2o}} + \sum_{\xi=0}^{t'-1} \frac{\partial S_{o,t'}}{\partial u_{t'}} \frac{\partial u_{o,t'}}{\partial u_{o,\xi}} \frac{\partial u_{o,\xi}}{\partial W_{h2o}}$$

and

$$\frac{\partial S_{o,t'}}{\partial W_{h2h}} = \sum_{\xi=0}^{t'} \frac{\partial S_{o,t'}}{\partial u_{h,\xi}} \frac{\partial u_{h,\xi}}{\partial W_{h2h}}$$

, and where W_{h2h} refers to weights between neurons in the hidden layers, and W_{h2o} to weights between hidden and output neurons. The discontinuous spiking

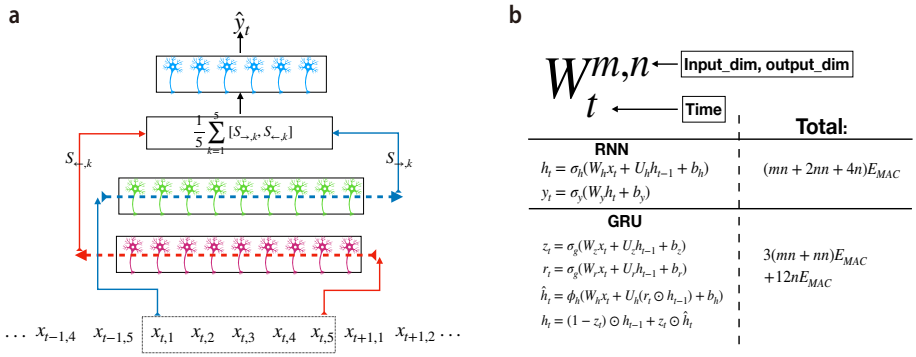


Figure 2.3: **a**, Bi-directional SRNN architecture. **b**, Computational cost computation of different layers for regular RNNs and GRU units. The computational complexity calculation follows [83].

function enters the gradient as the term $\frac{\partial S}{\partial u}$, and here we use the differentiable surrogate gradients [65].

For the Multi-Gaussian surrogate gradient, we found effective parameter values $h = 0.15$ and $s = 6.$ based on a grid search, and we set σ to 0.5. The standard surrogate gradients were defined following [65], with the Linear surrogate gradient as $\hat{f}_s{}'(u_t|\vartheta) = ReLU(1 - \alpha_{linear}|u_t - \vartheta|)$; the SLayer [73] gradient as $\hat{f}_s{}'(u_t|\vartheta) = \exp(-\alpha_{slayer}|u_t - \vartheta|)$, and the Gaussian surrogate gradient as $\hat{f}_s{}'(u_t|\vartheta) = \mathcal{N}(u_t|\vartheta, \sigma_G)$; for all gradients, α is positive. We optimized all surrogate gradients hyperparameters in the experiments using grid searches; in the experiments we used $\alpha_{linear} = 1.0$, $\alpha_{slayer} = 5.0$, and $\sigma_G = 0.5$.

2.2.1 Spiking Recurrent Neural Networks (SRNN)

We focus here on multi-layer networks of recurrently connected spiking neurons, as illustrated in Figure 2.2; variations include networks that receive bi-directional input (bi-SRNNs; Figure 2.3**a**).

Spiking neurons are derived from models that capture the behavior of real biological neurons [84]. While biophysical models like the Hodgkin-Huxley model are accurate, they are also costly to compute [85]. Phenomenological models like the Leaky-integrate-and-fire (**LIF**) neuron model trade-off levels of biological realism for interpretability and reduced computational cost: the LIF

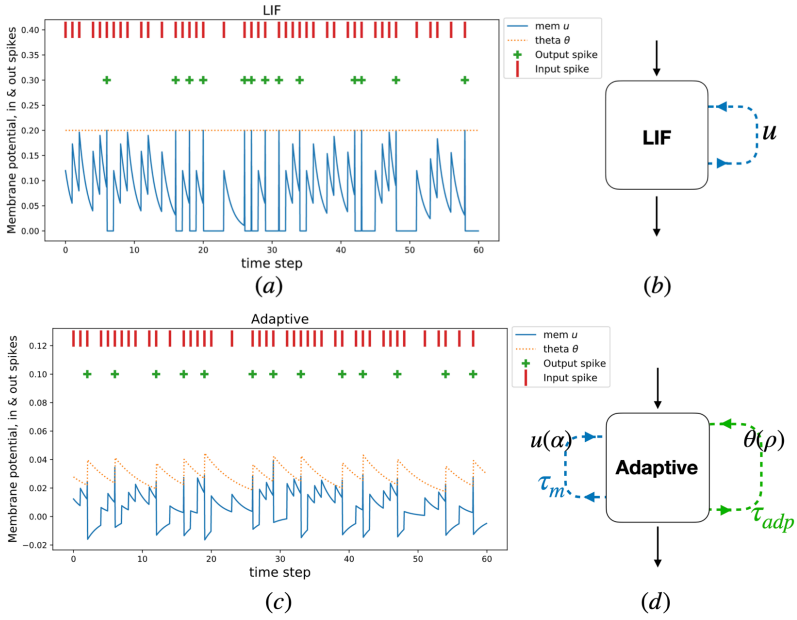


Figure 2.4: The decaying threshold and membrane potential of the LIF and ALIF neurons can be modeled as an internal state induced by self-recurrency.

neuron model integrates input current in a leaky fashion and emits a spike when its membrane potential crosses its threshold from below, after which the membrane potential is reset to the reset membrane potential; the current leak is determined by a decay time-constant τ_m.

As an exceedingly simple spiking neuron model, the LIF neuron lacks much of the complex behavior of real neurons, including responses that exhibit longer history dependency like spike-rate adaptation [85]. Bellec et al. [86] demonstrated how using a spiking neuron model that uses a generic form of adaptation improved performance in their SNNs. In this adaptive LIF (**ALIF**) neuron, the LIF neuron model is augmented with an adaptive threshold that is increased after each emitted spike, and which then decays exponentially with time-constant τ_{adp}. Both LIF and ALIF neurons can be thought of as neural units with self-recurrency, as illustrated in Figure 2.4.

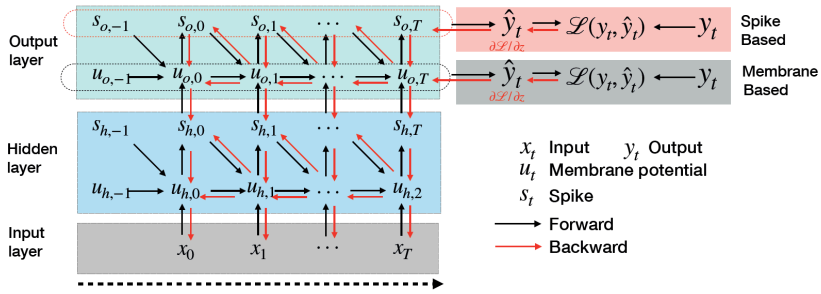


Figure 2.5: Roll-out of the computational graph of a spiking neuron as used for backpropagation-through-time.

BPTT, Surrogate-Gradient and Multi-Gaussian Given a loss-function $\mathscr{L}(t|\theta)$ defined over neural activity at a particular time t, the error-backpropagation-through-time (BPTT) algorithm [54] updates network parameters θ in the direction that minimizes the loss by computing the partial gradient $\partial\mathscr{L}(t)/\partial\theta$ using the chain-rule. Here, the parameters θ include both the synaptic weights and the respective neural time-constants. In recurrently connected networks, past neural activations influence the current loss and by unrolling the network the contribution of these past activations to a current loss is accounted for. The roll-out of network activity through which the gradient is computed is illustrated in Figure 2.5.

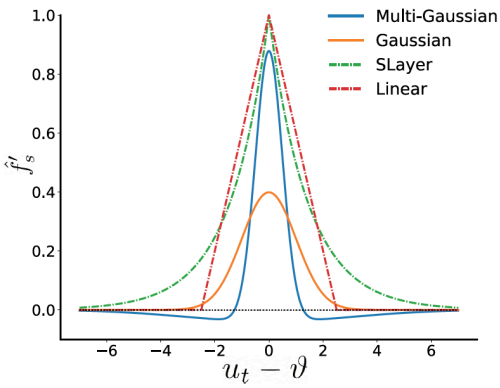


Figure 2.6: Illustration of different surrogate gradient functions $\hat{f}'_s$ as a function of the neuron's membrane potential and threshold , where the Multi-Gaussian is parameterized as in the experiments below ($s = 6, h = 0.15$).

The discontinuous nature of the spiking mechanism in spiking neurons makes it difficult to apply the chain-rule connecting the backpropagating gradient between neural

output and neural input [62]; in practice, replacing the discontinuous gradient with a smooth gradient function, a "**surrogate gradient**" has proven effective [63, 65, 77] and has the added benefit of allowing the mapping of spiking neural networks to recurrent neural networks in optimized Deep Learning frameworks like PyTorch and Tensorflow [65]. Multiple surrogate gradient functions have been proposed and evaluated, including Gaussian, linear [86] and SLayer [73] functions; for these functions however, no significant differences in performance are reported [65].

We here define the Multi-Gaussian (MG) as a novel surrogate gradient $\hat{f}_s'(\cdot)$ comprised of a weighted sum of multiple Gaussians $\mathcal{N}$ where the hyperparameter h and s are chosen such that the Multi-Gaussian contains negative parts:

$$\hat{f}_s'(u_t|\vartheta) = (1+h)\mathcal{N}(u_t|\vartheta,\sigma^2) - h\mathcal{N}(u_t|\sigma,(s\sigma)^2) - h\mathcal{N}(u_t|-\sigma,(s\sigma)^2), \quad (2.8)$$

where u_t is the spiking neuron's membrane potential and ϑ its internal threshold. The Multi-Gaussian surrogate gradient is inspired by the dSilu [80] activation-function which was shown to outperform the standard sigmoidal activation function both for accuracy and learning speed, and which has a derivative similar to the Multi-Gaussian. Effectively, the negative parts of multi-Gaussian gradient regularize activity, as it penalizes both relatively large inputs and small inputs [81]. he gradient function thus aids the SNN to achieve high accuracy with sparse neural activity. The shape of the Multi-Gaussian (MG) and various other surrogate gradient functions is illustrated in Figure 2.6.

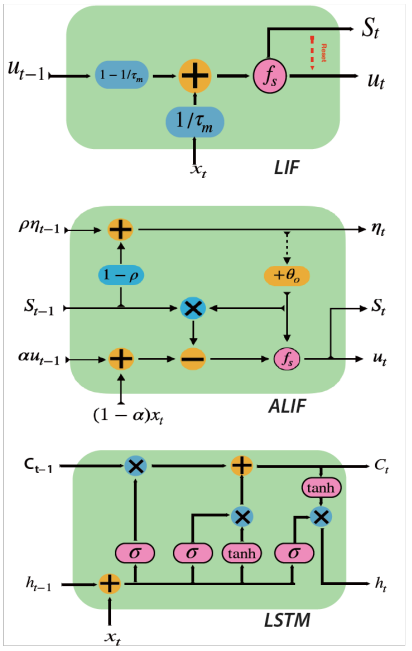


Figure 2.7: LIF, ALIF and LSTM internal operation schematic.

Computational Cost To estimate the efficiency of SNNs and compare them to ANNs, we calculate the number of computations required in terms

of accumulation (AC) and multiply-and-accumulate (MAC) operations [87]. We do this for an SRNN network with LIF or ALIF neurons and compare to a complex recurrent ANN structure like an LSTM [88] in Figure 2.8 – for other ANNs, see Figure 2.3b.

In ANNs, the contribution from one neuron to another requires a MAC for every timestep, multiplying each input activation with the respective weight before adding to the internal sum. In contrast, for a spiking neuron a transmitted

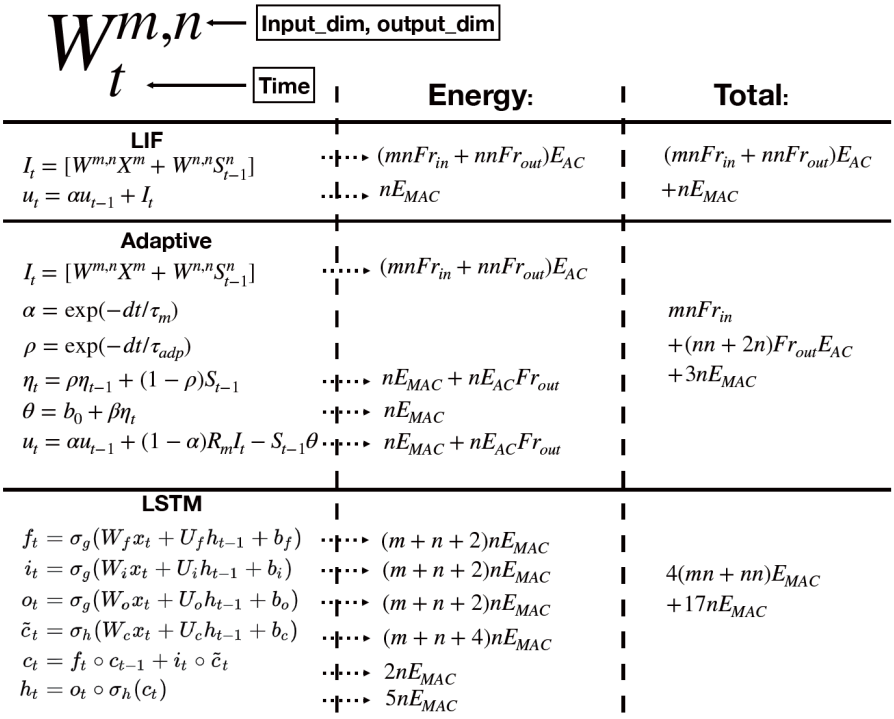


Figure 2.8: Theoretical energy computation of different layers. The computational complexity calculation follows Hunger [83]. Complexity is computed for a single recurrently connected layer where each neuron receives n feedforward inputs with average spike probability Fr_{in} and m recurrent inputs with average spike probability Fr_{out}; E_{AC} and E_{MAC} denote the energy cost for AC and MAC operations respectively.

spike requires only an AC at the target neuron, adding the weight to the potential, and where spike inputs may be quite sparse. In addition, the spiking neuron's internal state requires updating every timestep at the cost of several MACs depending on the spiking neuron model complexity [79]. As calculating MACs is much more energetically expensive compared to ACs (e.g., 31x on 45nm CMOS [18]), the relative efficiency of SNNs is determined by the number of connections times activity sparsity and the spiking neuron model complexity. Additionally, we remark that in hardware, multiplication circuits require substantially more die area compared to addition circuits [89].

2.3 Experiments

Recurrent neural networks (RNNs) provide state-of-art performance in various sequential tasks that require memory [90] typically in small and compact networks and can operate in an online fashion. We distinguish two kinds of sequential tasks: streaming tasks, where many inputs map to many specified outputs (many-to-many), and classification tasks where an input sequence maps to a single output value (many-to-one). Sequential classification tasks can additionally be computed in an online fashion, where classification is determined for each timestep.

2.3.1 Datasets

We selected benchmark tasks with an inherent temporal dimension that can also be computed with recurrent neural networks of modest size to fit the dynamics and constraints of spiking neural networks. For these tasks, we trained several different SRNN network architectures with various gradients, hyperparameters, and spiking neuron models and compared them to classical and state-of-the-art RNN architectures. Hyper-parameters were selected using three-fold cross-validation on the training data.

The electrocardiogram (**ECG**) [91] signal is composed of six different characteristic waveforms – P, PQ, QR, RS, ST, and TP – whose shape and duration inform clinicians on the functioning of the cardiovascular system. The task requires the continuous recognition of all six waveforms, where we use signals from the QTDB dataset [91]. The ECG-wave labeling is an online and streaming task using only past information. The sequential- and permuted-sequential **S/PS-MNIST** datasets are standard sequence classification tasks of

length 784 derived from the classical MNIST digit recognition task by presenting pixels one at a time. The permuted version also first permutes each digit-class removing spatial information. The Spiking Heidelberg Dataset (**SHD**) and Spiking Speech Command (**SSC**) Dataset [92] are SNN specific sequence classification benchmarks comprised of audio converted into spike trains based on a detailed ear model.

The **SoLi** dataset [93] gesture recognition task is comprised of a set of gestures where each gesture is measured as a sequence of radar returns collected from the SoLid-state millimeter-wave radar sensor (SoLi). We treat the SoLi task as both an online streaming and classification task by processing frames sequentially - we thus obtain two measures for the SoLi task, per-frame accuracy and whole sequence accuracy for streaming and classification respectively.

Both the Google Speech Commands (**GSC**) dataset [94] and the **TIMIT** dataset [95] are classical speech recognition benchmarks where for TIMIT, we compute the Frame Error Rate (FER) and where, similar to [77], we apply a bidirectional architecture such that also future information is used to classify each frame (illustrated in Figure 2.3**a**). Samples from the ECG, SHD and SoLi datasets are shown in Figures 2.9**a-c**.

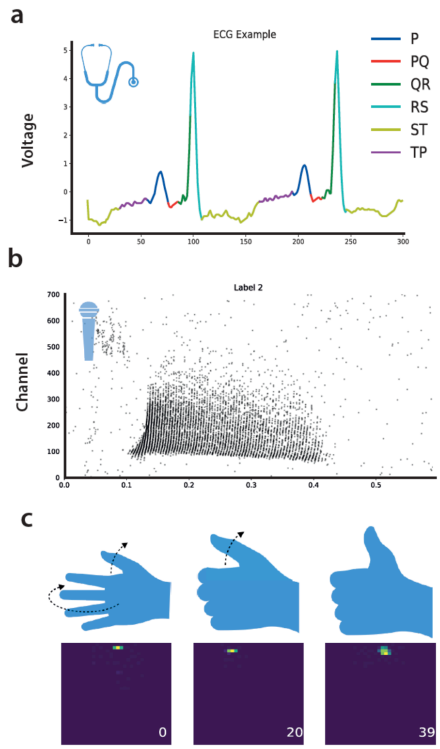


Figure 2.9: Examples of **a**, a single ECG signal channel labeled for each timestep; **b**, the input spike-trains for the spoken number "seven" in the SHD dataset and **c**, example of gesture data – the temporal evolution of the gesture (upper row) and the corresponding RDI (bottom row);

Network initialization. Compared to ANNs, SRNNs require initializing both weight and also the spiking neu-

rons' hyperparameters (i.e, neuron type, time constants, thresholds, starting potential). We randomly initialize the time constants following a tight normal distribution (μ, σ) with per-layer specific parameters given in Appendix A Table A1. For all neurons, the starting value of the membrane potential is initialized with a random value distributed uniformly in the range$[0, \vartheta]$. The bias weights of the network are initialized as zero and all feedforward weights are initialized using Xavier-uniform initialization; weights for recurrent connections are initialized as orthogonal matrices. We compared networks with constant, uniform, and normal initializers for the time-constants and found that the normal initializer achieved the best performance (Figure 2.10).

2.3.2 Results

For the various tasks, the loss-function, sequence length, maximum number of epochs, learning rate and decay schedule, and minibatch-size are specified in Appendix A Table A1. Validation showed that the SRNNs were not prone to overfitting and test accuracy was measured at the last epoch. Unless specified otherwise, the network architecture consists of inputs densely connected to one or more fully recurrently connected layers of spiking neurons connected to a layer of output neurons, as illustrated in Figure 2.2. For the **ECG** task, the QTDB dataset [91] consists of two channels of ECG signals. We apply a variant of level-

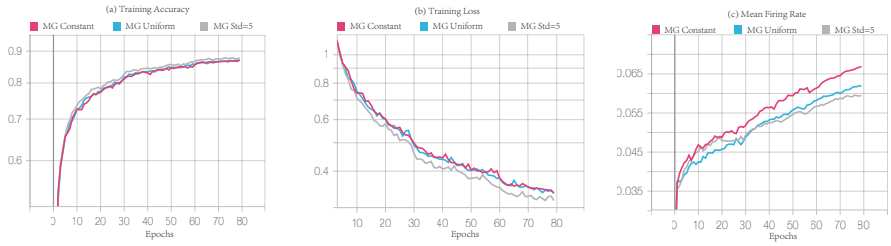


Figure 2.10: **Effects of different time constant initialization schemes on network training and performance on the SoLi dataset. (a)**, Training accuracy **(b)**, Training Loss **(c)**, Mean Firing rate of the network. The $\text{MG}_{constant}$ is the network where τ is initialized with a single value; for $\text{MG}_{uniform}$ the network is initialized with uniformly distributed time-constants near the single value of $\text{MG}_{constant}$; for MG_{std5}, a normal distribution with std 5.0 is used near the same single value.

crossing encoding [96] threshold on the derivative of the normalized ECG signal to convert the original continuous values x into a spike train: each channel was transformed into two separate spike trains representing value increasing events and value decreasing events respectively. The level crossing encoding we used is defined as

$$S_+ = \begin{cases} 1, & \text{if } x_t - x_{t-1} \geq L_+ \\ 0, & \text{otherwise} \end{cases}, \qquad S_- = \begin{cases} 1, & \text{if } x_{t-1} - x_t \geq L_- \\ 0, & \text{otherwise} \end{cases}$$

where x is the signal being encoded, S_+, S_- denote spikes for the positive and negative spike-train respectively and we used $L_+ = 0.3$ and $L_- = 0.3$.

For the **Spiking Heidelberg Dataset (SHD)**, the audio records were aligned to 1s by cutting or completing with zeros. As in Cramer et al [92], two speakers were held out for the test dataset, and 5% of samples from other speakers were also added into the test dataset. The training dataset thus comprised of 8156 samples and test dataset contains 2264 samples. For the **Spiking Speech Command Dataset**, the speech commands were also uniformly aligned to 1s with a 250hz sampling frequency, and the dataset was randomly split into training, validation and test dataset with a ratio of 72%-8%-20% respectively. For the **SoLi dataset**, the sequence of 40 Range-Doppler images (RDI) was fed into the model frame-by-frame as input and split into training and testset as in Wang et al [93]. The original RDIs have 4 channels, but we found empirically that using one channel was sufficient. For the SoLi task, the first layer of the SRNN, we use a feedforward spiking dense layer, followed by a recurrent layer. As in Wang et al, [93], separate networks were trained for per-frame accuracy (Acc_s) and per-sequence accuracy (Acc_c), for the streaming and classification version of the task respectively. In the **S-MNIST** tasks, the network read the image pixel by pixel; for the **PS-MNIST** task, pixels are read into the network using a sliding window of size 4 with stride 1. For both tasks, the pixel value is fed into the network directly as injected current into the neurons of the first hidden layer as a fully connected layer with its own weights. We use the **Google Speech Command** v1 [94]. For preprocessing, Log Mel filters and their first and second-order derivatives are extracted from raw audio signals using Librosa [97]. For the FFTs, a window of 30ms and a hop of 10ms is used. The timestep of the simulation is 10 ms. We calculate the logarithm of 40 Mel filters coefficients using the Mel scale between 20 Hz and 4kHz. Additionally, spectrograms are normalized to ensure that the signal in each frequency has a variance of 1 across time; we then selected the first three derivative orders as three distinct input channels. The input to the SRNN is thus a sequence of 101 frames, where each

Table 2.1: **Comparison of SRNN performance to respective RNN and SNN state-of-the art accuracy (Acc.).**

Task	Network	Method	Acc.	Task	Network	Method	Acc.
ECG	*RNN-SoTa*	Bi-LSTM	80.8%	SSC	*RNN-SoTa*	LSTM [92]	73.1%
	SRNN	**Ours**	**85.9%**		*CNN-SoTa*	CNN [92]	77.7%
SMNIST	*RNN-SoTa*	IndRNN [98]	99.5%		SNN-base	LIF [92]	50.1%
	RNN	LSTM [99]	98.2%		SRNN-SoTA	SNN [100]	60.1%
	SRNN-SoTa	LSNN [86]	96.4%		**SRNN**	**Ours**	**74.2%**
	SRNN	**Ours**	**98.7%**	SoLi	*CNN-SoTa*	CNN [93]	77.7%
PSMNIST	*RNN-SoTa*	IndRNN [98]	97.2%		RNN-SoTa	CNN+LSTM [93]	87.2%
	RNN	LSTM [99]	88%		**SRNN**	**Ours**	**91.9%**
	SRNN	**Ours**	**94.3%**	GSC	*RNN-SoTa*	Att RNN [101]	95.6%
SHD	*RNN-SoTa*	Bi-LSTM	87.2%		*CNN-SoTa*	SCNN [102]	94.5%
	CNN-SoTa	CNN [92]	92.4%		SNN-SoTa	LSNN [77]	91.2 %
	SNN-base	LIF [92]	71.4%		**SRNN**	**Ours**	**92.1%**
	SNN	SNN [103]	82.2%	TIMIT	*RNN-SoTa*	Bi-LSTM [104]	68.9%
	SNN-SoTa	SNN [100]	82.7%		SNN-SoTa	LSNN [86]	65.4%
	SRNN	**Ours**	**90.4%**		**Bi-SRNN**	**Ours**	**66.1%**

frame comprises of a 40-by-3 matrix.

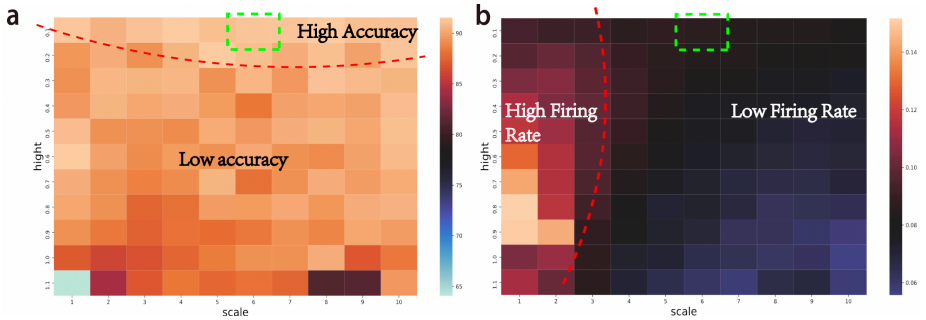


Figure 2.11: **a,b**, Grid search for h and s parameters for the Multi-Gaussian surrogate gradient on the SoLi dataset. The dotted line demarcates the top-left area of solutions with high accuracy (>0.91) in **a**, and high firing sparsity (>0.09) in **b**. The green box denotes the selected h and s values.

The **TIMIT** database contains 3696 and 192 samples in training and test data respectively. We preprocessed the original audio data as in Bellec et al [77] using MFCC encoding; 10% of the training dataset was randomly selected as validation dataset, and the network was trained on the remainder. Similar to bi-directional LSTMs, we use a bi-directional Adaptive SRNN for

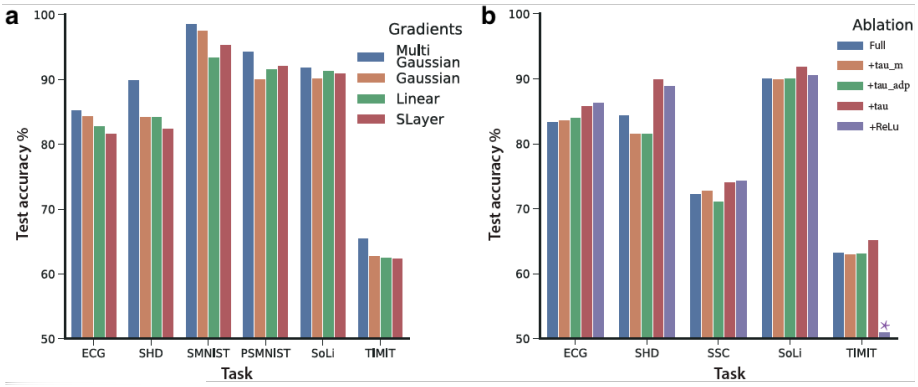


Figure 2.13: **a**, Effects of various surrogate gradients on performance. **b**, Effects of training the time constant hyperparameters τ_m and τ_{adp}; the legend denotes which hyperparameters are trained; ReLU denotes the non-spiking analog SRNN.

this task, illustrated in Figure 2.3**a**: we use two SRNN layers in the network, reading the sequence from the forward and backward direction respectively. The mean of these layer's output is then fed into the last layer, an integrator, to generate the class prediction.

As shown in Table 2.1, we find that these SRNNs achieve novel state-of-the-art performance for Spiking Neural Networks on all-but-one tasks, exceed conventional RNNs like LSTM models, and approach or exceed the state-of-the-art of modern RNNs. For GSC, we exceed current SNN state-of-the-art for recurrent and online processing and approach the non-streaming result of [102]. Moreover, we see that SRNNs substantially close the accuracy gap (SHD, SSC, GSC) compared to non-recurrent architec-

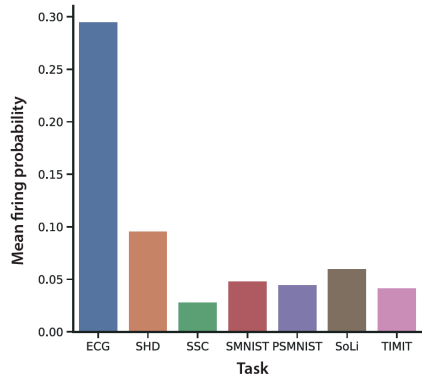


Figure 2.12: The per timestep spike probability of the SRNNs on various tasks.

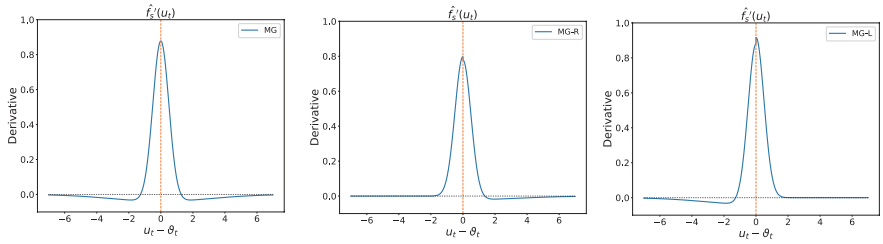


Figure 2.15: Variants of Multi-Gaussian gradient. As illustrated, we remove either the left(MG-R) or right(MG-L) negative part of the Multi-Gaussian gradient for comparison, leaving on the ablated part the positive Gaussian gradient.

tures like convolutional neural networks (CNNs) and Attention-based networks – the latter networks however are typically comprised of many more neurons or parameters and cannot be computed in an online or streaming fashion.

We plot the accuracy for the various tasks using different surrogate gradients in Figure 2.13: while we see that there is little difference between previously developed gradients like Gaussian, Linear, and SLayer, we find that the Multi-Gaussian function consistently outperforms these gradients. To better understand why the Multi-Gaussian is beneficial, we removed either the left or right negative part of the gradient for comparison.

We found consistently that both performance and sparseness improved for both parts (Table 2.2, Figures 2.15. and 2.16), demonstrating that the negative parts of the Multi-Gaussian act as effective regularizers.

	SOP	SOP/Step
ECG	35011.2	26.9
SHD	24690	98.76
SSC	19450	77.8
SMNIST	70810.8	90.32
PSMNIST	59772.1	76.24
SoLi	3645.4	91.13
TIMIT	21504	43.0
GSC	12600	126

Figure 2.14: Total average Spike Operations (SOP) per sample and SOPs per sample per step (timestep/frame).

We also find that independently of the surrogate gradient used, training the time-constants in the Adaptive LIF neurons consistently improves performance, as shown in the ablation study in Figure 2.13a: not training either τ_m or τ_{adp}, or neither, reduces performance. Much of the power of the SRNNs seem to derive from their multi-layer recurrent and self-recurrent architecture. When we make the spiking neurons non-spiking by

Task	Setting	Accuracy	Sparsity	Task	Setting	Accuracy	Sparsity
ECG	MG	81	0.1652	**SoLi**	MG	91.89	[0.042,0.0967]
	MG-R	76.58	0.1494		MG-R	88.14	[0.0515,0.105]
	MG-L	76.62	0.1718		MG-L	88.4	[0.0521,0.0977]
	MG-Dense	66.28	0.1623		MG-Dense	86.41	[0.0547,0.1459]
ECG-LIF	MG	69.04	0.3711	**GSC-V1**	MG	92.13	[0.0975,0.1156]
	MG-R	57.13	0.5424		MG-R	87.32	[0.108,0.137]
	MG-L	58.83	0.3402		MG-L	87.14	[0.0968,0.1357]
	MG-Dense	49.14	0.1288		MG-Dense	83.6	[0.0964,0.0857]
SMNIST	MG	98.34	[0.0302,0.123,0.135]	**SHD**	MG	90.31	0.0575
	MG-R	96.28	[0.0293,0.1531,0.1456]		MG-R	84.84	0.0618
	MG-L	96.43	[0.04,0.1451,0.1851]		MG-L	85.02	0.0605
	MG-Dense	93.85	[0.0586,0.2224,0.2333]		MG-Dense	81.757	0.058

Table 2.2: **Ablation study of different forms of Multi-Gaussian gradient, including results for dense (non-recurrent SNNs**. We apply two variants of Multi-Gaussian (MG) gradient – MG-L and MG-R to explore asymmetric shapes of the Multi-Gaussian. The MG-L and MG-R variations only have negative parts on the left or right part of the gradient respectively. To study the effect of recurrent connections in the network, we removed these connections to create the MG-Dense SRNNs. The sparsity value in the table is calculated by the average number of spikes at each time step. For MG-Dense SRNNs, we find removing the recurrent connections consistently lowers accuracy. All reported values are best of three runs. In addition, both ECG and ECG-LIF were run for 50 training epochs for comparison.

eliminating the spiking mechanism and communicating the RELU value of the membrane potential, we find that for almost all tasks we achieve performance that slightly exceeds that of the spiking SRNNs.

The trained SRNNs communicate sparingly: most networks exhibit sparseness less than 0.1, and only the ECG task requires more spikes as it was tuned to use the smallest SRNN network (46 neurons). Sparseness of neural activity, expressed as average firing probability per timestep per neuron, is plotted in Figure 2.12.

In general, we find that increasing network sizes improves accuracy while decreasing average sparsity (Figure 2.18) – though the total number of spikes used in the network increases. The total average number of spikes required per sample (SOPs) and per sample per step (SOP/step) for the highest performing SRNNs are given in Figure 2.14. We also evaluated to what degree the internal recurrency of spiking neurons contributes compared to the intra-layer recurrent connectivity: we find that the addition of intra-layer recurrent connections consistently improves accuracy (Table 2.2 in Supplementary Information).

Plotting the performance of networks

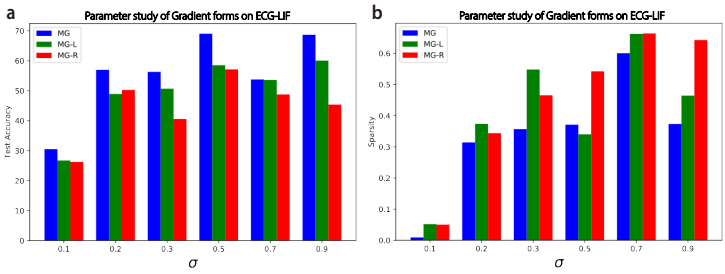


Figure 2.16: Study of different forms of gradients on ECG-LIF. (a,b) shows the result of the using various Multi-Gaussian negative gradient ablations on the ECG-LIF task where the σ of the central (positive) Gaussian as defined in Eq (2.8) is varied. The effect of varying σ is shown for test accuracy (a) and sparsity (b). We find that also then, the standard Multi-Gaussian outperforms variations in terms of accuracy and sparsity.

using either ALIF or LIF neurons, we find that using ALIF neurons consistently improves both performance and activity sparseness in the networks (Figure 2.18**a**). Similarly, splitting a single large recurrent layer into two layers of recurrently connected layers in the SRNN architecture improves both performance and sparsity in the SHD task (Figure 2.18**b**); we observed similar improvements in the other tasks.

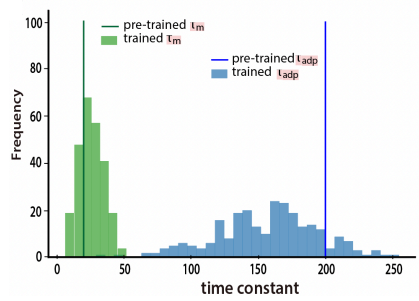


Figure 2.17: Evolution of spiking neuron time constants evolving before and after training

We carried out a grid search on the SoLi and SHD datasets for the h and s hyperparameters to determine the optimal parameter values for the Multi-Gaussian surrogate gradient using cross-validation. We find that there is a range of values where we can obtain both competitive accuracy and high sparsity (areas top-left of the orange dotted line in Figures 2.11**a-b**) – we used a similar hyper-parameter search for the other tasks using selected values only from the high-accuracy/low activity area identified here. The training procedure also substantially 'learns' the time-constants for the respective tasks: as shown in Figure 2.17 for the SHD task,

starting from a tight distribution of time-constants, the spiking neurons in the trained network converge to using a wide variety of time-constants - the same effect is observed in the other tasks (not shown).

The streaming and online nature of several of the tasks allows the network to make any-time decisions. Figure 2.19**a** shows the classification for the various ECG waveforms for every timestep. When a new wave is presented, there is a brief delay before this class is correctly identified. In Figures 2.19**b-f**, the average online classification performance is shown for the S-MNIST, PS-MNIST, SHD, SSC, and SoLi datasets. We see that the S-MNIST and PS-MNIST digits can be recognized reliably quickly, while the SSC sounds require distinctly more time. The SHD sound recognition is much more erratic, and inspection of the data shows that this is caused by the various classes being placed at different times in the sound clip. Figure 2.19**f** plots the accuracy as a function of the number of frames shown for the SoLi task. Most gestures can be recognized reliably already after having presented only 25 out of the 42 frames - comparing favorably with [93]: the SRNN allows decisions to be made earlier and with better accuracy.

Given the relative AC and MAC energy cost from [18, 79, 105] and the computational complexity calculations from Figure 2.8, we plot in Table 2.3 the relative energy efficiency of the various networks. We see that for the more complex tasks, SRNNs are theoretically at least 59x more energy efficient compared to

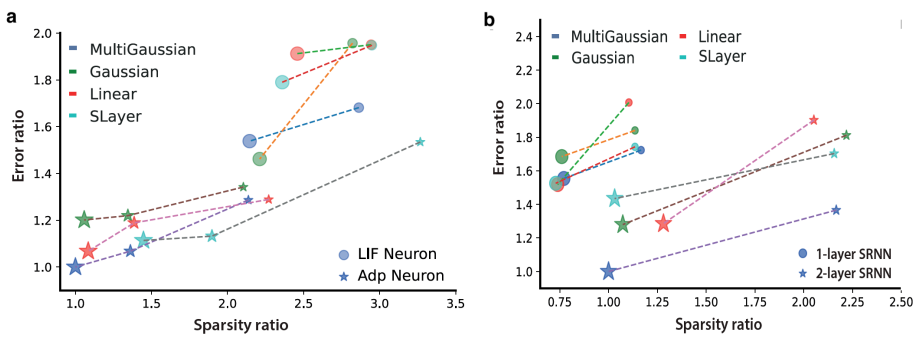


Figure 2.18: **a**,Effect of neuron types in terms of test accuracy and sparsity with various gradients (shown for SoLi dataset); the size of the nodes indicates the network size and the color of nodes represents the gradient type; **b**, Effect of the number of hidden recurrent layers on test accuracy and sparsity with various gradients (shown for SHD dataset).

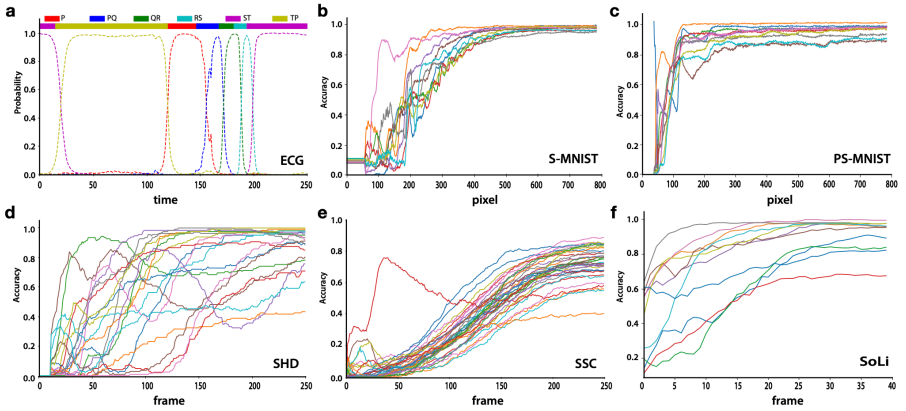


Figure 2.19: **a**, example of ECG streaming classification: the prediction probability of each output label is calculated from the normalized output neurons' membrane potential (dashed lines, bottom); top: the color-coded true labels. **b-f** Temporal evolution of classification accuracy for the S-MNIST recognition task **c**, the PS-MNIST task **d**, the SHD recognition task **e**,, the SSC dataset **f**,, and for the SoLi dataset **i**

RNNs at equivalent performance levels, where for most tasks the non-spiking (ReLU) SRNN compares most favourably.

More classical RNN structures like LSTMs require many more parameters and operations, often being 1000x less efficient – we also calculate similar estimates for other RNN structures in Appendix Table A2.

2.4 Discussion

We showed how multi-layered recurrent network structures are able to achieve new state-of-the-art performance for SNNs on sequential and temporal tasks. This was accomplished by using adaptive spiking neurons with learned temporal dynamics trained with backpropagation-through-time using a novel surrogate gradient, the Multi-Gaussian, where the Multi-Gaussian gradient proved to consistently outperform the other surrogate gradients. These results approach or equal the accuracy of conventional RNNs, where the non-spiking ReLU-SRNNs consistently slightly outperformed the spiking version, demonstrating the effectiveness of the SRNN network architecture. When expressed in terms of compu-

Table 2.3: **Comparison of SRNN energy consumption to respective RNN and SNN state-of-the-art accuracy.** Relative energy cost is calculated using the number of MACs and ACs required during inference with $Energy_{nn} = \sum_{t \in T} 3.1MAC + .1AC$ [18]; fr denotes the average spiking probability in the SRNNs per timestep. The Bidirectional-LSTM network 290* contains 290 LSTM units. The accuracy in SoLi dataset is per frame accuracy. For GSC, the ReLu SRNN did not converge. ** For TIMIT, the complexity of comparably accurate networks was not available.

Task	Method	Network	Acc	Energy/Step		Ratio
				MAC	AC*fr	
ECG	Bi-LSTM	290	80.8	181.8k		1.7Kx
	ReLU	4+36+6	86.4	1.9k		18x
	Ours (LIF)	4+36+6	49.7	42	.5k	0.5x
	Ours	4+36+6	85.9	90	.5k	**1x**
S-MNIST	ReLU	64+256+256	99.0	157.3k		59x
	Ours	64+256+256	98.7	2k	20k	**1x**
PS-MNIST	ReLU	64+256+256	94.5	157.3		63x
	Ours	64+256+256	94.3	2k	15.3k	**1x**
SSC	ReLU	400+400	74.4	766.6k		236x
	Ours	400+400	74.2	2.4k	26.1k	**1x**
SHD	Bi-LSTM	128+128+100	87.2	1.1M		1.7Kx
	ReLU	128+128	88.9	142.6k		125x
	Ours (ALIF)	128+128	90.4	788	10.7k	**1x**
SoLi	LSTM	512+512	77.7	2.7M		604x
	ReLU	512+512	79.6	1.1M		246x
	Ours	512+512	79.8	3.1k	42.4k	**1x**
GSC	ReLU	300+300		222.6k		167x
	Ours	300+300	92.2	1k	10.1k	**1x**
TIMIT**	Ours	256+61	66.1	1.6k	56.7k	**1x**

tational operations, they demonstrate a decisive theoretical energy advantage of one to three orders of magnitude over conventional RNNs. This advantage furthermore increases for more complex tasks that required larger networks to solve accurately.

The Multi-Gaussian gradient was inspired by a sigmoid-style saturating activation function developed for standard artificial neurons, the dSilu, which has a similarly shaped gradient. As with the dSilu, we also find that the negative

parts of the gradient help improve accuracy, and in the SRNN also sparseness. The latter suggests that the negative parts of the gradient act as effective regularizers.

Neither the SRNNs nor the presented RNNs were optimized beyond accuracy and (for the SRNNs) sparsity: no optimizations like pruning and quantization were applied. When we compare the SRNN for the GSC task against the Attention-based CNN-network TinySpeech [87], representing the recent state-of-the-art in efficiency-optimized speech recognition, we find that at an equivalent performance level, the SRNN still requires 19.6x fewer MACs, and where, unlike TinySpeech, the SRNN operates in an online and streaming fashion (data in Appendix Table A2).

We focused on temporal or sequential problems with relatively limited input dimensionality. With RNNs, such problems can be solved with relatively small neural networks and hold direct promise for implementation in ultra-low power EdgeAI solutions. This also was the reason for emphasizing streaming or online solutions where no or fixed preprocessing and buffering is required: problems where a temporal stream first has to be segmented and where these segments are then classified greatly increase the complexity of such solutions. As we demonstrated, most classification decisions could be made early with near-optimal accuracy.

The datasets discussed here were all selected for being amenable to streaming and online processing by SRNNs with very limited pre-processing, such as calculating Log Mel filters. In preliminary work, the use of conventional convolutional network layers to extract useful features proved helpful for simple subsequent layers of spiking neurons [67]. We similarly find[1] that deep pre-processing improves accuracy substantially on tasks like GSC and also the DVS128 dataset [106] where SRNNs obtained scores exceeding those reported by [67, 107]. This suggests that for even larger problems than those studies here, deep pre-processing holds much promise when balanced against the impact on complexity and energy requirement and also on the ability to process event-based streaming data.

Using surrogate-gradients, the BPTT-gradient in the SRNNs can be computed using standard deep learning frameworks, where we used PyTorch [108]. With this approach, complicated architectures and spiking neuron models can be trained with state-of-the-art optimizers, regularizers, and visualization tools.

[1] With a hybrid CNN-SRNN we obtained an accuracy of 97.91% on the DVS128 dataset and 96.5% on the GSC dataset, with CNN-SRNN code available at `https://github.com/byin-cwi/Efficient-spiking-networks/tree/main/DVS128`

At the same time, this approach is costly in terms of memory use and training time, as the computational graph is fully unrolled over all timesteps, precluding online and on-chip learning. Additionally, the abundant spatial and temporal sparsity is not exploited in the frameworks. This also limits the size of the networks to which this approach can be applied: for significantly larger networks, either dedicated hardware and/or sparsity optimized frameworks are needed [109]. Approximations to BPTT like eProp [77] or alternative recurrent learning methods like RTRL [110] may also help alleviate this limitation.

We remark that the energy advantage of SRNNs we computed is theoretical: while the computational cost in terms of MACs is well-accepted [87, 105], this measure ignores real-world realities like the presence or absence of sufficient local memory, the cost of accessing memory, and the potential cost of routing spikes from one neuron to another. In many EdgeAI applications, the energy-cost of conventional sensors may also dominate the energy equation. At the same time, the numbers we present are unoptimized in the sense that other than optimizing the surrogate gradient for both sparsity and accuracy, we did not prune the networks or applied other standard optimization and quantization techniques. Substantial improvements here should be fairly straightforward. Training parameters of spiking neuron models in the SRNNs can be extended further to approaches that include parameterized short-term plasticity [111] and more complicated spiking neuron models.

The effectiveness of adjusting time-constant parameters to the task may also have implications for neuroscience: though effective time-constants of real spiking neurons are variable and dynamic [84], the benefit of training these parameters in SRNNs suggests these neural properties may be subject to learning processes in biology.

Data Availability

All the datasets in the study are open source and publicly available; links to the datasets are listed in our repo.

Code Availability

The code used in the study is publicly available from the GitHub repository `https://github.com/byin-cwi/Efficient-spiking-networks`.

Acknowledgements

Bojian Yin is funded by the NWO-TTW Programme Efficient Deep Learning (EDL) P16-25. The authors gratefully acknowledge the support from the organizers of the Capo Caccia Neuromorphic Cognition 2019 workshop and Neurotech CSA, as well as Jibin Wu and Saray Soldado Magraner for helpful discussions.

Chapter 3

Training SRNN Through Truncated BPTT

Significance: **The previous chapter, we found that SNNs face many problems in their training. As described in *Challenge 2*, this results in a longer time for training and might leads to an unstable training process. As described in *Challenge 3*, training SNNs takes up absolutely more memory than RNNs. To alleviate these problems, we used Truncated-BPTT (TBPTT) in LIDAR image recognition to obtain lower energy consumption for event-based object detectors. To this end, we also introduce a new and open dataset for event-based LIDAR object recognition. This chapter is based on publication "Real-time classification of LIDAR data using discrete-time Recurrent Spiking Neural Network" [112].**

Abstract: *With the advancement of Edge AI and autonomous systems, AI applications are increasingly subject to energy, latency and environmental constraints. Biological neural systems naturally adhere to these constraints and, as such are a source of inspiration. Spiking Neural Networks (SNNs) are a more detailed model of biological neural processing. Recent work shows that they perform well in object recognition and detection in general, and in Autonomous Driving tasks based on ranged LIDAR data in particular. However, these LIDAR-SNN approaches do not optimize for latency, as they require the entire frame to be scanned before processing. They also require large SNNs, limiting the energy efficiency achieved. To reach both low-latency and high energy efficiency in LIDAR object recognition, we develop*

a compact recurrent SNN. First, we propose and examine an open LIDAR labeled dataset by processing the point clouds from the KITTI Vision Benchmark. We then train our recurrent SNNs on this dataset and propose specific optimizations, including input encoding, sparse connectivity and truncation of error-backpropagation. With these optimizations, we show that compact recurrent SNNs can exceed the performance of classical RNNs like LSTMs and approach the performance of large non-spiking CNNs. Additionally, they significantly reduce latency by allowing early and online object classification before the end of the sequence.

3.1 Introduction

With the increasing attention for untethered AI applications – *Edge AI* – there are multiple factors to consider when developing such solutions, from the energy usage of autonomous systems, to the latency of time-sensitive decisions, to the interference of environmental conditions on the sensors. For the technology to scale well, these requirements have to be integrated with the design from the ground up, and may require the use of specialized tools such as event-based sensors. LIDAR, in particular, can provide exact distance measurements and is less affected by lighting conditions, making it a key sensor for the Autonomous Driving industry. However, equipping Autonomous Systems with such instruments needs efficient algorithms that can process and make decisions on LIDAR data with low latency and high energy efficiency.

Latency, in particular, can benefit: considering the sensor used for the KITTI dataset as an example, it can rotate at up to 20Hz, with a significant loss in resolution (down to 1042 points per revolution). To get its full resolution (4167 points per revolution), the sensor needs to spin at 5Hz. Classifying objects online, throughout the scanning process, would potentially significantly increase the reaction speed of a vehicle while allowing the sensors to move at a slower pace and provide sharper input signals.

Spiking Neural Networks (SNN) are a variant of Artificial Neural Networks based on more detailed models of biological neurons. Neurons in SNNs communicate through binary signals – spikes – and can encode information in the relative timings of these spikes. The asynchronous, event-based and sparse communication of SNNs can lead to sizably reduced energy requirements when running these networks on specialized neuromorphic hardware [113].

To achieve both low latency and high energy efficiency, we turn to a recurrent variant of Spiking Neural Networks that is highly suitable for processing time series and that can directly process raw LIDAR data as input. We show

that a discrete-time Recurrent Spiking Neural Network (RSNN) can efficiently classify LIDAR data, in real-time, throughout the scanning process. This setup would assist an Autonomous System in confidently perceiving the environment before a full revolution of the sensor is finished. Moreover, predicting object types before they are fully scanned could allow a system to make low latency decisions.

We develop a reproducible and challenging benchmark for online LIDAR processing from the KITTI Vision Benchmark, by extracting point clouds representing objects and creating a LIDAR object recognition dataset. On this labeled dataset we train a discrete-time RSNN and study performance, latency and energy efficiency.

To achieve satisfactory classification performance, we study the effect of different input encoding methods on the performance of the network. We further show effective solutions can be optimized for better accuracy, shorter training times and better energy efficiency. In particular, we show that large sparsely connected SNNs achieve better accuracy than smaller networks with the same number of non-zero parameters. We present the benefits of using Truncated Backpropagation-Through-Time [57], which improves the training time of the networks, and can have a positive effect on the accuracy.

3.1.1 Background

SNNs are comprised of spiking neurons that capture varying degrees of complex processing in biological neurons. The Leaky-Integrate-and-Fire (LIF) spiking neuron [84, Chapter 4] is a simple phenomenological spiking neuron model that is used in many SNNs. Its behaviour is described by a single variable equation (3.1) and associated spiking condition:

$$\begin{aligned} \tau_m \frac{du}{dt} &= -u(t) + RI(t), \\ S(t) &= \hat{f}_s(u(t), v) \\ u(t) &= u(t)(1 - S(t)) + u_{reset} S(t), \end{aligned} \tag{3.1}$$

where u represents the membrane potential of the neuron, τ_m is the neuron's time constant, such that $\tau_m = RC$, and v is the threshold the membrane potential has to reach for an output spike to be created; $S(t)$ is the binary spike indicator, u_{reset} denotes the potential to which the emission of a spike resets the membrane potential, and $\hat{f}()$ is the spike-function. Compared to other spiking neuron models, the LIF preserves only a few features of biological spiking

neurons [85], but has low implementation cost.

Training SNNs is particularly challenging due to the discontinuous nature of the spiking mechanism, which makes such networks not amenable to standard gradient descent and error-backpropagation as generally used for training Artificial Neural Networks [16]. Various solutions to learning in SNNs have been proposed, including switching to another learning style [114], using a different method of encoding information within the network [115], or changing the way gradients are computed [65, 116]. Most alternatives for computing the gradient tend to be intimately tied to how information is represented by spikes. For example, one can compute the exact time of each spike and then use these continuous values for gradient computation, or consider discrete time steps in the network and approximate the gradient for any given step. The former method has been successfully used in multiple projects [16], [116], [117], including related work on LIDAR data [118]. However, by using such temporal encoding, these networks are generally constrained to one spike per cell for every input, which is not desirable in a setting where information is continuously processed.

The gradient approximation method, also used to train our networks, inserts a function for the non-existing gradient through the spike-function, as a "Surrogate Gradient" [65] [119]. Many choices for this function have been examined in the literature, such as a piece-wise linear function, the derivative of a sigmoid or the exponential function [65]. The Surrogate Gradient method provides flexibility in implementation, in particular in relation to existing ANN frameworks like PyTorch and Tensorflow, and enables convenient handling of both spatial and temporal credit assignment.

3.2 Related work

LIDAR scans the distance to objects in the environment by sweeping the surrounding area with a stream of laser pulses and measuring the return time to work out the distance for each pulse and angle. A full scan is done progressively for different azimuths and elevations.

Previous solutions for solving LIDAR based object-recognition tasks have required the data to be fully scanned and then processed into a point cloud that can then be the input of a classification system. Early work by Himmelsbach et al. [120] split the frame into a grid, computed the connected components of the cells and then used the results for object segmentation and feature computation, which became the input of an SVM classifier. In SqueezeSeg [121], the whole point cloud is projected to a 2D image and then fed into a CNN, which

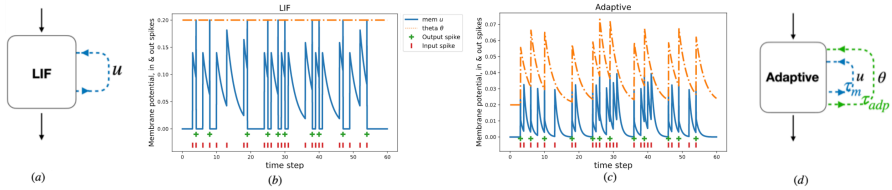


Figure 3.1: LIF and Adaptive spiking neuron behaviour.(a,d) The decay of the membrane potential and adaptation can be modeled as self-recurrent connections. (b,c) Impinging input spikes (red) increase (or decrease) the membrane potential (blue) which then decays back to the resting potential (0). When the potential reaches the fixed threshold θ_0 (yellow dotted line), an output spike is emitted (green cross) and the potential is reset to the resting potential. In the Adaptive spiking neuron (c), the threshold is increased for every emitted spike, and then decays back to the resting threshold b_0.

outputs point-wise labels to detect three types of objects: cars, pedestrians and cyclists. Prokhorov [122] processes the point cloud as a time series of points, which represents the input to a small Recurrent Neural Network that classifies objects as vehicles and non-vehicles. This approach can be used through the scanning process. All these ANN-based solutions are not analyzed from the point of view of energy consumption, which is a strong advantage of SNNs as well as an important factor in autonomous systems.

Zhou et al. and Wang et al. [118] [123] [124] propose SNN-based solutions for object detection and classification from LIDAR data. In [118], image classification is carried out using a modest-sized feed-forward network applied to a simple synthetic dataset obtained by simulating a LIDAR sensor that perceives different types of roads populated with cars, pedestrians and trucks. In [123] the authors expand this approach using feed-forward and convolutional SNNs for object recognition in three datasets: simulated LIDAR, objects extracted from the KITTI dataset and DVS_barrel [125]. Furthermore, [124] contains a Spiking Convolutional Neural Network approach to object detection, by incorporating the YOLOv2 architecture. This is tested on the bird's eye view and 3D detection benchmarks from the KITTI dataset.

Both [124] and [123] contain promising estimates for the energy consumption the networks would have if they were running on neuromorphic hardware. All of these SNNs networks have the constraint that each neuron is only allowed to spike once per input, which complicates effective implementations of

recurrent connections and processing incoming information as a time series.

3.3 Methods

3.3.1 Model

To sequentially process information like raw LIDAR data, we turn to recurrent SNNs as described by Yin et al. [20]: the network consists of LIF neurons simulated at discrete time points. These SNNs contain layer-wise recurrent connections and are trained using Backpropagation-Through-Time (BPTT) and Surrogate Gradients.

The neuronal behaviour of LIF neurons can be simulated at discrete time points using the forward-Euler first-order exponential integrator method, as described by (3.2)–(3.3):

$$u_t = \alpha u_{t-1} + (1-\alpha) R_m I_t - S_{t-1}\theta \tag{3.2}$$

$$\alpha = \exp(-dt/\tau_m), \tag{3.3}$$

where at each time step t, u_t represents the membrane potential of the cell, I_t the input current and S_t the binary activity of the neuron. The parameter α relates to the decay of the membrane potential, while R_m is the membrane resistance, τ_m the decay time constant of the membrane potential and θ is the threshold.

Neural adaptation is a feature of the biological neuron that can be added to the LIF model of (3.2). Its implementation is expressed by (3.4)–(3.6):

$$\theta = b_0 + \beta\eta_t \tag{3.4}$$

$$\eta_t = \rho\eta_{t-1} + (1-\rho) S_{t-1} \tag{3.5}$$

$$\rho = \exp(-dt/\tau_{adp}). \tag{3.6}$$

Here, the threshold becomes a function of time, with a fixed minimal value of b_0 and an adaptive contribution $\{\beta\eta_t\}$. The adaptive contribution decays with parameter ρ, which also depends on an additional time constant τ_{adp}. The behaviour of the variables in the LIF neurons with and without adaptation given a modulated spike-rate input is shown in Fig. 3.1.

The LIF and Adaptive LIF neurons are stateful cells that can be building blocks for feed-forward and recurrent networks. By replacing the gradient computation of the neuronal activation with a Surrogate Gradient, such networks

can be trained using Backpropagation-Through-Time in the same manner as more traditional ANNs [65]. Our SNNs, in particular, use a difference of Gaussian functions as a Surrogate Gradient for the neuron activation, where the highest values reside around the point where the membrane potential is equal to the threshold [19]. The network is implemented using the PyTorch framework [108] and relies on the autograd feature for all gradient computations.

3.3.2 Dataset

The KITTI Vision Benchmark [126] was created by collecting data on static and dynamic objects in diverse scenarios, such as city streets, rural areas and freeways. The data was captured using a vehicle equipped with multiple sensors; the laser scanner used for developing this benchmark is the commercially available Velodyne HDL-64E. The data from the various sensors was synchronized and processed in order to form a set of benchmarks [127], where we are using the 3D object detection benchmark.

We have developed an object extraction pipeline that creates 2D images of the same size, each containing one object, where the values of the pixels are derived from the depth of the 3D points[1]. For this, the 3D point cloud is projected to 2D and all the points that fall inside the specified window are selected. Each window is of constant size and contains the bounding box annotation of the respective object in its center. Annotated objects that are too big or too small in comparison with the window size are ignored. Then all the 3D points whose 2D projections fall inside the computed window are associated with the respective object. Thus, each object is encoded by a 2D image where the value of each pixel is associated with the depth of the corresponding point. The pixels where no points are projected are assigned the value 0, and in the cases where multiple points in the 3D point cloud project to the same pixel, the highest value is used (as in practice this gave most contrast and worked best). The images are then

[1] `https://github.com/ancadiana23/RSNN_LIDAR`

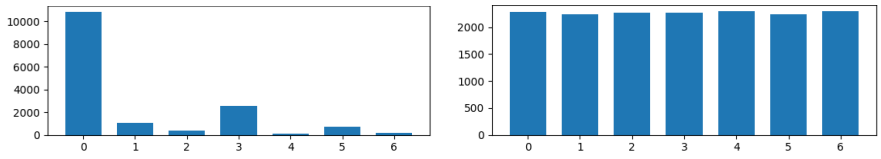


Figure 3.2: Example distribution by class of the original and resampled dataset.

scaled down by a predefined factor to have the resulting objects of a particular size, and the depth values in the dataset are normalized to be between 0.0 and 1.0.

Analyzing the class and value distribution of the resulting dataset shows that the dataset class distribution is severely imbalanced (Fig. 3.2). To resolve this, we use the torch WeightedRandomSampler in order to create a balanced training set. Testing uses the test set with the original class distribution in most experiments, unless explicitly stated that the test set is also resampled.

We also find that the resulting pixel values are mostly clustered in the interval [0.1, 0.4], with a long tail using the remaining dynamic range [0.4, 1.0]. To increase the effectively used dynamic range, we apply a depth clip to all the values, where we empirically select the maximum depth from the original values. These depth values are then normalized to be between 0.0 and 1.0. The result of this process can be observed in Fig. 3.3, which presents the value

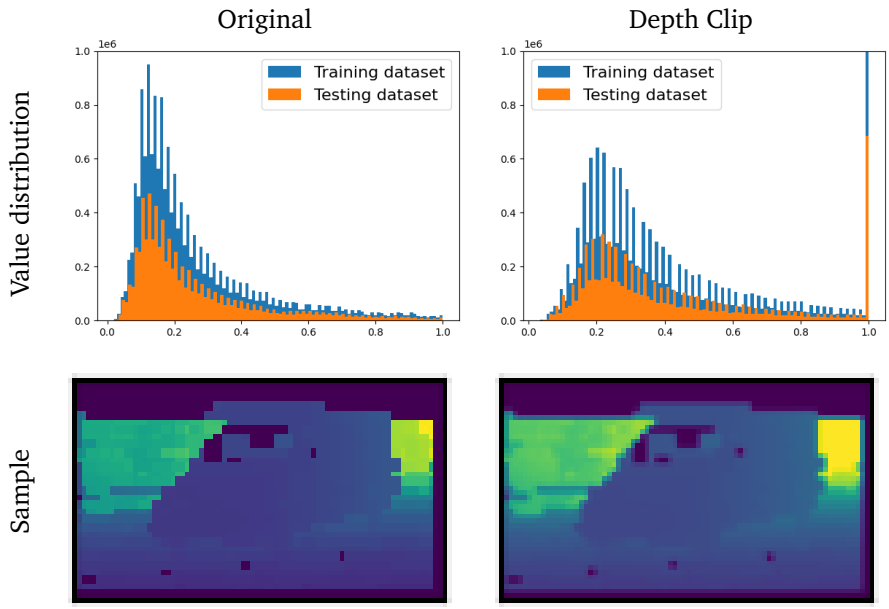


Figure 3.3: Value distribution of the original and clipped data, followed by example images.

distributions and sample images from the original and depth clipped set.

The dataset generation configuration was empirically determined by visually inspecting the results. The size of the windows applied to the 2D image is 232x392, which are then resized by a factor of 8, resulting in samples of size 30x50. We ignore any object that is either larger than the window or smaller than a fifth of the window. Since the testing examples of the KITTI benchmark are not publicly available, we used the first 5000 frames for constructing a training dataset and the last 2481 examples as our test dataset. As there can be multiple objects in the same frame, this procedure results in having 15892 extracted objects for training and 7926 objects in our testing set.

3.3.3 Depth encoding

When classifying the objects in the dataset, we use a sliding window whose stride and size can be adjusted through model hyper-parameters. For each point the sensor measures the delay between emitting and receiving a laser beam, from which the distance to the reflecting surface can be computed. Thus, the input to an object classification model can be either a binary signal at the receiving time, or the value of the time delay as a floating point number.

In order to simulate these two input types, we use the temporal and rate encoding respectively. Rate encoding signifies that the network receives the values described in the dataset section directly as current injection into the input neurons.

Our temporal encoding method receives float values which are translated into binary spikes, at specific discrete times, thus relaying one input window over multiple time steps. The value interval (0.0, 1.0] is split into equal segments, and then each value is replaced by a binary signal at

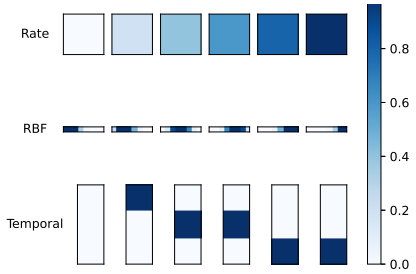


Figure 3.4: Encoding methods. The first row shows 6 equally spaced values between 0.0 and 1.0, which are examples of the raw values. The following rows show the output of different encoding methods, where the horizontal space represents neighbouring cells in one layer, and the vertical space signifies different time steps [128].

a time step corresponding to the value's segment (Fig. 3.4). The values strictly equal to 0.0 correspond to inactive input cells.

In addition to the two input types, rate and temporal encoding, we also use Radial Basis Functions (RBF) [128]. This type of Receptive Field Coding is a population coding method that uses multiple cells with different receptive fields to encode each input value. The receptive fields used here are described by Gaussian functions of identical shapes but with different means. Thus, for small input values, the leftmost neurons are activated, while larger values move the activation towards the right of the neuron population (Fig. 3.4).

Experimentally, we find that the effectiveness of RBF encodings varies depending on the number of neurons used to represent each pixel. Smaller values, such as 4 neurons per pixel, show less over-fitting but also decreased performance. Higher values however result in lower training accuracy and much lower testing accuracy. For the experiments with RBF encoding, we selected 8 neurons per pixel as the optimal value.

Table 3.1: Weight sparsity comparison. For each sparsity level here are the total number of parameters (weights), the number of non-zero weights, and the number of neurons. The default network here contains 128 and 64 neurons in the hidden layers.

Sparsity Level	#Param	#Non-Zero Param	#Neuron
0 %	35520	35520	199
30 %	48564	34033	235
50 %	66831	33343	278
70 %	107235	32082	356
80 %	158444	31584	436
90 %	307242	30884	613

3.3.4 Sparsity

Recent work has shown that making Recurrent Neural Networks sparser can be advantageous not only for expediting the training process but also for improving performance [129]. Distinctively, in [60, 130], the authors show that keeping a constant number of non-zero parameters while increasing the size and sparsity of a network leads to increased accuracy. They do so by creating larger networks at the beginning of the training process and populating a binary mask throughout the training process, always setting the smallest weights to zero.

Here, we test this idea in the context of RSNNs, by randomly constructing a binary mask at the beginning of the training process. Starting with the default sized networks (two layer RSNNs, see section IV), the layers are then scaled and masked with a probability p such that they use an approximately constant number of non-zero parameters. The mask remains constant throughout the training process. The sparsity and corresponding network sizes are shown in Table 3.1.

3.3.5 Truncated Backpropagation-Through-Time

Even with our down-sampled dataset, event sequences are relatively long, creating issues in training and testing the model with BPTT; these issues would only be exacerbated by scaling to a real-life object detection task. To mitigate this, we implement Truncated Backpropagation-Through-Time (T-BPTT). T-BPTT splits a long sequence into several segments and then computes the gradient and weight updates only using one segment at a time. This training optimization is selected when training the model and the number of backpropagation steps is controlled by a hyper-parameter, *backprop_step* .

In detail, BPTT follows the flow of information through time in reverse, unfolding the recurrent influences in the network, including self-recurrences in the neural cell: Fig. 3.5 shows the connections of one cell unfolded in time. The membrane potential and adaptive threshold create self-recurrent connections for each cell, which are depicted in pink and yellow, respectively. There is also a layer-wise recurrent connection (green) from the spikes in the previous time step to the membrane potential of the current one. The inset in Fig. 3.5 zooms in on the actual computation, and shows the interaction between variables from (3.2) – (3.6): the red parameters are learned, and the recurrent connections are represented in the same colors in all three views of the network.

To apply truncation of the unfolded sequence, we detach the internal parameters of the model from the computational graph every *backprop_step* number of time steps, such that future gradients with respect to these variables will not be backpropagated past the specified time point. Importantly, all variables have to be detached, including the membrane potentials and the firing activities of the cells, as they also persist through multiple time steps. Moreover, these variables effectively construct the memory of the network, and thus should maintain their values and not be reinitialized at truncation.

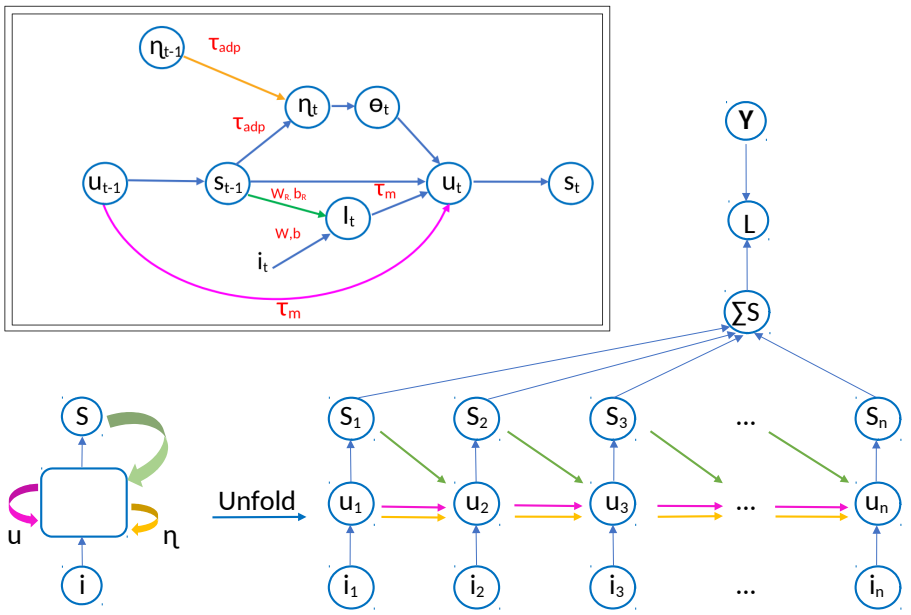


Figure 3.5: Recurrent SNN with adaptive LIF neurons. The membrane potential (pink) and the adaptive threshold (yellow) are represented as self recurrent connections. The layer-wise recurrent connections are represented by the green arrows. Inset: detailed graph for spiking neuron variables and parameters.

3.4 Experiments

To test the encoding and optimization methods, as well as the efficiency of the RSNN approach, we construct a default RSNN containing two hidden layers of 128 and 64 cells, respectively, and two traditional ANNs for comparison: an RNN with two LSTM layers of 128 neurons and one fully-connected layer (architecture optimized for the task); the second is a CNN with 2 convolutional layers and 3 fully connected ones. For all the Recurrent Networks (RNN and SNN) we split the images into time sequences using a sliding window, which by default covers one row at a time (window = stride = 50 px).

Table 3.2: Accuracy and number of active neurons for the best performing models (mean results from 5 trained networks). All RSNN networks are of default parameter size.

Model	Test Accuracy	Spikes/Frame
Rate	80.4	1127
Rate + sparsity(90%)	79.3	3355
RBF	83.7	1235
RBF + sparsity(90%)	**88.2**	3458
Temporal Coding (4)	75.7	2944
LSTM	84.0%	N/A
CNN	92.3%	N/A

3.4.1 Overall Performance

The RSNN and RNN models and their accuracy on the original test set are presented in Table 3.2. The RSNN results surpass those of the LSTM and approach that of the frame-based CNN: pairing RBF encoding and 90% sparsity results in 88.2% test accuracy.

At the same network size in terms of parameters, rate-coding and temporal coding are substantially less accurate. At the same time however, rate-coding in particular requires substantially fewer spikes per frame for classification. We also find that for non-sparse RBF-RNNs, modest sized networks are needed, with most network sizes performing approximately as well as the default-size network (Fig 3.9). Overall, these results confirm that RSNNs are able to successfully classify LIDAR data.

3.4.2 Early classification

The sequential nature of processing in RSNNs enables the network to classify objects already during the scanning process and provide tentative results before the end of the input sequence. Fig. 3.6 shows the classification performance along the sequence, where the model's accuracy is plotted at different sequence steps or scanning percentages. The left plot presents a monotonically increasing accuracy over the length of the sequence, demonstrating that the RSNN is able to classify partial inputs correctly. Viewed as a function of the object, the right plot shows the accuracy of the network against the percentages of object area scanned. For this plot, we use the original 2D bounding boxes and plot

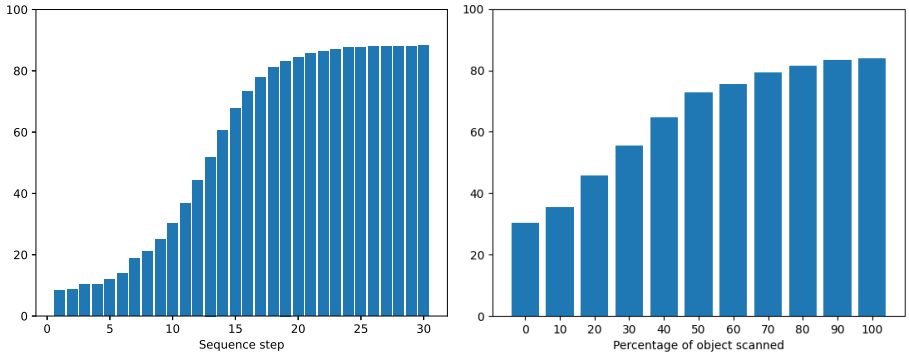


Figure 3.6: Network performance over sequence steps (left) and over the object scan percentage (right).

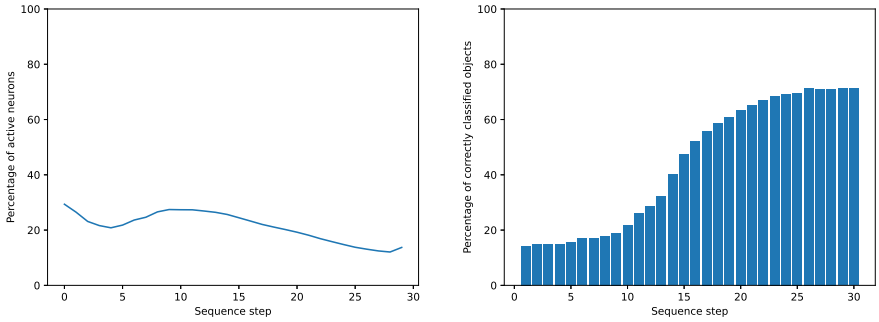


Figure 3.7: Network activity (left) and performance (right) as a function of sequence steps.

percentages scanned in order to normalize over objects of multiple sizes – we note again the monotonous increase in accuracy. An example of the resulting online classification is shown in Fig. 3.8.

We furthermore study the average activity in the network while processing a sequence. Fig. 3.7 presents on the left the mean percentage of active neurons over the sequence, and on the right the model's accuracy over the sequence. We find that on average around 20% of neurons emit a spike at any time step. There is also a noticeable increase in activity between time steps 5 and 15, which corresponds to a steep increase in accuracy.

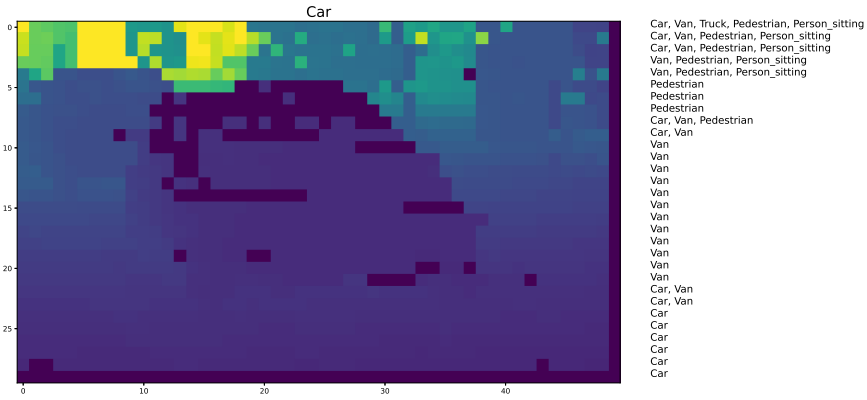


Figure 3.8: Examples of network output over row by row sequences. Above the image is the correct label of the object, and to the right of each row is the network output after processing the row.

These findings suggest that, in a practical setting, not only would the RSNN model be able to classify objects before the end of a 360° scan, but it would also be capable of predicting its type before the object is fully scanned.

3.4.3 Sparsity

As shown in Table 3.2, we find that (when using the RBF encoding) network performance is improved for increased network sparsity. This technique maintains a fixed number of network non-zero parameters - and increases the number of neural cells. At the same time, we find that networks with sparse connections show increasing average network activity, especially for very sparsely connected networks (Fig. 3.10).

To further explore its effects, we plot the mean and standard devia-

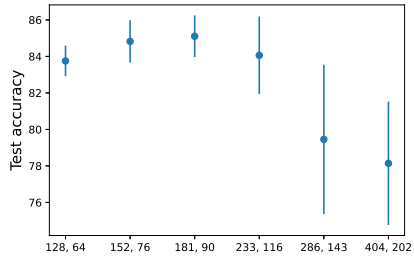


Figure 3.9: Non-sparse RBF network accuracy as a function of network size.

tion of the network's accuracy for network size and sparsity, as calculated over the last 25 epochs of training and test accuracy. Fig. 3.11 shows that sparsity improves the network accuracy when using the RBF encoding, but it is not beneficial with rate-coded input. Our experiments revealed that sparsity works well with rate coding only on the resampled test set.

We speculate that larger sparser networks manage better class separation due to their larger latent space. When having a highly skewed testing set, class separation is less important than learning to classify the majority class, and thus the network has good results without using sparsity; however, in this case, spreading the predictions over more classes can lead to worse results.

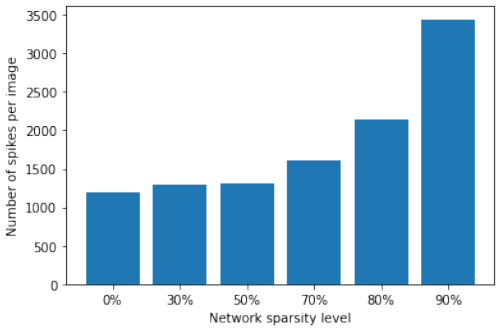


Figure 3.10: Number of spikes per image for RBF models at different levels of sparsity.

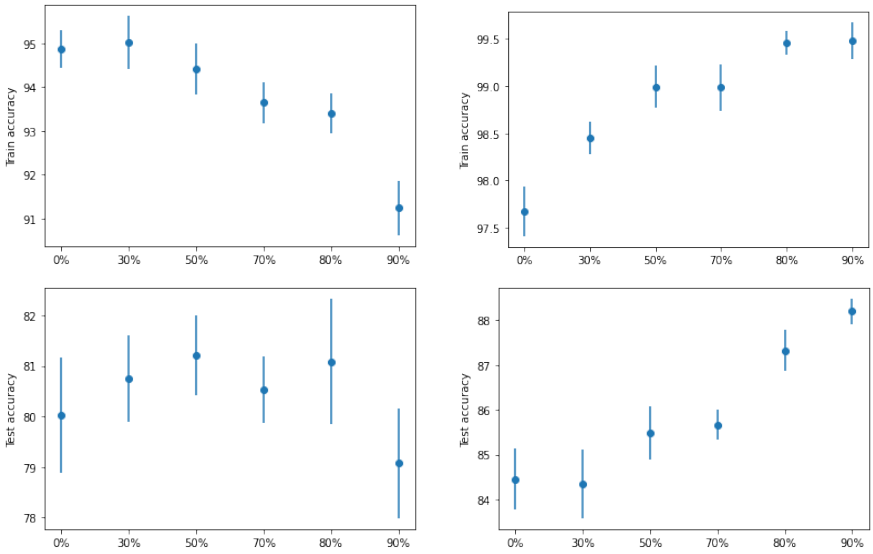
Overall, using the RBF encoding consistently results in better accuracy, especially on the skewed (original) test set, which leads us to believe that this coding method improves the class separation for the majority class. Thus, when both sparsity and RBF are employed they improve the performance of the network regardless of the testing set.

3.4.4 Truncated Backpropagation-Trough-Time

The effect of Truncated Backpropagation-Trough-Time on the training process is shown in Fig. 3.12, plotting the training time in seconds from experiments with various input encodings for the default RSNNs. Invariably, we find that using Truncated Backprop Through Time leads to shorter training times.

The effect of T-BPTT on testing and training set accuracies is shown in Fig. 3.13, when using RBF coding and varying length for the truncation intervals. We find that truncation leads to relatively good results, approaching full BPTT accuracy, with 10 steps proving to be the sweet spot for this particular encoding method.

We conclude that Truncated Backpropagation-Through-Time always shortens the training time of the model. Concerning performance, however, it can be



a) Rate b) RBF

Figure 3.11: Sparsity effect on the original test set for rate-coding (left) and RBF(8) coding (right).

a useful tool in certain cases.

3.4.5 Energy Requirements

Next to classification accuracy, energy requirements of the networks are an important secondary concern, as the practical applications of such systems are Autonomous System often subject to energy constraints.

We study the energy requirements of the network in more detail by approximating the number of required computations. Table 3.3 calculates the energy usage of a particular network layer with respect to the energy of multiply-and-accumulate operations (MAC) and accumulate operations (AC) [20].

With these equations, we calculate the corresponding energy use for the LSTM, default RSNN, and best performing RSNN. As calculated in Table 3.4, we find that then that the theoretical energy required by the default RSNN is one order of magnitude lower than that of a traditional recurrent network even

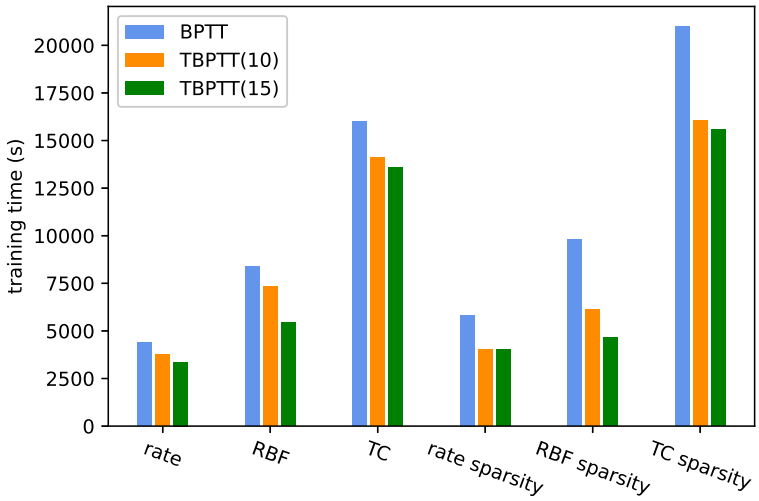


Figure 3.12: Training times using BPTT and T-BPTT with 10 and 15 steps truncation intervals.

before factoring in the sparse activation rates. Using 20% firing rates the energy usage of both SNNs falls well below that of the RNN.

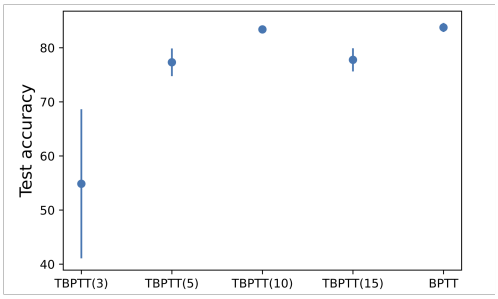


Figure 3.13: The mean testing accuracies of the last 25 epochs of training, using T-BPTT with different truncation intervals and RBF(8) encoding.

We can also approximate the energy requirements the SNN would have when implemented on actual neuromorphic hardware, using the data published by the Intel Loihi Team [72]. They measured the energy of a neuronal update when the cell is active/inactive as 81 pJ and 52 pJ, respectively. Our default RSNN has 199 cells resulting in 11nJ energy usage per step when running with 20% firing rate. The larger, better performing RSNN with 90% sparsity and RBF(8) encoding reaches 35nJ usage with the same fir-

Table 3.3: Energy requirement formulas for different types of NN layers. Here E_{AC} and E_{MAC} represent the energy cost per AC and MAC respectively, m is the input size and n is the output size of the layer, while Fr is the firing rate.

Network	**Layer type**	**Energy/Layer**
SNN	Adaptive	$(mn+2n)E_{AC}Fr+2nE_{MAC}$
	Adaptive & Recurrent	$(mn+nn+2n)E_{AC}Fr+2nE_{MAC}$
Traditional NN	FC	mnE_{MAC}
	LSTM	$(4mn+4nn+3n)E_{MAC}$

Table 3.4: Approximate amount of energy required to process one input (one line of an image) using a traditional recurrent neural network, compared to a spiking neural network. The networks used in these computations are the ones presented in the previous experiments. The theoretical approximations are computed using the computation complexity formulas from Table 3.3, while the practical ones depend on the Intel Loihi 1 measurements [72].

	Encoding	Theoretical	Practical
RNN	-	223,872 E_{MAC}	-
SNN 128 × 64	Rate	35,918FrE_{AC} +398E_{MAC}	11 nJ
SNN 90% sparse	RBF	449,868FrE_{AC} +1226E_{MAC}	35 nJ

ing rate. In the scenario where a Velodyne sensor was moving at 20Hz with 50 images per 360° frame, running the larger SNN would require about 1mW. The caveat of this calculation is that a Loihi implementation of our approach has yet to be constructed. Thus, we do not have the certainty that this network's time steps would match onto the neuronal updates used in their measurements.

3.5 Discussion

We find that we can train compact RSNNs to successfully processes LIDAR data using various types of input encoding, where adding an RBF-style population encoding significantly improves the network's results. We also find that network sparsity improves the RSNN performance when using a) rate encoding and training and testing sets with the same class distributions; b) RBF encoding.

Even though the number of non-zero parameters stays the same, the number of cells grows with the size of the network. At the same time, the number of spikes per frame grows rapidly with the increasing number of neurons in the network, which leads to higher computational and energy requirements for such sparse networks.

Overall, we demonstrated that RSNNs successfully classifies the provided LIDAR data, reaching over 88% accuracy on the testing set, exceeding the performance of classical RNNs like LSTMs and approaching that of non-sequential CNNs which process the entire frame. We find moreover the RSNNs can preemptively classify examples before the end of the sequence and even before the target object is fully perceived.

While theoretically highly efficient, the practical approximation of energy consumption is a rough estimation, and an actual implementation on neuromorphic hardware, such as the Intel Loihi [72], would be required to confirm this estimate. Building on the developed solution, we also note that the input to the network can be expanded to also contain the reflectance values provided by LIDAR, which may improve performance further. Other multimodal solutions could also be considered, such as aggregating data from the multiple sensors equipped in an Autonomous Driving Setup.

Chapter 4

Training SRNN Through Forward Propagation Through Time

Significance: **Based on the previous works, we observed that: First, Challenge 1 is an unavoidable problem in the gradient-based training algorithm, but we can mitigate it by simplifying the gradient computation process (i.e., solving Challenge 2); Second, the memory problem described in Challenge 3 cannot be fundamentally solved by the algorithm in the framework of offline updates.**

Therefore Challenges 1-3 are difficult to address by previous BPTT-like algorithms. In this chapter, we will use FPTT [131] as an online training algorithm to train SNNs, and then fundamentally solve Challenges 2-3 and alleviate the errors introduced by Challenge 1. We present a new gated spiking neuron model – the Liquid Time-Constant neuron – to obtain higher performance SNNs when training of FPTT. In the FPTT framework, we can train arbitrarily long sequences with constant memory. In addition, FPTT also allows us to train larger and deeper neural networks with a more complex neural circuit. With the help of FPTT, we demonstrate a first end-to-end trained SNNs (SPYv4) for object detection and recognition network.

This chapter is based on publication "Accurate online training of dynamical spiking neural networks through Forward Propagation Through

Time" [132].

Abstract: *With recent advances in learning algorithms, recurrent networks of spiking neurons are achieving performance competitive with vanilla recurrent neural networks. Still, these algorithms are limited to small networks of simple spiking neurons and modest-length temporal sequences, as they impose high memory requirements, have difficulty training complex neuron models, and are incompatible with online learning. Here, we show how recently developed 'Forward-Propagation-Through-Time' (FPTT) learning combined with novel Liquid Time-Constant spiking neurons resolves these limitations. Applying FPTT to networks of such complex spiking neurons, we demonstrate online learning of exceedingly long sequences while outperforming current online methods and approaching or outperforming offline methods on temporal classification tasks. FPTT's efficiency and robustness furthermore enables us to directly train a deep and performant spiking neural network for joint object localization and recognition, demonstrating for the first time the possibility of training large-scale dynamic and complex spiking neural network architectures.*

4.1 Introduction

The binary, event-driven and sparse nature of communication between spiking neurons in the brain holds great promise for flexible and energy-efficient AI. Recent work has demonstrated effective and efficient performance from spiking neural networks (SNNs) [133], enabling competitive and energy-efficient applications in neuromorphic hardware [134] and novel means of investigating biological neural architectures [135, 136]. This success stems principally from the use of approximating surrogate gradients [63, 65] to integrate networks of spiking neurons into auto differentiating frameworks like Tensorflow and Pytorch [108], enabling the application of standard learning algorithms and in particular Back-Propagation Through Time (BPTT).

The imprecision of the surrogate gradient approach however expounds on the existing drawbacks of BPTT. In particular, BPTT has a linearly increasing memory cost as a function of sequence length T, $\Omega(T)$ and suffers from vanishing or exploding backpropagating gradients, which limits its applicability on long time sequences [131] and large-scale SNN models [137]. Alternative approaches like real-time recurrent learning (RTRL) [138] similarly exhibit excessive computational complexity, and low complexity approximations to BPTT like e-prop [77] or OSTL [139] at best approach BPTT performance. Training

on long temporal sequences in SNNs is of particular importance when the tasks require a high temporal resolution, for instance to match the physical characteristics of low-latency clock-less neuromorphic hardware [134, 140].

Kag et al. [131] recently introduced a novel online learning algorithm, Forward Propagation Through Time (FPTT), for online learning in recurrent networks, demonstrating better generalization on many benchmark tasks compared to BPTT, improving in particular on long sequence training in Long Short-Term Memory networks (LSTMs). FPTT differs from BPTT in that it does not calculate a gradient through time, and instead considers learning-through-time as a coordinated consensus problem. Using regularized synaptic tags, FPTT enables immediate, online learning in RNNs similar to feedforward networks, eliminating the problematic dependence of the gradient calculation in BPTT on the products of partial gradients along the time dimension: FPTT exhibits linear $\Omega(T)$ computational cost per sample. For training recurrent SNNs however, as we demonstrate, a straightforward application of FPTT on long sequence training does not improve performance as it does in [131], and we observed this also with vanilla RNNs. Therefore, we deduce that FPTT particularly benefits from the gating structure inherent in LSTM-style gated RNNs, which is lacking in vanilla RNNs and SRNNs.

Taking inspiration from the concept of Liquid Time-Constant (LTCs) [141], we introduce a novel class of spiking neurons, the Liquid Time-Constant Spiking Neuron (LTC-SN), where time-constants internal to the neuron are dynamic and input-driven in a learned fashion, resulting in functionality similar to the gating operation in LSTMs. We integrate these neurons in networks that are trained with FPTT and demonstrate that the resulting LTC-SNNs outperform various SNNs trained with BPTT on long sequences while enabling online learning and drastically reducing memory complexity. We show this for several classical benchmarks that can easily be varied in sequence length, like the Add-task and the DVS-GESTURE benchmark [67, 106]. We also show how FPTT-trained LTC-

Algorithm	Gradient Update	parameter Update	Memory Storage
BPTT	$\Omega(c(T)T)$	$\Omega(1)$	$\Omega(NT)$
RTRL	$\Omega(c(T)T^2)$	$\Omega(T)$	$\Omega(NT)$
e-prop / OSTL	$\Omega(c(1)T)$	$\Omega(T)$	$\Omega(N)$
FPTT	$\Omega(c(1)T)$	$\Omega(T)$	$\Omega(N)$

Table 4.1: Computational complexity of gradients, parameter updates and memory storage per sample, with N the batch-size. Computational expense increases as the length T of the sequence grows. i.e. $c(1) < c(T)$ after [131].

SNNs can be applied to large convolutional SNNs, where we demonstrate novel state-of-the-art for online learning in SNNs on a number of standard benchmarks (S-MNIST, R-MNIST, DVS-GESTURE) and to near (Fashion-MNIST, DVS-CIFAR10) or exceeding (PS-MNIST, R-MNIST) state-of-the-art performance as obtained with offline BPTT.

Finally, the training and memory efficiency of FPTT enables us to directly train SNNs in an end-to-end manner at network sizes and complexity that was previously infeasible. We demonstrate this in a new You-Look-Only-Once (YOLO) LTC-SNN architecture for object detection on the Pascal Visual Object Classes (Pascal VOC) dataset [142]. Object detection is a challenging task, as it involves accurate multi-object identification and precise bounding box coordinate computation; previous SNN approaches have been limited to either ANN-to-SNN conversions [143–145], requiring many thousands of time-steps at inference time, or small scale and inefficient SNNs with performance far removed from that of modern ANNs [124]. Our FPTT-trained YOLOv4 [146] implementation – SPYv4 – uses 21 layers, 6.2M LTC spiking neurons, and 14M parameters to achieve new state-of-the-art for SNNs, exceeding the performance of converted ANNs while achieving extremely low latency.

With FPTT and LTC spiking neurons, we demonstrate end-to-end online training of large and high-performance SNNs comprised of complex spiking neuron models that were previously infeasible.

Related Work

The problem of training recurrent neural networks has an extensive history [54, 147, 148]. To account for past influences on current activations in a recurrent network, the network can be unrolled, and errors are computed along the paths of the unrolled network. The direct application of error-backpropagation to this unrolled graph is known as Backpropagation-Through-Time [54]. BPTT needs to wait until the last input of a sequence before being able to calculate parameter updates and, as such, cannot be applied in an online manner. Alternative online learning algorithms for RNNs have been developed, including Real-Time Recurrent Learning (RTRL) [138] and approximations thereof [149]. RTRL however is prohibitive in time and memory complexity, and while approximations improve complexity (see [139], and Table 4.1), they yield variable and task-dependent accuracy deficits compared to exact gradients [77, 150].

Spiking neural networks are neural networks composed of spiking neurons: stateful neural units that communicate using binary values, i.e. spikes. Their

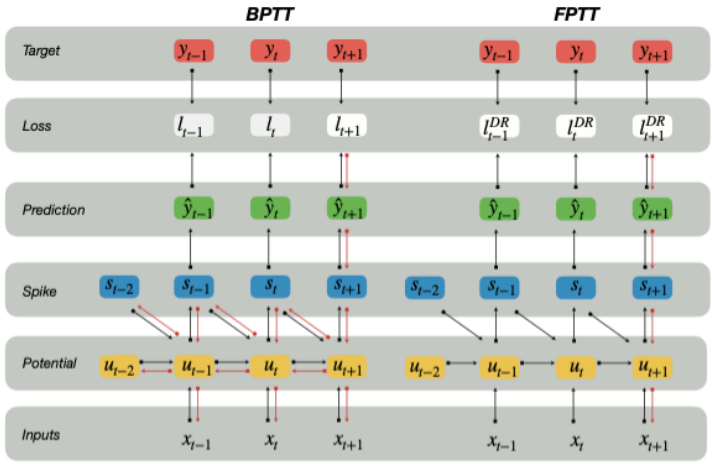


Figure 4.1: Roll-out of the computational graph of a spiking neuron as used for BPTT (left) and that of FPTT(right)

state is determined by current and past inputs, and this state then determines the (binary) value of the emitted output through a spiking mechanism. The discontinuity of the spiking mechanism challenges the application of error-backpropagation, which can be overcome using continuous approximations [16, 63], so-called "surrogate gradients" [65]. Recurrent SNNs trained with surrogate gradients and BPTT now achieve competitive performance compared to classical RNNs [20, 67, 133]. In these and other studies, more intricate spiking neuron models, like those including adaptation, outperformed less complex models such as standard Leaky-Integrate-and-Fire neurons [77]. Additionally, training internal spiking neuron's model parameters like the time-constants of adaptation and membrane-potential decay then further improves performance [20, 133].

Still, the application of BPTT in SNNs has several drawbacks: in particular, BPTT accumulates the approximation error of surrogate gradients along time. Moreover, the spike-triggered reset of some state variables in typical spiking neuron models (e.g. the membrane potential) causes a vanishing gradient when applying BPTT. We found these effects to be particularly problematic when training networks with complex and more biologically detailed neuron models like Izhikevich and Hodgkin-Huxley models. Furthermore, because the

SNN training accuracy heavily depends on hyperparameters, obtaining convergence using BPTT in SNNs is non-trivial.

For spiking neural networks, approximations to BPTT like e-prop [77] and Online Spatio-Temporal Learning (OSTL) [139] achieve linear time complexity and have proven effective for many small-scale benchmark problems, ATARI GAMES and also large-scale networks like cortical microcircuits [151]. In terms of trained accuracy, however, none of these approximations have been shown to outperform standard BPTT and, applications to deeper networks use approximate spatial credit assignment approaches like learning-to-learn [77].

4.2 Methods

As introduced in [131], FPTT can be implemented directly on the computational graph of SNNs, similar to BPTT approaches. This is illustrated in Fig. 4.1; FPTT intuitively provides a more robust and efficient gradient approximation for spiking recurrent neural networks than BPTT, as FPTT simplifies the complex gradient computation path in SNNs. This potentially lessens surrogate gradients' cumulative effect, avoiding or reducing the gradient vanishing or explosion problem.

As we will show, FPTT applied directly to SNNs like the Adaptive Spiking Recurrent Neural Networks (ASRNNs) from [133] converges but without the learning improvements reported for RNNs [131] – and we observed this also for vanilla non-spiking RNNs (not shown). As FPTT was successfully applied to RNNs with gating structures (LSTM), we introduce the Liquid Time-Constant Spiking Neuron (LTC-SN) as a spiking neuron model with a similar gating structure. We observe that in spiking neurons, the time-constant of the membrane potential acts similar to the forget-gate in LSTMs; the LSTM forget-gate however is dynamically controlled by learned functions of inputs. Inspired also by Hasani et al. [141], the LTC-SN's internal time-constants are a learned function of the inputs and hidden states of the network (illustrated in Fig. 4.3).

4.2.1 Forward Propagation Through Time.

FPTT considers learning as a consensus problem between the network updates at different time-steps, where the network update at each single time-step needs to move toward the same converged optimal weights. To achieve this, FPTT updates the network parameters W by optimising the instantaneous risk function

ℓ_t^{dyn}, which includes the ordinary objective $\mathscr{L}_t$ and also a dynamic regularisation penalty $\mathscr{R}_t$ based on previously observed losses $\ell_t^{dyn} = \mathscr{L}_t + \mathscr{R}_t$ (see Appendix A). In FPTT, the empirical objective $\mathscr{L}(y_t, \hat{y}_t)$ is the same as for BPTT, representing a function of the gap between target values y_t and real time predictions $\hat{y}_t$.

The FPTT-specific dynamic regularization is controlled by a form of running average of all the weight-updates calculated so far $\bar{W}$, where the update schema of this regularizer $\mathscr{R}_t$ is as follows:

$$\mathscr{R}(W_t) = \frac{\alpha}{2} \parallel W_t - \bar{W}_t - \frac{1}{2\alpha}\nabla l_{t-1}(W_t) \parallel^2 \tag{4.1a}$$

$$W_{t+1} = W_t - \eta \nabla_W l(W_t) \tag{4.1b}$$

$$\bar{W}_{t+1} = \frac{1}{2}(\bar{W}_t + W_{t+1}) - \frac{1}{2\alpha}\nabla l_t(W_{t+1}). \tag{4.1c}$$

Here, the state vector $\bar{W}_t$ summarises past losses: the update is first a normal update of parameters W_t based on gradient optimization with fixed $\bar{W}_t$, after which $\bar{W}_t$ is optimized with fixed W_t. This approach allows the RNN parameters to converge to a stationary solution of the traditional RNN objective [131]. Note that in Equation 4.1c, $\nabla l_t(W_{t+1})$ is estimated as in [131], avoiding propagating the gradient through Equation 4.1b.

The FPTT learning process requires the acquisition of an instantaneous loss l_t at each time step. This is natural for sequence-to-sequence modeling tasks and streaming tasks where a loss is available for each time step; for classification tasks however, the target value is only determined after processing the entire time series. To adapt FPTT to classification tasks, or rather, to perform online classification tasks, a divergence term was introduced [131] in the form of an auxiliary loss to reduce the distance between the prediction distribution $\hat{P}$ and target label distribution Q:

$$l_t = \beta l_t^{CE}(\hat{y}_y, y) + (1-\beta) l_t^{div}, \tag{4.2}$$

where $\beta \in [0,1]$; l_t^{CE} is the classical cross-entropy for a classification loss and $l_t^{div} = -\sum_{\bar{y}} Q(\bar{y}) \log \hat{P}(\bar{y})$ is the divergence term. We use the auxiliary loss in all experiments as in [131] with $\beta = \frac{t}{T}$.

Training networks of spiking neurons. To apply FPTT to SNNs, we define the spiking neuron model and specify how BPTT and FPTT are applied to such

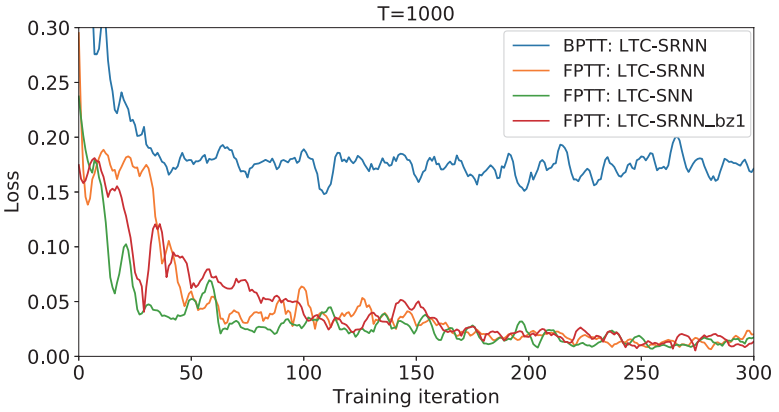


Figure 4.2: Example learning curves on the Add-task of LTC-SRNN trained with BPTT, FPTT, FPTT and batchsize 1 ("LTC-SRNN_bz1", red curve) and non-recurrent LTC-SNN trained with FPTT (green curve).

networks. All networks were trained using batches to exploit GPU parallelism; reduction to batch-size=1 yielded similar results (e.g. Figure 4.2).

An SNN is comprised of spiking neurons which operate with non-linear internal dynamics. These non-linear dynamics consist of three main components:

(1) Potential Updating: the neurons' membrane potential u_t updates following the equation:

$$u_t = f(u_{t-1}, x_t, s_{t-1} \| W, \tau) \tag{4.3}$$

where τ is the set of internal time constants and W is the set of associated parameters, like synaptic weights. The membrane potential evolves based on previous neuronal states (e.g. potential u_{t-1} and spike-state $s_{t-1} = \{0,1\}$) and current inputs x_t. Training the time constants τ in the spiking neurons is known to optimize performance by matching the neuron's temporal dynamics of the task [20, 67].

(2) Spike generation: A neuron will trigger a spike $s_t = 1$ when its membrane potential u_t crosses a threshold θ from below, described as a discontinuous function:

$$s_t = f_s(u_t, \theta) = \begin{cases} 1, & \text{if } u_t \geq \theta \\ 0, & \text{otherwise} \end{cases} \tag{4.4}$$

(3) Potential resting: When a neuron emits a spike ($s_t = 1$), its membrane potential will reset to resting potential u_r:

$$u_t = (1 - s_t)u_t + u_r s_t, \tag{4.5}$$

where in all experiments, we set $u_r = 0$.

BPTT for SNNs BPTT for SNNs amounts to the following: given a training example $\{x, y\}$ of T time steps, the SNN generates a prediction $\hat{y}_t$ at each time step. At time t, the SNN parameters are optimized by gradient descent through BPTT to minimize the instantaneous objective $\ell_t = \mathscr{L}(y_t, \hat{y}_t)$. The gradient expression is the sum of the products of the partial gradients, defined by the chain rule as

$$\frac{\partial \ell_t}{\partial W} = \frac{\partial l_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial s_t}\frac{\partial s_t}{\partial u_t}\sum_{j=1}^{t}\left[\prod_{m=j}^{t}\left(\frac{\partial u_m}{\partial u_{m-1}} + \frac{\partial u_m}{\partial s_{m-1}}\frac{\partial s_{m-1}}{\partial u_{m-1}}\right)\right]\frac{\partial s_{m-1}}{\partial W}, \tag{4.6}$$

where the partial derivative of spiking $\frac{\partial s_t}{\partial u_t}$ is calculated by a surrogate gradient associate with membrane potential u_t. Here, we use the Multi-Gaussian surrogate gradient function $\hat{f}'_s(u_t, \theta)$ [133] to approximate this partial term.

The computational graph of BPTT is illustrated in Table 4.1 and shows that the partial derivative term depends on two pathways. The product of these partial terms may explode or vanish in RNNs, and this phenomenon becomes even more pronounced in SNNs as the discontinuous spiking process is approximated by the continuous surrogate gradient and the incurred gradient error accumulates and amplifies.

FPTT for SNNs. FPTT can be used for training SNNs by minimizing the instantaneous loss with the dynamic regularizer $\ell_t^{dyn} = \mathscr{L}(y_t, \hat{y}_t) + \mathscr{R}(\bar{W}_t)$. The update function Equation (4.6) then becomes:

$$\frac{\partial \ell_t^{dyn}}{\partial W} = \frac{\partial l_t^{dyn}}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial s_t}\frac{\partial s_t}{\partial u_t}\frac{\partial u_t}{\partial W}. \tag{4.7}$$

Compared to Equation (4.6), Equation (4.7) has no dependence on a chain of past states, and can thus be computed in an online manner.

4.2.2 Liquid Time-Constant Spiking Neurons

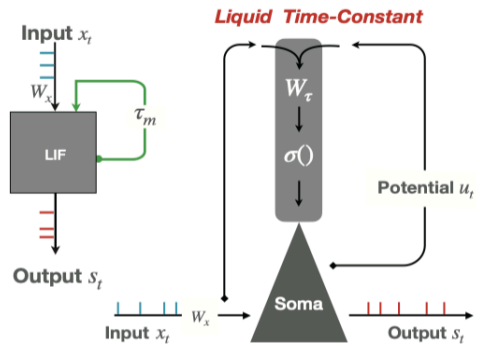


Figure 4.3: LIF neuron and Liquid Time-Constant spiking neuron (LTC-SN) as recurrent network structures.

The LTC-SN is modeled as a standard adaptive spiking neuron [77, 133] where the time constants τ (here, membrane time constant τ_m and adaptation time constant τ_{adp}) are a dynamic and learned function of internal dynamic state variables like membrane potential u and deviation [77] b. In the network, time-constants are either calculated as a function $\alpha = exp(-dt/\tau_m) = \sigma(Dense[x_t, u_{t-1}])$, for non-convolutional networks, or using a 2D convolution for spiking convolutional networks, $\alpha = exp(-dt/\tau_m) = \sigma(Conv(x_t + u_{t-1}))$, where we use a sigmoid function $\sigma(\cdot)$ to scale the inverse of the time constant to a range of 0 to 1, ensuring smooth changes when learning. This results in a Liquid Time-Constant Spiking Neuron defined as:

$$\tau_{adp}\, update : \rho = exp(-dt/\tau_{adp}) = \sigma(Dense_{adp}[x_t, b_{t-1}]) \tag{4.8a}$$
$$\tau_m\, update : \alpha = exp(-dt/\tau_m) = \sigma(Dense_m[x_t, u_{t-1}]) \tag{4.8b}$$
$$\theta_t\, update : b_t = \rho b_{t-1} + (1-\rho)s_{t-1}; \theta_t = 0.1 + 1.8b_t \tag{4.8c}$$
$$u_t\, update : du = -u_{t-1} + x_t; u_t = \alpha u_{t-1} + (1-\alpha)du \tag{4.8d}$$
$$spikes_t : s_t = f_s(u_t, \theta) \tag{4.8e}$$
$$resetting : u_t = u_t(1-s_t) + u_{rest}s_t, \tag{4.8f}$$

where the neuron uses an adaptive threshold θ_t as in the Adaptive Spiking Neurons [77], resting potential $u_{rest} = 0$, and time-constants τ_m and τ_{adp} are computed as liquid time-constants.

I.e., for adapting spiking neurons, the time-constant of the membrane potential decay, τ_m and the time-constant of the adaptation decay, τ_{adp} can be made dynamic. A spiking neuron with such varying internal dynamics can respond in flexible and unexpected ways to input currents: as shown in Fig. 4.4**a**, depending on the effective weights W_τ, the current-spike-frequency response curve can be muted-and-saturating, near-linear, or rapidly increasing-and-then-

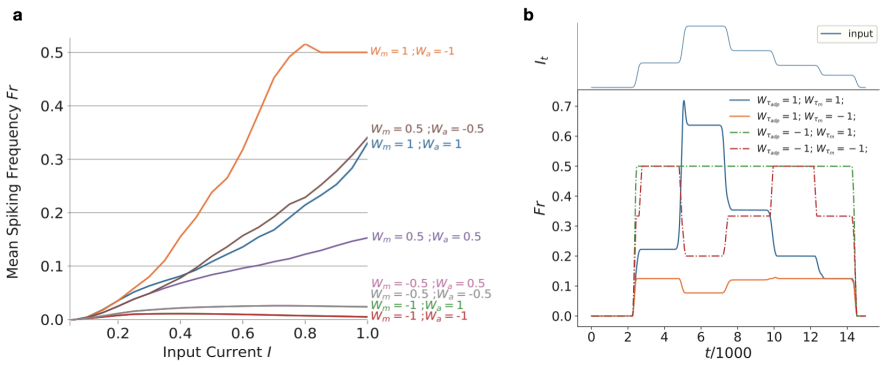


Figure 4.4: **a**, F-I curve of a LTC-SN spiking neuron model for different combinations of effective LTC weights $W_m = \{-1 - 0.5, 0.5, 1\}$ and $W_a = \{-1 - 0.5, 0.5, 1\}$ associated with respective dynamic time constant functions σ subject to input current I. **b**, Response Fr of LTC-SN to varying current input I (top) for different combinations of effective LTC weights $W_{\tau_m} = \{-1, 1\}$ and $W_{\tau_{adp}} = \{-1, 1\}$.

saturating; when subject to a dynamically varying input current, a higher input current into LTC-SN units can result in a reduced firing rate, and transient dynamics can be absent or present (Fig. 4.4**b**, samples of trained behavior are shown in Fig. 4.5).

4.3 Experiments

We demonstrate the effectiveness of FPTT training for LTC-SNNs on several classical benchmarks, including the Add-task as in [131] and several established SNN benchmarks (the DVS-GESTURE and DVS-CIFAR10 classification tasks, and the Sequential, Sequential-Permuted, rate-based and Fashion MNIST classification tasks). We moreover demonstrate how the memory efficiency of FPTT enables training large-scale LTC-SNNs for applications like object localization.

4.3.1 Datasets

The **Add-task** [88] is used to evaluate the ability of RNNs to maintain long-term memory. An example data point consists of two sequences (x_1, x_2) of

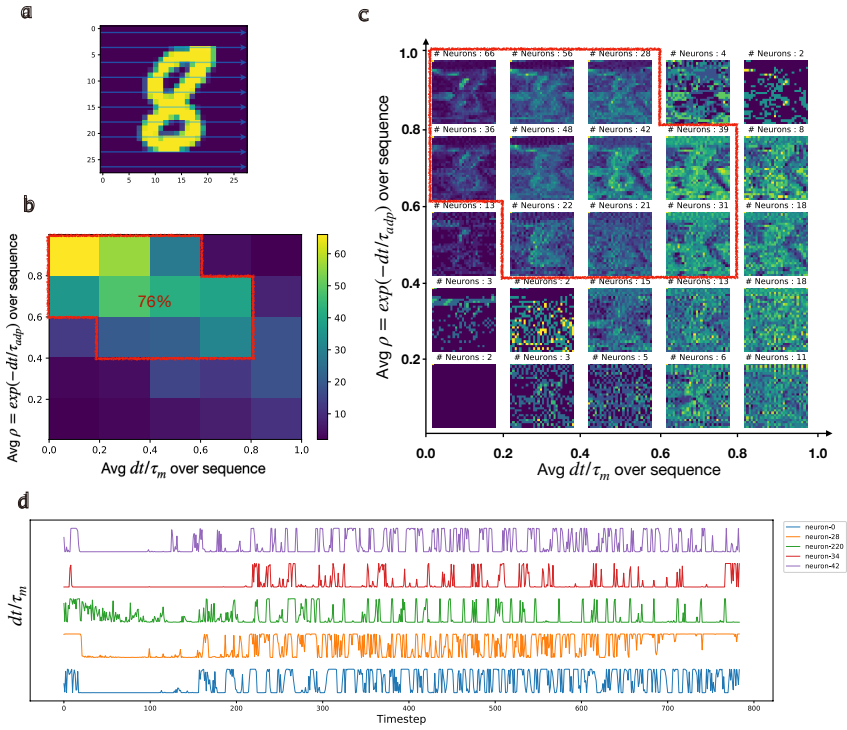


Figure 4.5: Dynamic neural responses of LTC-SNs. (a) an MNIST sample is sequentially fed into the LTC-SNN, pixel-by-pixel along the row-direction. (b) for illustration, we calculated the histograms of the resulting average dt/τ values after training for this single sample (c) average responses of neurons in terms of firing rate binned by $dt/\tau_m, exp(-dt/\tau_{adp})$ values (d) Example of dynamics of inverse time constant dt/τ_m for four randomly selected neurons during presentation of the sequence.

length T and a target label y. The sequence x_1 contains values sampled uniformly from [0, 1], x_2 is a binary sequence containing only two 1s, and the label y is the sum of the two entries in the sequence x_1, where $x_2 = 1$. The **IBM DVS Gesture** dataset [106] consists of 11 kinds of hand and arm movements of 29 individuals under three different lighting conditions captured using a DVS128 camera. **DVS-CIFAR10** is a widely used dataset in neuromorphic vi-

sion, where the event stream is obtained by displaying moving images of the CIFAR-10 dataset [152]. The **Sequential and Permuted-Sequential** MNIST (S-MNIST, PS-MNIST) datasets were developed to measure sequence recognition and memory capabilities of learning algorithms: the S-MNIST dataset is obtained by reshaping the classical MNIST 28x28 pixel images into a set of one-dimensional sequences consisting of 784 time-steps per sample, where pixels are then sequentially entered one-by-one as input to the network; the PS-MNIST dataset is generated by performing a fixed permutation on the S-MNIST dataset. Theoretically and in practice, PS-MNIST is a more complex classification task than S-MNIST because it lacks temporally correlated patterns. The **rate-coded MNIST** (R-MNIST) is an SNN-specific benchmark where a biologically inspired encoding method is used to generate the network input that produces streaming events (a spike train), by encoding the grey values of the image with Poisson rate-coding [45]. We also applied FPTT-trained LTC-SCNNs to the traditional static **MNIST** and **Fashion-MNIST** datasets for comparison with other models trained offline. Here, we input pixel values directly as injected current into the spiking input layer of the network, repeated 20 times to mimic a constant input stream. For object localization, we used the **PASCAL VOC** benchmark [142] which is comprised of standard 416416 RBG images with 20 different objects and corresponding bounding box locations.

Datasets and networks For the Add-task, the trained networks consist of 128 recurrently connected neurons of respective types LTC-SN (LTC-SNN), LSTM, or Adaptive Spiking Neuron [77] (ASRNN), and a dense output layer with only 1 neuron.

For the DVS-GESTURE dataset, each frame is a 128-by-128 size image with 2 channels. Each sample in the DVS-GESTURE dataset was split into fixed-duration blocks as in [67], where each block is averaged to a single frame. This conversion results in sequences of up to 1000 frames depending on block length. As input for the shallow SRNN as used in the increasingly long sequence setting, we first down-sample the frame of a 128-by-128 image into a 32-by-32 image by averaging each 4-by-4 pixel block. The 2D image at each channel is then flattened into a 1D vector of length 1024. For each channel of the image, the network consists of a spike-dense input layer consisting of 512 neurons as an encoder, where the information of each channel is then fused into a 1D binary vector through concatenation. This 1D vector is then fed to a recurrently connected layer with 512 hidden neurons. Finally, a leaky integrator is applied to generate predictions, resulting in a network architecture

[1024,1024]-[512D,512D]-512R-11I[1]. All networks were trained for 30 epochs using the Adam optimizer with initial learning rate 3e-3, using the same initialization schemes and learning-rate scheme for all networks to compare the effect of neural units. Hyperparameters for ASRNNs were taken from [133].

To achieve high performance with SCNNs, we applied FPTT with LTC-SNs to high-performance architectures from the literature, where we used hyperparameter settings for the surrogate gradient from [20] and from [131] for FPTT training. Specifically, in (P)S-MNIST, we used the networks architecture from [77], a shallow network with one recurrent layer comprised of 512 hidden neurons and an output layer consisting of 10 (number of classes) leaky integrator neurons. The FPTT-trained LTC-SNNs are optimized using Adam [82] with a batch size of 128 using 200 training epochs. We set the initial learning rate to 3e-3 and decay by half after 30, 80, and 120 epochs. For the S-MNIST and PS-MNIST tasks, we also find that the leak time-constants of the output units after training averaged 87ms ±13ms (S-MNIST) and 65ms±12ms (PS-MNIST), substantially shorter than the sequence length and demonstrating that the recurrent network maintains working memory. For R-MNIST, we follow the architecture from [139]: an SNN with two hidden layers of 256 neurons, each followed by 10 output neurons. The SNN is given 20 presentations of the image, after which the classification is determined. For the static MNIST and Fashion MNIST datasets, we apply the architecture from [67]: an SCNN with 3 convolutional layers, 1 Dense layer and 1 leaky Integrator output layer (ConvK3C32S1P1-MPK2S2-ConvK3C128S1P1-MPK2S2-ConvK3C256S1P1-MPK2S2-512D-10I[1]). The resulting LTC-SCNN network was then optimized using FPTT and Adamax with a batch size of 64 and an initial learning rate of 1e-3. For the DVS-GESTURE and DVS-CIFAR10 datasets, we also follow the high-performance architecture from [67] and use 20 sequential frames, where the network makes a prediction only after reading the entire sequence. The LTC-SCNN thus has a structure ConvK7C64S1P3-MPK2S2-ConvK7C128S1P3-MPK2S2-ConvK3C128S1P1-MPK2S2-ConvK3C256S1P1-MPK2S2-ConvK3C256S1P1-MPK2S2-ConvK3C512S1P1-MPK2S2-512D-11I. These networks were optimized through Adamax [82] with a batch size of 16 and initial learning rate of 1e-3.

For all networks, to measure GPU memory consumption, the GPU management interface “nvidia-smi” was used.

[1]To describe the network structure, we follow standard conventions as follows: ConvK7C64S1P1 represents the convolutional layer with $output channels = 64$, $kernel size = 7$, $stride = 1$ and $padding = 3$. MPK2S2 is the max-pooling layer with $kernel size = 2$ and $stride = 2$. 512D and 512R represents the fully connected layer and recurrent layer respectively with $output features = 512$. 10I indicates the leaky integrator with the $output feature = 10$.

Spiking Tiny YOLOv4 – SPYv4. The SPYv4 network follows the Tiny YOLO-v4 architecture [146, 153]. The backbone of the network consists of three CSP-Blocks in the cross-stage partial network. In contrast to regular additive residual connections, the CSP-Block is spike-based rather than event-based as residual connections are constructed by concatenation rather than using an adding operation, ensuring that only binary spikes are used for information transfer between layers. Raw pixel values are fed directly into the network as input currents. As the object recognition task necessitates a more precise output vector to draw the bounding box, we use a single standard conventional convolutional layer instead of a spiking convolutional layer.

We trained and evaluated our model on Pascal VOC dataset [142] as was done in [143]: the network was trained using a combination of VOC2007 and VOC2012 (16551 images), and evaluated on the validation dataset of VOC2007 (4952 images).

For target detection, we use a threshold on the Intersection Over Union (IOU), i.e., the ratio of the intersection of the prediction box and the ground truth box of the target, to indicate whether the detection is correct. A target is considered to be correctly detected when the IOU exceeds the threshold. The average precision (AP) is then computed as the average of all the precision for all possible recall values, and the mAP is the average of the AP of multiple classification tasks. Training maximizes the mAP@ϑ: the mean average precision of the calculated bounding box exceeding an overlap threshold ϑ over the actual boundaries of the object in the dataset, where we use $\vartheta = 0.5$.

We applied both mosaic data augmentation [153] and label smoothing during training to obtain better performance. In the mosaic enhancement, the network uses a mosaic of 4 images during training instead of a single image; this is applied only during the first 200 training cycles. The network was optimized by Adagrad with a batch size of 32 and an initial learning rate of 1e-3.

Loss	τ_m	$\tau_m(x)$	$\tau_m(x,u)$
τ_{adp}	0.17	0.0027	0.0025
$\tau_{adp}(x)$	0.17	0.0024	0.003
$\tau_{adp}(x,b)$	0.16	0.0021	0.0019

Table 4.2: Ablation study of LTC-SN performance on Add Task where either τ_m, τ_{adp}, or both, are trained (τ_m,τ_{adp}), dynamic from external input x ($\tau_m(x)$,$\tau_{adp}(x)$) or both external and internal input ($\tau_m(x,u)$,$\tau_{adp}(x,b)$).

4.3.2 Results

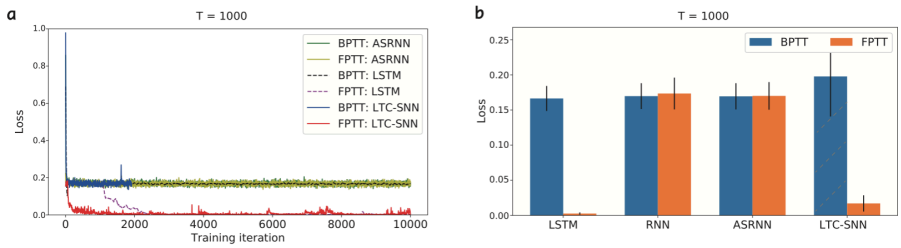


Figure 4.6: **a**, Example loss-curves for networks trained on the Add Task for a sequence of length 1000 time steps. The loss of LTC-SNN becomes NaN after 2000 iterations when training with BPTT. **b**, plots the loss of the last 100 training iterations averaged over 5 runs for sequence lengths.

FPTT-SNN requires Liquid Time-Constant Spiking Neurons. We apply both BPTT and FPTT to the Add-task as originally studied in [131] to illustrate the need for more complex spiking neurons like LTC-SNs when applying FPTT learning. We do this for various network types, including non-spiking LSTMs as a baseline, ASRNNs [133], and LTC-SNNs.

For a long adding sequence of length 1000, example loss-curves are plotted in Fig. 4.6**a** and averaged converged losses in Fig. 4.6**b**. We find that as in [131], a standard LSTM network trained with BPTT fails to converge to zero loss, while the same network does converge using FPTT. For SNNs, we find that ASRNNs trained with either FPTT or BPTT do not fully converge, and for the LTC-SNNs trained with BPTT learning rapidly diverges due to exploding gradients. LTC-SNNs trained with FPTT however successfully minimize the loss: ablating the LTC-SN dynamics, we find that the input-dependent dynamic gating of the membrane-potential time-constant is critical for convergence (Table. 4.2), and the memory provided by the LTC-SN self-recurrence does in fact suffice for solving this task (Fig. 4.2).

FPTT allows for longer sequence training. Next, we study the ability of FPTT-trained LTC-SNNs to learn increasingly long sequences. For this, we use the DVS-GESTURE dataset and systematically investigate the performance of a fixed architecture shallow SRNN model on increasingly many frames sampled from the same gesture signals as in [67], ranging from 20 to 1000 frames. For

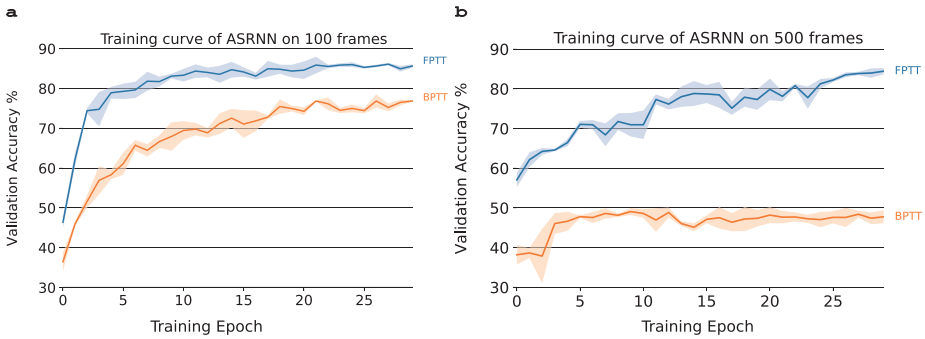


Figure 4.7: Learning curve of ASRNN trained via FPTT and BPTT on **a** 100 frames, and **b** 500 frames

each frame-encoded dataset, we train various networks types for a fixed number of epochs with an identical number of neural units, and report the best performance for BPTT and FPTT trained networks; networks either converged before the final epoch or diverged (Fig 4.7**a**,**b**).

The results for different sampling frequencies are shown in Fig. 4.8. Of all methods and networks, the LTC-SNN trained using FPTT achieves the best accuracy in all cases, also outperforming standard (BPTT-trained) LSTMs. Furthermore, the FPTT-trained LTC-SNNs and the ASRNNs exhibit constant accuracy over the whole range of sequence lengths, where the LTC-SNNs consistently outperform the ASRNNs.

In contrast, the accuracy of both LTC-SNNs and ASRNNs trained with BPTT quickly deteriorates as sequence length increases. For the LTC-SNNs, the net-

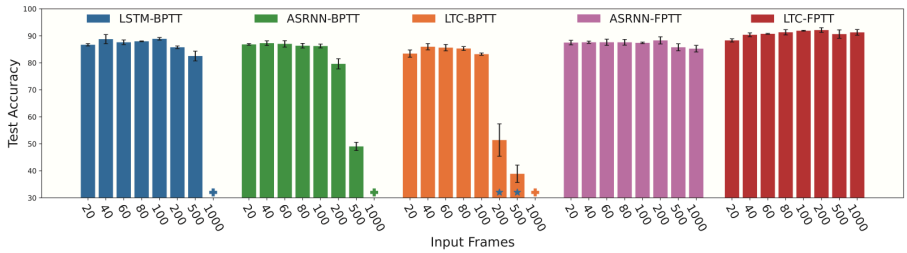


Figure 4.8: The Bar chart of Performance comparison between BPTT and FPTT on the DVS gesture dataset.

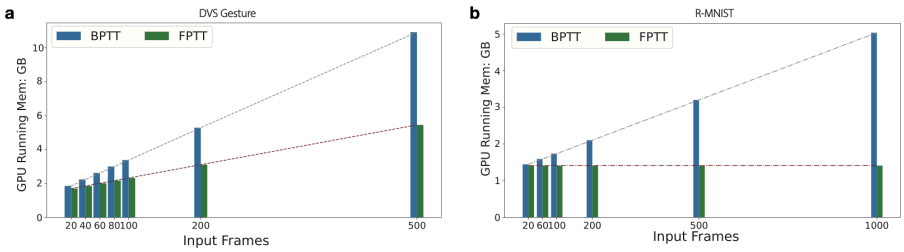


Figure 4.10: **a**, Memory Efficiency: GPU Memory required for network training of the DVS Gesture dataset for different sequence length. **b** GPU Memory cost required for training of the R-MNIST dataset.

works failed to converge for frame-lengths 200 and 500, and best validation accuracy is reported in Fig. 4.8. For the baseline standard LSTM, this effect is also there, albeit more moderate, as performance decreases from 88.9% at 100 frames to 82.5% at 500 frames. This suggests that indeed the gradient approximation errors in SNNs add up when training with BPTT. The plot also illustrates the memory-intensiveness of BPTT: when applied to the 1000 frame-length data, GPU-memory (24GB) was insufficient for training LSTM, ASRNN and LTC-SNN.

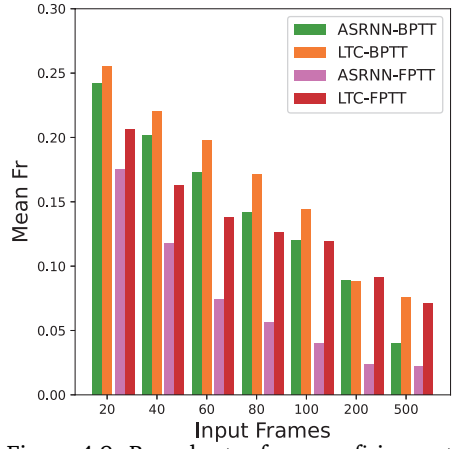


Figure 4.9: Bar chart of mean firing rate (fr) comparison between BPTT and FPTT on the DVS gesture dataset.

Comparing sparsity (average firing rate) for the different SNN models, we find no meaningful differences (Fig. 4.9); parameter-matched networks showed similar performance as unit-matched networks, and the inclusion of an auxiliary loss [131] only aided BPTT-trained LSTMs but not BPTT-trained SRNNs. Making only the membrane-decay time-constant dynamic has a small negative impact (ASRNN– Table 4.3).

Table 4.3: **Performance comparison between BPTT and FPTT on the DVS gesture dataset.** Each number in the table is the average of three runs. All networks have an equal number of neural units unless indicated otherwise. (*) denotes that training diverged; reported accuracy is the best accuracy before divergence. (+) denotes out-of-GPU-memory when training. The $ASRNN^{+}$ is an ASRNN with the same number of parameters as the LTC-SNN. LTC^{-} denotes an LTC-SNN network where only the membrane time constant is dynamic, and the adaptation time constant is a learned fixed parameter. For the Eprop ASRNN, we use a single layer network with 1400 neurons, matching the number of parameters in the LTC-SNN.

Frames	BPTT						FPTT			Eprop
	LSTM+Aux	LSTM	LTC-SNN+Aux	LTC-SNN	ASRNN+Aux	$ASRNN^{+}$+Aux	LTC^{-}	LTC-SNN	ASRNN	ASRNN
20	86.69±0.43	82.29±2.46	83.42±1.35	84.37±2.27	86.82±0.31	80.78±1.99	87.83±1.21	**90.39**±0.71	87.51±0.85	75.46±1.73
40	88.77±1.71	84.95±0.71	85.96±1.16	84.37±1.24	87.29±0.87	82.99±0.70	88.53±0.57	**90.39**±0.71	87.61±0.43	75.92±0.87
60	87.61±0.86	85.15±0.75	85.62±1.18	83.91±0.71	87.02±1.19	81.77±0.34	88.08±1.40	**90.74**±0.16	87.62±1.15	75.42±0.86
80	87.97±0.14	84.83±1.42	85.30±0.71	80.44±3.6	86.34±0.87	76.84±0.85	88.66±1.14	**91.31**±0.98	87.60±1.06	77.54±1.34
100	88.89±0.49	83.79±0.71	83.21±0.43	78.70±0.91	86.22±0.71	74.61±1.25	89.93±1.13	**91.89**±0.16	87.40±0.28	75.46±1.15
200	85.76±0.49	81.87±2.58	51.39±6.0	(43.98±2.35)*	79.62±1.89	65.89±1.69	89.24±1.56	**92.13**±0.87	88.31±1.33	75.33±1.02
500	82.52±1.82	78.81±1.5	38.89±3.22	(36.46±1.5)*	49.03±1.52	52.43±1.32	86.69±0.43	**90.64**±1.56	85.76±1.30	75.04±0.97
1000	+	+	+	+	+	+	90.05±1.56	**91.28**±1.05	84.24±1.23	74.86±1.08
Param	4.2M	4.2M	4.7M	4.7M	1.6M	4.7M	3.2M	4.7M	1.6M	4.8M

FPTT Requires Less Memory. FPTT-trained LTC-SNNs require increasingly less memory as sequence length increases as measured on GPU. For the DVS-GESTURE dataset (Fig. 4.10**a**), FPTT memory-use is both less and increases less rapidly compared to BPTT as a function of the number of frames used per data sample. FPTT's increasing memory use can be attributed to the rapidly inflating size of the frame-encoded dataset, increasing in size from 7.4 GB for 20 frames to 368.1 GB for 1000 frames. We validated this by training an LTC-SNN network on the R-MNIST classification problem, where longer sequences are simulated by showing the same sample for an increasing number of frames. We then find, as expected, that the memory required for FPTT training remains fixed. At the same time, BPTT memory-use linearly increases (Fig. 4.10b).

FPTT with LTC Spiking Neurons improves over Online Approximate BPTT. To demonstrate the power of FPTT as an online training method, we used state-of-the-art deep spiking convolutional network architectures (SCNNs) for standard sequential benchmarks from the literature and trained these architectures with LTC-SN neurons and FPTT.

In Table 4.4, we compare the LTC-SNNs to existing state-of-the-art online and offline SNNs. We find that LTC-SCNNs trained with FPTT consistently and substantially outperform SNNs trained with online BPTT approximations like OSTL and e-prop. Compared to offline BPTT approaches, the FPTT-trained LTC-

Table 4.4: Test accuracy of deep SRNNs/SCNNs on various tasks. **Bold-faced** denotes state-of-the-art (SoTa) online performance, ***slanted bold*** denotes overall SNN state-of-the-art. For S-MNIST, PS-MNIST and RMNIST, FPTT-LTC uses identical network structures as in [77, 139], for the other benchmarks the network structure matches [67].

Task	**Online baseline**		**This work**		**Offline SoTa**	
S-MNIST	-	-	FPTT+LTC	**97.37%**	BPTT+ASRNN [133]	***98.7***%
PS-MNIST	-		FPTT+LTC	***94.77%***	BPTT+ASRNN [133]	94.3%
RMNIST	OSTL + SNU [139, 154]	95.34%	FPTT+LTC	***98.63%***	BPTT+SNU [154]	97.72%
MNIST	-		FPTT+LTC	**99.62%**	BPTT+PLIF [67]	***99.72%***
Fashion MNIST	-		FPTT+LTC	**93.58%**	BPTT+PLIF [67]	***94.38%***
DVS-GESTURE	DECOLLE+SNN [155]	95.54%	FPTT+LTC	**97.22%**	BPTT+PLIF [67]	***97.57%***
DVS-CIFAR10	-		FPTT+LTC	**73.2%**	BPTT+PLIF [67]	***74.8%***

Table 4.5: Memory efficiency and total training time. (*): the reported number is obtained using a halved batch size compared to the other entries to fit into GPU memory

GPU memory				
	S-MNIST	R-MNIST	MNIST	DVS-Gesture
BPTT	11.1GB	1.5GB	9.67GB	15.72GB(*)
FPTT	1.9GB	1.4GB	2.23GB	3.75GB
Training Time per epoch				
BPTT	2518s	192s	362s	108s
FPTT	1233s	204s	384s	112s

SNNs achieve new state-of-the-art for SNNs (PS-MNIST, R-MNIST) or achieve close to similar performance (S-MNIST, DVS-GESTURE), DVS-Cifar10); additionally, the memory requirements for FPTT vs. BTTP trained networks were lower by up to a factor of 5 while the training time was only slightly longer (Table 4.5).

4.3.3 Large-scale Object-detection: Spiking YOLO

The memory efficiency of FPTT-trained LTC-SNNs enables us to train SNNs of comparable complexity as modern ANNs: we demonstrate this by training a large spike-based object-detection model based on the Tiny YOLO-v4 architecture [146, 153]. The 'You-Only-Look-Once' (YOLO) architecture calculates both bounding boxes locations and object identities for all identifiable objects in an

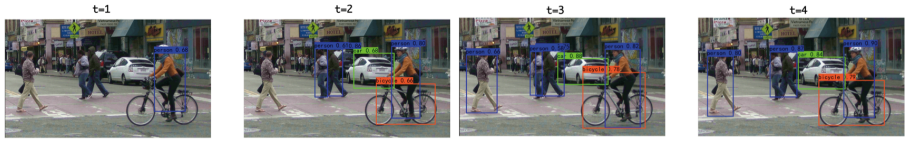


Figure 4.11: An example of object recognition process of Spiking-YOLO on a sequence of images. Objects are localized and identified for each image independently.

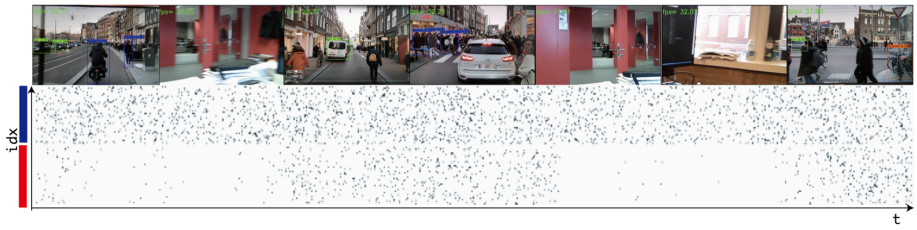


Figure 4.12: SPYv4 applied to streaming video (example images on top), with raster plot of spiking activity below. Spikes are shown for 100 random select neurons from the in first and last layer.

image using a single pass through a deep neural network.

Our SPiking tiny Yolo-v4 network - SPYv4 - has 19 spiking convolutional layers with about 6.2M spiking neurons, 2 convolutional output layers and 14M parameters in total, illustrated in Fig.4.14**a**. This makes it both larger and deeper than previous end-to-end trained spiking models. Training a single time-step in the network requires less than 14GB of GPU-memory, and as BPTT scales linearly with the number of time-steps, learning with BPTT in such a large network over multiple time-steps is infeasible.

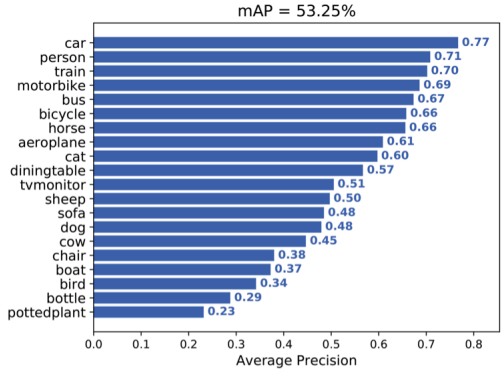


Figure 4.13: The mAP@.5 calculation in the classification and recognition of 20 different kinds of objects in SPYv4 on VOC07.

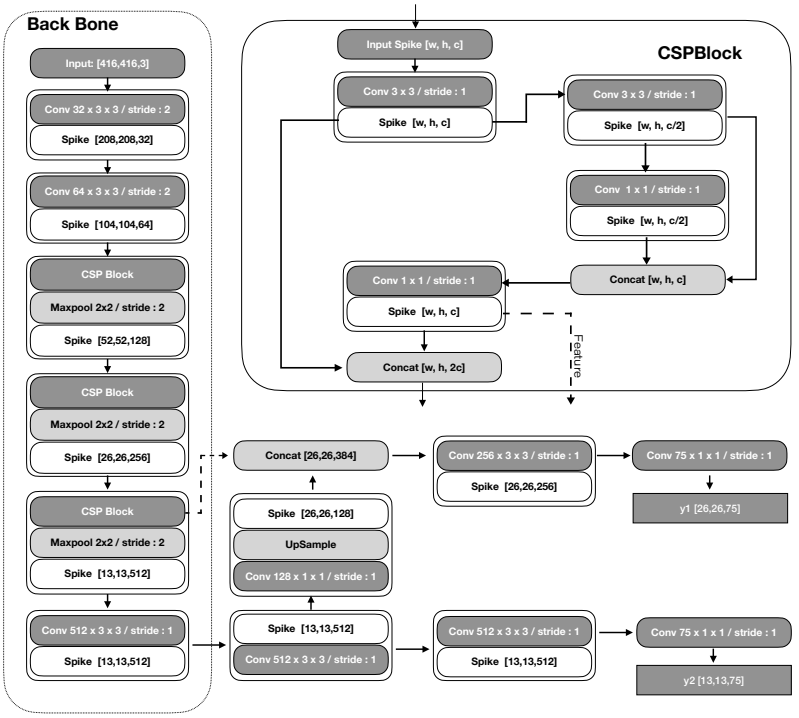


Figure 4.14: Spiking YOLOv4 (SPYv4) neural network architecture.

To carry out object detection, the network uses multiple reads of the input image, e.g. 4 or 8 times, as time-steps in the network to obtain the final result; trained with FPTT, SPYv4 achieves a mean Average Precision (mAP) of 51.38% at 4 reads and 53.25% at 8 reads (Fig. 4.13) on the VOC dataset (see Methods); Fig. 4.11 shows examples of the detected and classified objects. Neural activation in the network is highly sparse, with about 10% of neurons active on average at each time-step. When receiving direct camera inputs images, inference achieves about 60 time-steps per second on an NVIDIA RTX3090 equipped workstation, corresponding to processing 7 or 15 images per second. Fig. 4.12 shows example activity from neurons from respectively a shallow (near input) and deep network layer: neurons in the deeper layer fall silent when irrelevant stimuli are shown, while neurons closer to the inputs remain

active. Earlier work like Spiking-YOLO [143] achieved mAP 51.83% with 8000 simulation time-steps; our SPYv4 network thus outperforms these networks in performance, sparseness, and latency.

4.4 Discussion

We showed how a recently proposed novel training approach for recurrent neural networks, FPTT, can be successfully applied to long sequence learning with recurrent SNNs using novel Liquid Time-Constant Spiking Neurons. Compared to BPTT, FPTT is compatible with online training, has constant memory requirements, and can learn longer sequences. The increased memory efficiency of FPTT allows for training much larger SNNs as was previously feasible, as we demonstrated in the SPYv4 network for object detection. In terms of accuracy, FPTT outperforms online approximations of BPTT like OSTL and e-prop, and enabled a first demonstration of online learning in tasks like DVS-CIFAR10. When training large convolutional LTC-SNNs with FPTT, excellent performance is achieved, approaching or exceeding offline BPTT-based solutions using corresponding network architectures – LTC-SNN specific architecture searches may improve results further.

To achieve efficient and accurate online learning with FPTT, we introduced Liquid Time-Constant Spiking Neurons (LTC-SNs), where the neuron's time-constants are calculated as a learned dynamic function of the current state and input. When training on various tasks, BPTT failed to converge when applied to LTC-SNNs on long sequences due to diverging gradients, while FPTT consistently converged. As we speculated, this suggests that FPTT provides for a more robust learning signal. LTC-SNs maintain binary communication between neurons, but impose additional calculations to determine the neuron state. In neuromorphic implementations, LTC-SNs could be implemented as multi-compartment neurons or require novel spiking neuron model implementations.

The LTC-SN is inspired by multi-compartment modeling of pyramidal neurons in brains. Pyramidal neurons are known to have complex non-linear interactions between different morphological parts far exceeding the simple dynamics of LIF-style neurons [41, 156], where the neuron's apical tuft may calculate a modulating term acting on the computation in the soma [157] that could act similar to the trainable Liquid time-constants used in this work. In a similar vein, learning rules derived from weight-specific traces may relate to synaptic tags [158, 159] and are central to biologically plausible theories of

Table 4.6: Performance of SNNs with Izhikevich and Hodgkin Huxley models on DVS-Gesture dataset. The network structure is same as the network in Table 4.3, where for the Izhikevich the a and b parameters are trainable and we kept c and d fixed, for the Hodgkin-Huxley model all neuron parameters were kept fixed; the network structure and the training-related hyperparameters were not further fine-tuned.

Frames	Izhikevich	Hodgkin Huxley
100	84.03%	87.50%

learning working memory [160]. In general, we find that FPTT, unlike BPTT, can also train networks of complex biologically realistic spiking neuron models, like Izhikevich and Hodgkin-Huxley models (e.g. for the DVS-GESTURE task, Table 4.6). These considerations suggest variations of FPTT may be potential candidates for temporal credit assignment mechanisms in the brain. As a candidate for biologically plausible learning, the spatial error-backpropagation employed in FPTT would need to be replaced with a plausible spatial credit assignment solution, where we anticipate that at least some of the current proposals [161, 162] may be compatible. With such local spatial credit-assignment, FPTT-training of LTC-SNNs can also likely be implemented efficiently on neuromorphic hardware.

FPTT also holds promise for network quantization: FPTT uses both (a form of) synaptic traces in the form of $\bar{W}$ and the actual weights W, where the traces are only used for training. One could imagine networks where the two-parameter sets are each calculated with different quantizations, where W could potentially be computed with lower precision compared to $\bar{W}$. Once trained, only the lower precision weights W are then needed for inference.

Taken together, our work suggests that FPTT is an excellent training paradigm for large-scale SNNs comprised of complex spiking neurons, with implications for both decentralized AI based on local neuromorphic computing and investigations of biologically plausible neural processing.

Acknowledgements BY is supported by the NWO-TTW Programme "Efficient Deep Learning" (EDL) P16-25, and SB is supported by the European Union (grant agreement 7202070 "Human Brain Project"). The authors are grateful to Henk Corporaal for reading the manuscript and providing constructive remarks.

Chapter 5

Network Continual Inference on Streaming Data

Significance: **SNNs, as a more complex type of RNNs, are trained to always learn temporal and spatial information in the data stream. Besides the data fragments that the network aims to learn, the time series also contains many noise fragments. The ideal learning process should be to learn as much as possible from the data fragments rather than the noise. Attention mechanisms in the brain wake up the network for learning or inference at specific moments.**

After training, the trained model must be able to accurately perform the given task over a long period of time. Unlike feedforward networks, inference RNNs heavily relies on the history of hidden states. This leads to performance degradation of the network on long sequences, that is, Challenge 4.

In this Chapter, we transformed the continuous classification problem on streaming data into a decision making problem. We uses a temporal intensity of input data as an attention signal to locate data fragments in the sequence. A brain-inspired decision circuit is also employed to collect evidence and decide when to reset the network. With the help of the above methods, the SNNs can maintain their performance for a long time running. This chapter is based on publication "Attentive Decision-making and Dynamic Resetting of Continual Running SRNNs for End-to-End Streaming Keyword Spotting" [163].

Abstract: *Efficient end-to-end processing of continuous and streaming signals is one of the key challenges for AI in particular for Edge applications that are energy-constrained. Spiking neural networks are explored to achieve efficient edge Artificial Intelligence (AI), employing low-latency, sparse processing, and small network size resulting in low-energy operation. Spiking Recurrent Neural Networks (SRNNs) achieve good performance on sample data at excellent network size and energy. When applied to continual streaming data, like a series of concatenated keyword samples, SRNNs, like traditional RNNs, recognize successive information increasingly poorly as the network dynamics become saturated. SRNNs process concatenated streams of data in three steps: i) Relevant signals have to be localized. ii) Evidence then needs to be integrated to classify the signal, and finally, iii) the neural dynamics must be combined with network state resetting events to remedy network saturation.*

Here we show how a streaming form of attention can aid SRNNs in localizing events in a continuous stream of signals, where a brain-inspired decision-making circuit then integrates evidence to determine the correct classification. This decision then leads to a delayed network reset, remedying network state saturation. We demonstrate the effectiveness of this approach on streams of concatenated keywords, reporting high accuracy combined with low average network activity as the attention signal effectively gates network activity in the absence of signals. We also show that the dynamic normalization effected by the attention mechanism enables a degree of environmental transfer learning, where the same keywords obtained in different circumstances are still correctly classified. The principles presented here also carry over to similar applications of classical RNNs and thus may be of general interest for continual running applications.

5.1 Introduction

Many observational tasks are inherently of an intermittent and continuous nature: while one has to continuously observe surroundings for dangers, the proverbial tiger is fortunately present most sparingly. In a more applied context, keyword spotting requires a similar continuous, or *streaming*, environmental monitoring with relevant stimuli appearing relatively rarely. In each case, a proper balance has to be found between the false alarm rate (seeing a tiger where there is none) and the false reject rate (overlooking the tiger).

Continuous online processing of streaming information is a particular chal-

lenge in energy-constrained situations such as applications running on battery-operated devices. Event-based neural networks like spiking neural network are explored as a means to achieve both low-latency and sparse neural processing, and Spiking Recurrent Neural Networks (SRNNs) in particular achieve good performance on sample data at excellent network size and energy. When continually applied on streaming data however, for example a series of concatenated keyword samples with or without extended pauses, SRNNs, like traditional RNNs, recognize successive information increasingly poorly as the network dynamics become saturated [69]. For RNNs, including modern transformer-based variants like the Conformer [164], solutions have been sought in periodically resetting the internal state of the network, where resetting is typically done using empirical measures tuned for the task at hand [69].

Here, we take inspiration from biology to dynamically reset compact SRNNs to process concatenated continuous streams in continually. For this, we introduce an efficient form of self-attention to localize relevant signals, which also gates information to be integrated into the decision-making circuit to obtain a classification of the detected event. The actual classification is then used as a trigger for resetting the SRNN network state.

We show that compact spiking recurrent neural networks trained on single samples integrated into such extended circuitry can then successfully classify sequences of concatenated keywords. Moreover, they can do this without signal buffers or additional post-processing, demonstrating an efficient and compact end-to-end event-based solution. We also show that the dynamic normalization effected by our attention mechanism enables a degree of environmental transfer learning, where the same keywords obtained in different circumstances are still correctly classified.

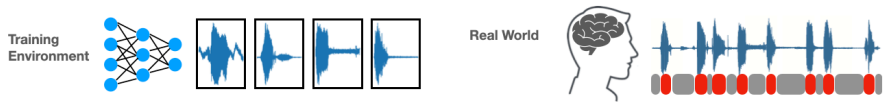


Figure 5.1: The model in the experiment learns on well-segmented samples, while in practice, the model runs continually as decision-making process; note the pauses in the speech signal where no utterance is present

5.2 Related Work

Current approaches to continuous and streaming keyword spotting include three independent steps [165]. First, a stream is typically chunked into segments, for example, using Voice-Activation-Detection algorithms [69]. Second, each segment is processed using a set of overlapping fixed windows on the signal into a set of feature maps. Third, the concatenated features maps are processed to determine a classification label. Single windows can be processed into feature maps with learned approaches, such as Convolutional Neural Networks (CNNs), trained on single labeled samples. CNN's outputs are then converted into a sequence of labels using, for example, recurrent neural networks (RNNs) trained on the Connectionist Temporal Classification loss (CTC) [166]. These RNNs can also be replaced by modern transformers [164, 167] which improve performance but are still applied to segmented utterances. In each of these examples, a post-processing stage transduces observed signal sequences into a labeled sequence.

In [168], a spiking neural network (SNN) version of a CNN is applied to single keyword samples, demonstrating competitive performance with the equivalent non-spiking CNN. The WaveSense model [68] is derived from the classical WaveNet network [169] and is shown to effectively process single samples from various benchmarks up to 5s in length.

Still, these approaches rely on pre-processing to obtain segments and buffering to map sequences into labels. For energy-constrained continuous-monitoring applications, there is a need for continual running end-to-end SNN solutions that minimize pre- and post-processing and have minimal memory and processing requirements.

5.3 Methods

We are specifically interested in continual online keyword recognition and localization in recurrent spiking neural networks, where the network omits buffering and only has access to the current information. To achieve this, we turn to a form of "attention" to help guide the recurrent spiking neural network in localizing and classifying utterances compatible with continual running.

5.3.1 Attentive Spiking Recurrent Neural Networks

Attention has been at the center of current Transformer models, where initially attention was introduced to learn long-range dependencies on image classification and Natural Language Processing (NLP) tasks [170]. In the context of limited or no buffering, attention in recurrent spiking neural networks only allows forms of local and causal self-attention without relying on long-term temporal dependencies. We observe that a straightforward measure of current signal-variability (Temporal Intensity) resembles a temporally local attention-like signal with the potential for localizing speech patterns in a sequence.

We define a specific version of Temporal Intensity based on the Mel-frequency spectrum representation of the signal, which is also the input to the network. We define a temporal average over a single time-step as $\mu_t = |x_t + x_{t-1}|/2$ and associated signal variability as $\sigma_t = |x_t - x_{t-1}|$ based on current input x_t. The real-time Temporal Intensity $tvar_t$ is then derived from μ_t and σ_t and rescaled to [0,1] by the $tanh$ function:

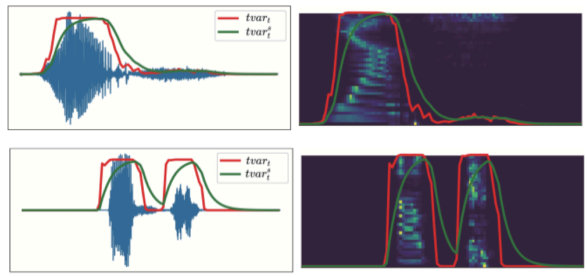


Figure 5.2: Example of speech audio data and corresponding MFCC figure. The red and green curves correspond to the Temporal Intensity and smoothed Temporal Intensity measures.

$$tvar_t = \tanh(\eta \sigma_t \mu_t), \tag{5.1}$$

where η is a hyperparameter we emperically set to $\eta = 4$ in all experiments.

We further define a smoothed Temporal Intensity $tvar_t^s$ as $tvar_t^s = tvar_{t-1}^s + (1-\phi)(tvar_t - tvar_{t-1}^s)$ where $\phi = \exp(-1/\tau_{tvar})$ determines the smoothness. This smoothed Temporal Intensity is used to facilitate the process of evidence accumulation along speech as the the curve of $tvar_t$ tends to be discontinuous (illustrated in Fig 5.2). In contrast to advanced attention models, our $tvar_t^s$ directly localizes speech patterns in ongoing speech sequences, is parameter free, and can be computed in an online manner.

The $tvar_t$ and $tvar_t^s$ measures are illustrated on several keywords speech audio samples and corresponding MFCC representation in Fig. 5.2: in the samples, we see that both measures map closely to the envelope of the signal.

SRNNs. To implement continual running spike-based RNNs, we use adaptive Spiking Recurrent Neural Networks, SRNNs, as developed in [19], comprised of adaptive spiking neurons [19, 171]. Here, the SRNNs are comprised of an input layer converting the input spectrum into spikes. This input layer is densely connected to a single recurrent layer, where the $tvar_t^s$ is also added as an input to the recurrent layer. The final layer is comprised of leaky integrators to generate the prediction probabilities, p_t^i for the ith class at a timestep t. The structure is illustrated in Fig 5.4, described as a structure of 512D-(512+1)R-(12/36+1)I, where the number of output neurons (12 or 36) is task-dependent, and D denotes a dense layer, R the recurrent layer, and I the layer of integrators. The network omits any bias units as they proved detrimental for continual running. As illustrated in Fig 5.4, the SRNN reads the spectrum row by row at each time step, where we call each row a frame; the SRNN thus makes an online prediction at each time step.

We train the parameters of the SRNN using BPTT [19], with some modifications. In the continual running model, the SRNN needs to extract a class-probability label at every timestep. This means that also when learning, the SRNN needs to assign a label to each timestep. For pre-segmented samples however, in many cases only the label for the whole segment is given, and while the actual signal is somewhat centered, it is often flanked by silence or noisy frames. When trained on such pre-segmented samples, the ASRNN ideally only learns from the actual signal and not from the silent or noisy flanks. To achieve this, we introduce an instantaneous Temporal Intensity -gated loss-function between prediction $\hat{y}_t$ and target y for the labeled sample:

$$l_t = loss(y, \hat{y}_t) * tvar_t^s. \tag{5.2}$$

As the Temporal Intensity calculates an envelope of the signal (i.e., Fig. 5.2), this loss helps the network to learn primarily from actual signal data.

5.3.2 Streaming Decision Making

When a network continually generates class predictions for every frame, the challenge is to concatenate this sequence of class predictions into a sequence of predicted labels. Complex methods like the CTC exploit interdependence

between frames or segments combined with implicit sequence modeling to determine the most likely sequence interpretation. However, in an online setting

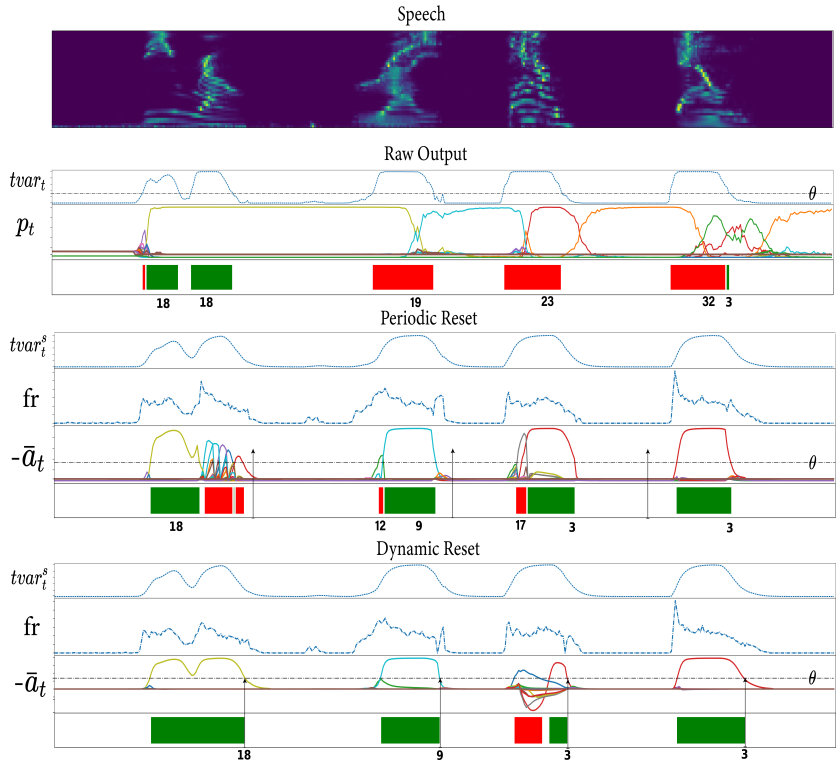


Figure 5.3: An example of the different decision-making process on concatenate speech audio sequence. Top: MFCC spectrum of four concatenated speech utterances. Next is plotted the Temporal Intensity as directly calculated and framewise classification probabilities p_t and resulting classifications (green: correct label, red: incorrect label). The plot below, 'Periodic Reset', demonstrates the effect of smoothing the Temporal Intensity ($tvar_t^s$), the resulting firing rate (fr) and gated action value ($\bar{a}_t$), and resulting framewise classifications given fixed periodic resets. The 'Dynamic Reset' plot illustrates what happens when resets instead are triggered by the decision-making circuit, and additionally winning actions inhibit other actions. Vertical black arrows denote the time of resets.

of concatenated keywords and silence/noise parts, labels are independent and temporally sparse, and the task is more closely related to sequential decision-making. We take inspiration from neural models of decision-making [172,173], and introduce a decision-making circuit with dynamic resetting modeled after the Basal Ganglia brain structure, which is specifically involved in decision-making and context-dependent gating.

The decision-making circuit is shown in Fig. 5.4, in the grey box: it accomplishes action selection by integrating class-probability inputs p_t^i for class i, where actions correspond to labels. An action is selected when a pre-defined threshold θ is reached, and results in the temporary inhibition of other actions. Resetting of the network is triggered when the integrated evidence falls below the threshold again while the Temporal Intensity signal is also rapidly declining at the same time (the effect of this latter condition is that in Fig. 5.3, for dynamic resetting, the third sample is correctly classified even though initially the wrong action/label is selected).

Action selection The activity of the action selection system is modeled as a leaky integrator where a leak time-constant τ_ρ is associated with the typical duration of each action [173]. In the circuit, the action value a_t^i of class i at time t is computed as:

$$\begin{aligned} a_{t+1}^i &= a_t^i + (1-\rho)(u_t^i - a_t^i) \\ \text{where} \quad u_t^i &= -w_z^- I_t^i + w_z^+ \sum_{j \neq i}^{n} I_t^j, \end{aligned} \tag{5.3}$$

and where the input $I_i^t = p_t^i$ and $\rho = \exp(-dt/\tau_\rho)$. As in [172, 173] the balance between disinhibition and inhibition is chosen as $w_z^+ / w_z^- = 1/n$, where n is the number of classes and $\tau_\rho = 20$, chosen to match the average speech length.

To pick up the prediction of the network only on the data, we use a gated action value as a conditional prediction probability or confidence to determine when speech is present. The gated action value is calculated as $\bar{a}_t^i = \tanh(tvar_t^s a_t^i)$ for class i. The frame-wise class label for each timestep is derived from the gated action with minimal value (maximal disinhibited) as well as

$$z_t^k = \begin{cases} 1, & \text{if } k = \underset{i \in 1,2,\dots n}{\arg\min}(\bar{a}_t^i) \wedge \underset{i \in 1,2,\dots n}{\min}(\bar{a}_t^i) < -\theta. \\ 0, & \text{otherwise.} \end{cases} \tag{5.4}$$

where we use a default value $\theta = 0.3$. Note that the same measure can be used as an indicator for speech/no-speech at time t.

Action Inhibition. Once an action (class label) is selected, all other classes are inhibited (where $z_t^j = 0, j \neq i$) when speech is **first** detected at time t. Inhibition is implemented by providing negative inputs to the non-selected action values in the action selection system: an exponentially decaying inhibitory current is added at timestep t' as follows:

$$I_{t'}^k = \begin{cases} p_{t'}^k, & \text{if } z_t^k = 1. \\ -\exp\left(\frac{t'-t}{\tau_\phi}\right) p_{t'}^k, & \text{otherwise.} \end{cases} \tag{5.5}$$

where τ_ϕ controls the leaky speed of the inhibition current for un-selected classes. We empirically set $\tau_\phi = 20$ to match the average speech length.

Network resetting. To counter the state saturation problem associated with continual running in RNNs, network state resets are one solution [69, 164],

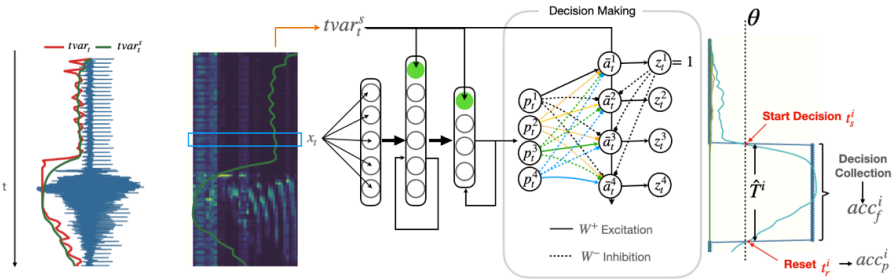


Figure 5.4: End-to-end decision-making procedure. The Temporal Intensity measures and MFCC spectrum are continually computed in an online manner. For each frame t, the smoothed Temporal Intensity and MFCC spectrum are fed into the SRNN, resulting in class probability outputs p_j^t. These outputs are integrated in the decision-making action value nodes a_j^t and associated gated values $\bar{a}_j^t$. Once a threshold θ is reached, the most activated action is selected and the other action values are suppressed. Once the action value for the selected action falls below a set threshold again, at the end of the utterance typically, the label is assigned and the network state is reset. The decision duration $\hat{T}^i = t_s^i, t_{s+1}^i, ..., t_r^i$ represents the period between the starting time of decision collection t_s^i and the reset timestep t_r^i. acc_f^i is the framewise accuracy and acc_p^i is the prediction accuracy for the sample i.

where the challenge is to determine when to reset the network state. Here, we reset the network as a function of when a decision is made *and* when the following empirically derived criterion is satisfied:

- $\min(\bar{a}_t^i) < -\theta$
- $tvar_t^s$ is decreasing and $tvar_t^s - tvar_{t-1}^s < 0.1$.

We see the effect of this condition in the 'Dynamic Resetting' bottom row in Fig. 5.4, where the initially incorrect framewise classification in the third sample does not result in a reset, and the sample is correctly classified according to the acc_p measure. Resetting is only applied in the continual running inference phase and not during training.

Metrics. For network accuracy, we measure two metrics: the framewise accuracy acc_f and the prediction accuracy acc_p.

The framewise accuracy for a single sample i is computed as the average accuracy during the network decision process as well:

$$acc_f^i = \frac{1}{t_r^i - t_s^i} \sum_{t=t_s^i}^{t_r^i} \delta(\hat{y}_t^i, y^i), \tag{5.6}$$

where $\hat{y}_t^i$ is the prediction at timestep $t \in \hat{T}^i$, y^i the correct label for the sample i, and $\delta(\hat{y}_t^i, y^i)$ the Kronecker delta. The average framewise accuracy acc_f is computed as the average over all samples, $\frac{1}{N}\sum_i^N acc_f^i$.

For a sample i, the prediction accuracy acc_p^i is calculated as:

$$acc_p^i = \delta(\hat{y}_{t_r}^i, y^i) \quad \text{iff} \quad |\hat{T}^i| > 10, \tag{5.7}$$

where $\hat{T}^i$ represents the evidence collection duration calculated as the difference between starting time t_s^i and reset time step t_r^i in sample i, as in Fig. 5.4. The average prediction accuracy over N samples acc_p is calculated as $acc_p = \frac{1}{N}\sum_i^N acc_p^i$.

Summary In Algorithm 1, we illustrate the detail of the decision-making procedure, including network initialization, network prediction computation, action value calculation based on inhibitory input, dynamic resetting, and metric evaluation.

Algorithm 1: Dynamic resetting

Variables:
- is_rest – network has been reset(1) or not(0)
- h_t – networks' hidden states
- acc_f – framewise accuracy
- acc_p – prediction accuracy
- t_s – start point
- t_r – reset time

Initialization:
- $h_0 = reset(SNN)$
- is_rest=1, $acc_f = acc_p = 0$

for $t = 0$ **to** T **do**
- // **tvar information:**
  - $\mu_t = \|x_t + x_{t-1}\|/2$, $\sigma_t = \|x_t - x_{t-1}\|$
  - $tvar_t = \tanh(\eta\sigma_t\mu_t)$
  - $tvar_t^s = tvar_{t-1}^s + (1-\phi)(tvar_t - tvar_{t-1}^s)$
  - $dtvar_t = tvar_t^s - tvar_{t+1}^s$ – derivative of smoothed tvar
- //**Network prediction:**
  - $p_t = SNN(x_t, tvar_t^s, h_t)$
- //Action selection:
- **for** $i = 1$ **to** n **do**
  - // **Inhibition current :** if detected class is k at t_0
  - **if** *is_reset ==0 and* $i = k$ **then**
    - $I_t^i = p_t^i$
  - **end**
  - **if** *is_reset ==0 and* $i \neq k$ **then**
    - $I_t^i = -\exp\left(\frac{t-t_0}{\tau_\phi}\right) p_t^i$
  - **end**
  - // **Action value:**
  - $u_t^i = -w_z^- I_t^i + w_z^+ \sum_{j\neq i}^n I_t^j$
  - $a_{t+1}^i = a_t^i + (1-\rho)(u_t^i - a_t^i)$
  - $\bar{a}_t^i = \tanh(tvar_t^s a_t^i)$
- **end**
- Prediction: $z_t = \arg\min_{i\in 1,2,\ldots n}(\bar{a}_t^i)$
- // **Decision making:**
- **if** $\min(\bar{a}_t) < -\theta$ *and* $dtvar_t > 0$ *and is_rest ==1* **then**
  - // Start decision collection, detected class k at t_0
  - // rest statue
  - is_reset = 0 ; t_s = t
- **end**
- // **Framewise Accuracy:** compare current prediction z_t and target label y
- **if** *is_reset ==0* **then**
  - $acc_f += (z_t == y_t)$
- **end**
- **if** $\min(\bar{a}_t) < -\theta$ *and* $dtvar_t < 0$ *and is_rest ==0* **then**
  - // End decision collection and reset hidden states
  - $h_t = reset(SNN)$
  - **if** $(\hat{T} = t - t_s) > 10$ **then**
    - // **Prediction Accuracy:** compare final prediction z_t and target label y
    - $acc_p += (z_t == y_t)$
  - **end**
  - // rest statue
  - is_reset = 1
- **end**

end

5.4 Experiments

We trained our SRNN on both single training samples from the Google speech dataset v1 (GSCv1) or v2 (GSCv2) [94].

5.4.1 Datasets

We trained a conventional GRU network with an equal number of parameters augmented with the same Temporal Intensity -gating and decision-making structures to provide baseline performance – the GRU network was made up of two densely connected GRU layers with 256 units each. We evaluated these networks by training them to classify all 10 keywords in the GSCv1 and 35 keywords in the GSV2 dataset. Each dataset also contains an additional class for "unknown", and GSV1 also contains a "silence" class. In GSCv1, there are 22,236 training samples and 3,081 test samples, GSCv2 dataset comprises 36,923 training samples and 11,005 test samples. The raw audio is pre-processed via MFCC bandpass filters: each audio sample is passed through 40 2nd order bandpass filters distributed along the Mel-scale between 20Hz and 4kHz. We rescale the response of 40 bandpass filters at each timestep by dividing by the standard deviation across the spectrum. Each keyword sample is converted to a sequence of 101 timesteps in a 40-by-3 matrix representing the spectrum at each time step.

Direct, single sample performance is measured on the respective test datasets. We evaluate the continual running of both the SRNN and GRU on long sequences comprised of concatenations of keywords. To evaluate the network performance on single keyword prediction, we define prediction accuracy acc_p by comparing the prediction and the target when the speech pattern disappears and the network is reset (see also Fig. 5.4). We also measure the network performance on long sequences by comparing average frame-wise accuracy acc_f as measured over the whole sequence length. To evaluate the networks' robustness to noise, we applied background noise to each speech audio. Different levels of synthetic noise were applied on the first 10 filter bandpass filters. The noise was generated as Gaussian, $r\mathcal{N}(0,1)$ where r is the noise ratio.

5.4.2 Results

Performance on single samples. We evaluated the networks on single speech audio samples. For this, we evaluated the GRU network including the Temporal Intensity gating and the decision-making circuit. Then, we compared it to

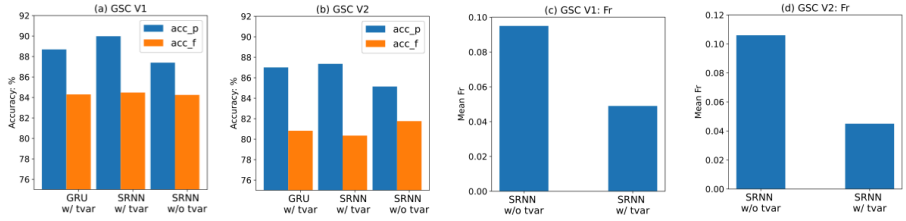


Figure 5.6: Single sample performance. a) Classification accuracy acc_p and average framewise accuracy acc_f for various baseline networks for GSCv1, and b) for GSCv2. c) average network activity (spike probability per timestep) for GSCv1 and d) GSCv2.

SRNN networks with or without Temporal Intensity gating. Results are shown in Fig. 5.6: we find that the SRNN with Temporal Intensity gating slightly outperforms the other networks in terms of classification accuracy, including the GRU network, for GSCv1 (Fig. 5.6a) and GSCv2 (Fig. 5.6b). Compared to the literature, in [19] ASRNNs achieve 92.14% on GSCv1, slightly better than our dynamic SRNN (89.98%), while the GSCv2 accuracy (87.31%) represents new State-of-The-art (SoTa), exceeding the 79.6% reported in [174].

Noting average activity in the network (Fig. 5.6c,d), we see that using Temporal Intensity gating lowers the required number of spikes by some 50% for both GSCv1 and GSCv2 tasks. We also find that 68% of spikes in the networks are on average generated during the "active" parts where Temporal Intensity exceeds the signal threshold θ.

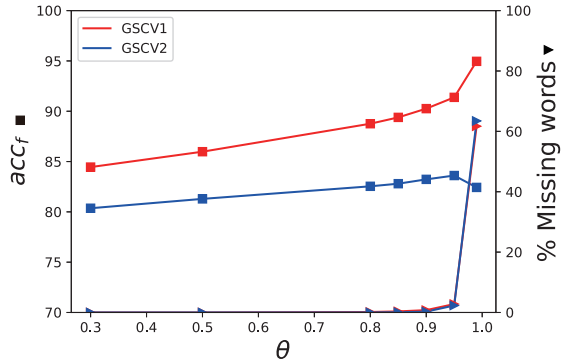


Figure 5.5: Effect of threshold θ on frame-wise accuracy and % of missing words. The calculation of framewise accuracy only accounts upon detected words.

Effect of Threshold θ. The parameter θ distinguishes between noise/quiet and speech patterns. Smaller θ will result in noise being more likely treated as part of the speech pattern, while larger θ will cause

Dataset	Model	T=1	T=2	T=4	T=8	T=16	T=32	T=64	T=128
GSC V1	GRU raw output	84.31	60.96	47.72	41.37	40.67	37.86	27.98	35.10
	GRU Dynamic Reset	84.31	84.45	83.28	83.74	84.02	82.84	83.07	82.86
	SRNN raw output	83.85	59.54	40.10	27.10	21.28	16.82	15.42	15.03
	SRNN periodicc reset	83.85	83.31	83.07	83.05	82.95	82.98	82.92	82.84
	SRNN action selection with dynamic resetting	84.10	84.09	83.34	83.09	83.03	82.77	82.99	82.88
GSC V2	GRU raw output	80.82	55.25	44.46	39.95	37.59	35.73	35.30	33.02
	GRU Dynamic Reset	80.82	79.58	80.11	79.46	79.77	79.85	79.94	79.82
	SRNN raw output	79.39	48.53	31.45	21.58	15.62	13.49	12.24	11.70
	SRNN periodic reset	79.42	79.38	79.03	79.02	79.31	78.98	78.93	78.88
	SRNN action selection with dynamic resetting	81.73	80.54	80.36	80.12	79.98	80.39	80.05	80.35

Table 5.1: Average frame-wise accuracy acc_f when min $\tilde{a}_t^i < -\theta$ for different concatenated sequence lengths.

the network to not identify
more words in the recognition process. As such θ directly controls the false positive and false negative rates. In Fig. 5.5, we plot how θ influences keyword detection as measured in terms of average framewise accuracy acc_f. We see that indeed, as θ increases, the number of missed words grows, and accuracy improves.

Continual running: long sequences. The same networks are also evaluated in the continual-running setting, carrying out continuous inference on speech sequences over longer periods of time. In Table 5.1, we note the SRNN and GRU networks' performance on concatenated sequences of commands, ranging from a single keyword to 128 concatenated keywords from either GSCv1 or GSCv2. For easy comparison, we report average frame-wise accuracy acc_f when $tvar_t^s > \theta$ for raw output of the network, networks with periodic resetting, and networks with dynamic resetting.

We make several observations from Table 5.1: first, without resetting both, GRU and SRNN networks saturate, and recognition performance suffers dramatically. Including a periodical reset resolves this issue for the SRNN network (and also for the GRU network, not shown). We then see that our dynamic resetting scheme based on the action selection circuit provides essentially equal (GSCv1) or even slightly better (GSCv2) accuracy.

We also find that with the dynamic resetting mechanism, adding longer silences between concatenated speech samples does not affect the framewise classification accuracy; an example of such added silence is shown in Fig. 5.7.

To further quantify the quality of dynamic resetting, we calculate the editing

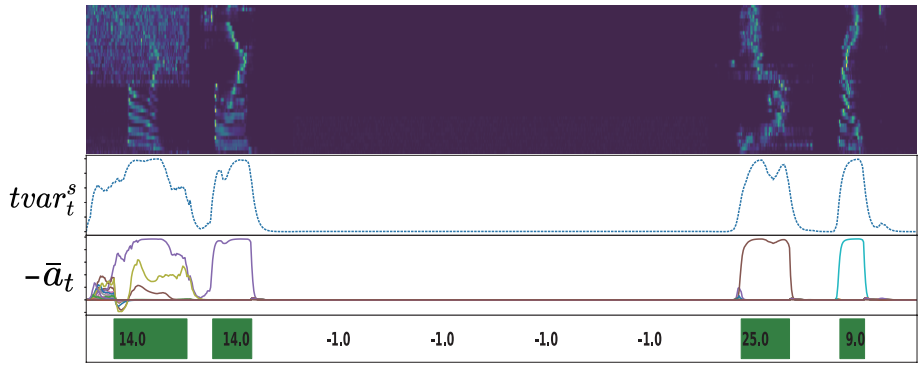


Figure 5.7: Effect of long salience on SRNN. Label "-1" denotes the added silence audio patches.

distance [175], both for the GRU and for the SRNN, on concatenated sequences of samples that are correctly classified when presented as single samples. In Fig. 5.8a, we plot the average number of editing operations needed when evaluating a sequence of 1000 concatenated samples on GSCv2: we find that for both GRU and SRNN, the dynamic reset outperforms the fixed periodic reset (159 vs 158 for GRU and 112 vs 70 for SRNN); we also find that the SRNN network substantially outperforms the GRU network (70 vs 158 operations).

Temporal Intensity compensates distribution shift

While typical speech benchmarks are comprised of clean samples recorded under essentially ideal conditions, new recordings processed under different circumstances may result in a shifted frequency distribution leading to degraded performance. For example, in [94] a standard CNN model was trained on either of the two versions of GSC datasets and then assessed on both datasets. Depending on the type of CNN, performance was more or less degraded when a network trained on one dataset was evaluated on the other.

Here, we optimized RNNs on either GSCv1 and GSCv2, and evaluated their performance on both datasets. As shown in Fig. 5.8, we find that standard GRU networks, not gated by Temporal Intensity, show substantial susceptibility to distributions shifts, as average performance drops by 9% (GSCv1 vs. GSCv2) and 7% (GSCv2 vs. GSCv1). For SRNN networks not gated by Temporal Intensity, we find a similar issue; SRNNs with Temporal Intensity -based attention,

however, prove to not be sensitive to distribution shift and maintain accuracy (GSCv1 vs. GSCv2) or even improve accuracy (GSCv2 vs. GSCv1, due to the larger training dataset).

5.5 Discussion

We demonstrated how the inclusion of a local signal-detection measure combined with brain-inspired decision-making circuitry allows compact and high-performance SRNNs to be applied to continual running scenarios. For signals comprised of concatenated keywords, this results in constant-accuracy continual running. Importantly, Temporal Intensity-gating resulted in much reduced average activity in the SRNNs, potentially improving energy consumption. Measured in terms of editing distance, we find that dynamic resetting results in substantially better accuracy, where the SRNN networks outperform GRU networks. We also showed how the decision-making criteria enable the trading-off of false alarms versus missed keywords. A next step will be to evaluate SRNNs on real-world continual running scenarios, which we omitted for lack of a current suitable public benchmark to use and compare to.

We observed furthermore that the Temporal Intensity-gated SRNNs are in-

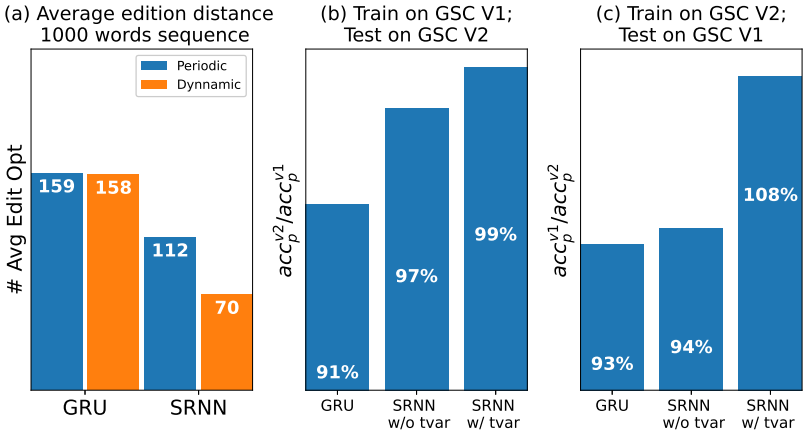


Figure 5.8: (a) Average editing distance and (b,c) distribution shift robustness computed as percentage of accuracy on the original distribution.

sensitive to a distribution shift, as measured in terms of environmental transfer performance from GSCv1 to GSCV2 and vice versa. We find this observation somewhat curious, but as noted, similar observations have been made for CNN architectures where some architectures are more or less susceptible to distribution shift.

The absolute classification performance achieved by the SRNN networks is compelling and approaches or exceeds state-of-the-art for SNNs. Still, we believe that the accuracy of the SRNNs can likely be further improved by, for instance, replacing the MFCC features with custom learned ones [176], and optimizing circuit parameters like reset intensity, decision thresholds, and action-selection triggered via lateral inhibition for class specificity. Furthermore, more complex SRNNs can additionally improve sample recognition rates [19], potentially at the expense of increased computational complexity.

Our results open new possibilities in the design of always-on keyword-spotting devices such as the one presented in [177]. This device exploits switched ring oscillators for generating event-based frequency outputs from audio streams. Today, these outputs are processed in a frame-based way using a GRU network. By replacing the frame-based GRU network with SRNNs that directly process event-based features, it will be possible to compute on-demand on the streamed frequency output features, thus further reducing the overall system's power and latency [19] while increasing accuracy.

Chapter 6

Discussion & Research directions

SNNs are simplified models of biological neural networks that elegantly fill part of the gap between biological neural networks and ANNs. Thus, SNNs, like the human brain, should exhibit highly accurate performance with low energy consumption when applied to complex tasks. My research shows we are now getting closer to constructing such SNNs. Here, I review each of the challenges presented in Chapter 1, discuss the contributions and limitations of our approach to solving each challenge, and examine possible future research directions.

6.1 Challenge 1: Discontinuity of the spike generation function

The spike generation function as an essential component of neuronal dynamics and is nonlinearly non-differentiable. This non-differentiability cannot be bypassed during the gradient-based training of SNNs. A possible solution to this challenge is surrogate gradients, which use a function of membrane voltage as an approximation of binary spikes' gradient information. Compared with other approaches, this approach allows more accurate gradient information to be computed and transmitted in the BP of multilayer SNNs. Surrogate gradients are currently widely used in gradient-based network optimization algorithms and have achieved promising results in various tasks [67,77,134,139,178,179].

In Chapter 2, we introduced a new surrogate gradient, the Multi-Gaussian gradient, inspired by a sigmoid-style saturating activation function, the dSilu. As with dSilu, we also find that the negative part of the gradient helps to improve the accuracy and, in SRNN, the sparsity. We shown this activity-regularizing surrogate gradient with recurrent spiking networks of tunable and adaptive spiking neuron obtains State-Of-The-Art performance for SNNs, and outperforms that of classical recurrent neural network.

Instead of these surrogate gradient-based solutions, we can also implement some gradient-free or robust gradient-based update algorithms to eliminate the impact of the gradient shape on the SNNs' performance.

- Event-based algorithms such as the STDP algorithm [12] and its variants [74, 75] have shown promise for network training. However, these approches face some difficulties in training: they require a very long simulation time for training, and the precision of the gradients provided by their updating mechanism is insufficient to train deep SNNs. However, such algorithms are attractive for on-chip learning. Moreover, we can accomplish more accurate, more biologically plausible, and more hardware-friendly training by combining the advantages of the STDP and FPTT algorithms into approaches such as global–local learning [180].

- We can also use simplified gradients on network optimization algorithms in the BP framework. Training with simplified gradients such as the directional gradient [181] and random gradient [182, 183] can be described for SNNs in future studies.

- We can perform derivative-free network optimization for SNN training by applying algorithms such as the parametric search algorithm [184, 185], the particle swarm optimization algorithm [186], or the evolutionary learning algorithm [187, 188]. However, these algorithms consume more computational resources than other algorithms in optimising large-scale neural networks, and achieving the desired model accuracy is difficult.

The spike generation function gives neurons an efficient way to propagate information between layers, but it also brings some errors in the gradient calculation. We can use the surrogate gradient method to solve this problem reasonably directly, but the approximation error continues accumulating if the BPTT algorithm is applied. We can also use derivative-free algorithms and event-based STDP algorithms to bypass gradient computation on spikes in SNN op-

timization. Alternatively, we can optimise an existing gradient computation circuit to reduce the effect of surrogate gradient errors on network training.

6.2 Challenge 2: Cumbersome gradient flow

Compared with RNNs, the complex neuron model in SNNs endows neurons with more advanced memory capabilities but also leads to more complex gradient computation paths in BP in the time dimension.

6.2.1 Offline learning

We describe the training of SNNs with offline learning updates in Chapters 2 and 3, which results in excellent network performance on various tasks. However, the constraint of unfolding the chain rule over the time dimension is invariably present in the BPTT algorithm. This exponentially increases the risk of gradient explosion in network training if the sequences become long or the models become deep. Although this can be addressed by truncating, simplifying, or sparsifying the gradient computation process to reduce its effect and accelerate training, these methods do not fundamentally solve the problems encountered when training SNNs in an offline learning framework. As we demonstrated in Chapter 3, the Truncated BPTT, which always reduces the training time and improves the sparsity of the model .

In the BPTT algorithm framework, the averaged weight changes over all time-steps of samples in a batch are applied for weight updating. However, this BP uses a large amount of computational resources for the optimization of SNNs. In contrast, the sparse spike gradient approach [189] uses a light updating method, which only renews the synaptic strength of the connections with a spike. However, this approach can only complete learning when spikes flow into a network, so the stability of its algorithm needs further investigation.

6.2.2 Online learning

The network optimization in our studies has usually been performed in the BPTT algorithm framework. In this framework, a network is updated based on the weighted average of all examples at all moments. This approach is biologically implausible but effectively optimizes a network with the support of a vast amount of data. In addition, we have applied an online updating algorithm – the forward propagation through time (FPTT) algorithm – to train SNNs. Unlike

the BPTT algorithm, the FPTT algorithm updates weights based on the neuronal membrane potential values at each moment. My work suggests that FPTT is an excellent training paradigm for large-scale SNNs comprised of complex spiking neurons. FPTT intuitively provides a more robust and efficient gradient approximation for spiking recurrent neural networks than BPTT, as FPTT simplifies the complex gradient computation path in SNNs. This potentially reduces the risk of gradient vanishing or explosion during the training, resulting in the ability to train on longer sequences.

FPTT enables the neurons in a network to renew their connection weights without delivering any signal (spike) to the next neuron. This is different from the updating mechanism of biological neurons, which only adjusts synaptic strength when a spike is emitted [12].

Online algorithms can greatly simplify the gradient calculation path and thus serve as a more biologically plausible solution than the off-line methods for effectively solving these problems. Existing online learning algorithms such as the e-propagation algorithm [77], the STDP algorithm [75], and the DELLC algorithm [155] are excellent for training networks on long sequences but are challenging to apply to large network models. However, the FPTT algorithm breaks the chain of the gradient computation in the time dimension, such that the complex computational loop of neurons is no longer an obstacle to training. Accordingly, with the help of the FPTT algorithm, we showed that we can train more complex neurons and larger network models than was previously possible.

However, the FPTT algorithm could be improved in several ways, such as those described below.

- In online learning (the FPTT algorithm), we could use a spike-triggered weight update similar to [189] by defining

$$\frac{\partial s_t}{\partial u_t} = \hat{f}'(u_t) \quad \rightarrow \quad \hat{f}'(u_t) s_t.$$

  In this way, we can create a sparse BP path that only updates the weights of active synapses, such that neuron spikes dynamically truncate the computational graph. This can also reduce the risk of gradient vanishing and explosion in network training.

- The FPTT algorithm requires an instantaneous loss for gradient calculation and parameter updates at each moment. In reality, however, the instantaneous desired output is not accessible. Theoretically, a network should be updated only when the gradient information is accessible. The

gradient information is thus derived from comparing the network output and the desired output in the BP framework. In seq2seq [190] or control tasks [52], immediate losses are available to the network, and so the FPTT algorithm can be applied directly to network training. However, this is not be suitable for tasks that provide labels only at the end of the sequence, as FPTT learning in supervised learning requires labels at each moment. An effective solution to this problem is for a network to perform self-supervised or unsupervised learning when no target information is available and then run supervised learning to assign the cluster or representation when the labels are accessible.

- Next, a network is better trained with more precise labels than with general labels in supervised learning tasks. In Chapter 4, the same target labels are given for each moment in the input sequence. This process inevitably marks some noise fragments as data to be learned. Thus, we can apply the intensity metric proposed in Chapter 5 to determine how much the network will learn from each moment to obtain more accurate labels for learning.
- In addition, the parameter updating process performed at each moment requires a spatial gradient BP calculation, so the FPTT algorithm is local in time but not in space. In this case, feedback alignment algorithms, such as random feedback weights [182], DFA [191], and LocoProp [192], can be used to help the FPTT algorithm become a truly local online network learning algorithm. Similarly, the synthetic gradient [193] approach and its variants [194, 195] can be used to break the chain of spatial gradient computation. Alternatively, a BP network can be defined to generate gradient information for each layer.

6.3 Challenge 3: Memory

SNNs are a type of RNN with more complex loops, which means that SNNs have inherently larger models than traditional RNNs. This is reflected by the memory consumption of SNNs increasing linearly as the length of a sequence increases and as a model becomes larger and deeper. In an offline training framework, the memory costs of SNNs are linearly greater than those of RNNs. However, by using online learning algorithms, the effect of sequence length on SNN memory consumption can be eliminated. Thus, after breaking the memory limit with the FPTT algorithm, we can train SNNs as feedforward networks

on sequences of arbitrary length with a constant memory consumption, which provides computational possibilities for future optimizations of large SNNs. The increased memory efficiency of FPTT allows for training much larger SNNs as was previously feasible, as we demonstrated in the SPYv4 network for object detection (Chapter 4). In the experiment, we note that the FPTT implementation is unoptimized in the used version of PyTorch, where for instance the memory allocated to historical hidden states is not de-allocated for FPTT, resulting in unnecessary large memory use, and low-level optimized FPTT implementations should further reduce memory to near constant.

6.4 Challenge 4: Continual running

A trained model eventually runs continuously on a data stream, rather than for a fixed period. Feedforward networks can be applied directly to a data stream without considering performance degradation. However, the performance of RNNs is heavily dependent on their hidden states, which results in performance degradation when a network is run continuously for an extended period.

We have found that resetting a network can help to alleviate the performance degradation of RNNs during continuous inference. However, it is crucial to determine when to reset. To solve this problem, in Chapter 5, we use the temporal intensity of the input data as a real-time attention signal to assist the network in locating the data features. Inspired by Basal Ganglia structure in the brain, a decision circuit is also applied to collect evidence, which is used to decide when to make a final decision and when to restart the network. This approach can significantly reduce the performance decay of SNNs during continuous inference.

Nevertheless, this solution could be improved in the following ways.

- In Chapter 5, we use a data feature-based attention approach to locate data features, which are then used to make decisions. However, attention can be categorised into two directions based on function: bottom-up attention, which is attention to external stimuli, especially those that contrast with the background and are objective; and top-down attention, which is internal attentional guidance based on a priori knowledge, plans and goals. Here, we only apply bottom-up attention. Similarly, we can design top-down attention to dynamically decide when to activate a sub-network and when to make decisions based on the state of deep layers' activities. In future, we can use this more complex attention loop to solve

this challenge. Alternatively, we can allow a network to wake up layer-by-layer, based on the input data features, to complete recognition.

- Resetting the network in a simulation is easy, but repeated resets during a hardware deployment consume a large amount of energy. Thus, we can instead suppress or clear the state of a network by entering a negative input to the network to perform the reset function.
- The decision-making process is not only involved in the continuous inference process on serial inputs, but there is also the problem of deciding when to learn in continuous learning. Thus, the decision process of a network should contain multiple stages, such as deciding when to wake up the network, deciding which part of the network to wake up, deciding when to make a prediction, deciding when to learn new knowledge, and deciding when to restart the network. The bottom-up and top-down attention mechanisms play different functions in these decision processes, and this idea has been applied in recent sparse models. The dynamic routing mechanism based on bottom-up and top-down attention will help large models to accomplish multiple tasks by using the part of the sub-network that is awakened for processing specific data.

This completes the review of the four significant challenges encountered in the training of SNNs, as presented in Chapter 1.

6.5 Outlook

Our contributions are significant for subsequent research on both training and deployment of large-scale SNN models.

We showed how multi-layered recurrent network structures, trained with BPTT and its variants, are able to achieve new state-of-the-art performance for SNNs on sequential and temporal tasks. When expressed in terms of computational operations, they demonstrate a decisive theoretical energy advantage of one to three orders of magnitude over conventional RNNs. This advantage furthermore increases for more complex tasks that required larger networks to solve accurately. To demonstrate the performance of large-scale models, we introduce FPTT into SNN training. Our work suggests that FPTT is an excellent online training paradigm for large-scale SNNs comprised of complex spiking neurons, with implications for both neuromorphic computing and investigations of biologically plausible neural processing.

At the same time, our results open new possibilities in the design of always-on keyword-spotting devices such as the one presented in [177]. The principles presented here also carry over to similar applications of classical RNNs and thus may be of general interest for continual running application.

On the basis of our work, I believe that SNNs will be able to obtain competitive accuracy with ANNs on large-scale networks soon, and can continuously operate on edge devices with high accuracy and much lower power consumption than ANNs.

List of Publications

The following articles were published in the process of this research:

- Chapter 2 corresponds to the article, "*Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks*", by **Bojian Yin**, Federico Corradi and Sander M. Bohté, published in *Nature Machine Intelligence* 3, 905-913 2021. An early work has published in *International Conference on Neuromorphic Systems 2020 (ICONS2020)*, as "*Effective and efficient computation with multiple-timescale spiking recurrent neural networks*", by **Bojian Yin**, Federico Corradi and Sander M. Bohté.
- Chapter 3 contains results published as "*Real-time classification of LIDAR data using discrete-time Recurrent Spiking Neural Networks*", by Anca-Diana Vicol, **Bojian Yin** and Bohte, S.M. in *International Joint Conference on Neural Networks, 2022 (IJCNN2022)*.
- Chapter 4 includes material form the paper "*Accurate online training of dynamical spiking neural networks through Forward Propagation Through Time*", by **Bojian Yin**, Federico Corradi, Sander M. Bohté. This paper was accepted and will be published on *Nature Machine Intelligence*.
- Chapter 5 presents the results of "*Attentive Decision-making and Dynamic Resetting of Continual Running SRNNs for End-to-End Streaming Keyword Spotting*" by **Bojian Yin**, Qinghai Guo, Henk Corporaal, Federico Corradi and Sander M. Bohté, published in *International Conference on Neuromorphic Systems 2022 (ICONS2022)*.

Following papers are not included in this thesis:

- **Bojian Yin**, H Steven Scholte, Sander M. Bohté. "*LocalNorm: Robust Image Classification Through Dynamically Regularized Normalization*". Published in *International Conference on Artificial Neural Networks 2021 (ICANN2021)*.

- **Bojian Yin**, Marleen Balvert, Rick AA van der Spek, Bas E Dutilh, Sander Bohté, Jan Veldink, Alexander Schonhuth. "*Using the structure of genome data in the design of deep neural networks for predicting amyotrophic lateral sclerosis from genotype*". Published in I*ntelligent Systems for Molecular Biology and European Conference on Computational Biology 2019 (Bioinformatics)*.

Summary

Efficient and Accurate Spiking Neural Networks

Deep learning techniques have greatly improved advances in object detection, speech recognition, robotics, and many other areas. However, the success of deep learning is achieved by deep neural networks trained with extensive training examples and massive computational resources. In contrast, the brain allows us to perform complex tasks and learn abstract concepts with ultra-low energy consumption and a limited number of samples. Inspired by detailed modelling of biological neurons, spiking neural networks (SNNs) have been studied as biologically plausible models for high-performance neural computation. Sparse and binary communication between spiking neurons has the potential to enable powerful and energy-efficient neural networks. Current learning algorithms of SNNs are limited to small networks of simple spiking neurons and modest-length temporal sequences. Firstly, the performance of SNNs, however, has remained lacking compared to artificial neural networks (ANNs). Secondly, offline methods impose high memory requirements, have difficulty training large-scale and complex neuron models, and are incompatible with online learning.

This thesis contributes to the effective training of large-scale, energy-efficient, and accurate spiking neural networks that are applicable to complex tasks. It presents four significant challenges for the training of SNNs and proposes corresponding solutions in each chapter.

We first introduced how an activity-regularizing surrogate gradient incorporated with recurrent networks of tunable time constant and adaptive spiking neurons, ASRNN, yields state-of-art for SNNs on challenging benchmarks. We also demonstrate that SNNs trained by offline algorithms and their variants

are theoretically one to three orders of magnitude more computationally efficient compared to advanced RNNs with competitive performance. Second, we train our ASRNN on an open LIDAR labeled dataset and propose specific optimizations, including various input encoding, sparse connectivity and truncated BPTT. Next, we show how recently developed "Forward Propagation Through Time" (FPTT) combined with novel Liquid Time-Constant spiking neuron provides a more accurate solution for online training large-scale SNNs with more complex neuronal models on longer sequences. We illustrate online learning of long sequences while outperforming current online methods and approaching classical offline methods on temporal classification tasks. The efficiency and robustness of the new online algorithm enable us to train deeper networks and more complex neurons. As a result, we demonstrate for the first time the possibility of directly training large-scale spiking neural networks (SPYv4) for object detection. We find that FPTT can also train the network of more biologically-detailed models like Izhikevich and Hodgkin-Huxley models. Finally, we present the combination of temporal attention and brain-inspired decision circuits that enable precisely continual inference in the stream of signals. The principles we proposed in this work also extend to similar RNN applications and may be of general significance for applications in continual running.

The significance of this study is that it improves SNNs performance by introducing a new training paradigm for multi-scale SNNs, and provides performance guarantees for later deployment on neuromorphic hardware.

Acknowledgments

In the snap of a finger, after four years of a PhD, I finally got here by working all the way. I have met many people want to thank along the way. We have fought with the COVID and have stressed about the papers.

First, I would like to thank Sander M. Bohté for his guidance in research throughout my PhD. Our adjacent offices facilitated the breakout of new inspiration and ideas with high-frequency discussions. Every meeting started with a summary of the recent work and ended with a schedule of subsequent works. These discussions helped me to correct research direction promptly and practice new ideas quickly. A wealth of knowledge, an active mind, and keen insight from Sander enabled me to clearly understand the research problem, clarify the research direction, and broaden research ideas. Most importantly, he taught me that for a PhD, I should have a piece of wide-ranging knowledge and, more importantly, a focused perspective on research and a literary taste.

I would also like to thank Federico Corradi for his research ideas from a hardware implementation standpoint. During the discussions with him, I learned more about the demands of industry and the urgent problems that need to be solved. His extensive experience with hardware implementation has dramatically enriched my horizons.

I gratefully knowledge the DEL program, especially imec, who sponsored my PhD and provided the internship opportunities. Thanks to Henk Corporaal and Sander Stuijk for helping me to complete PhD project. Of course, thanks to all my colleagues at CWI and members in Machine Learning group for tolerating my lack of communication when working. I also thank CWI for providing me with comfortable research and office environment.

My sincere thanks to Giacomo Indiveri, Elisabetta Chicca, Friedemann Zenke, Guido De Croon, Terrence C. Stewart, Sebastian Otte, Emre Neftci, Manolis Sifalakis, Jesse Hagenaars, Sherif Eissa, Qinghai Guo and Yansong Chua, I have

significantly benefited from the discussions with all above. In addition, there was the Capo Caccia neuromorphic summer school and the like-minded people I met there.

Thank you to my friends, Wen Tao and Xue Jing, for happily enjoying the few social life I have. Last but not least, great thanks to my wife (Chen Xi), my parents and relatives and friends for your unconditional and unlimited love and support.

Bojian Yin
Amsterdam, Oct 2022

Appendix A: Multi-timescales Spiking Recurrent Neural Networks

task		ECG	SHD	SSC	GSC	SMNIST	PSMNIST	SoLi	TIMIT
seq length		1301	250	250	101	784	784	40	500
time constant	$\tau_m 1$	(20;.5)	(20;5)	(20;5)	(20;5)	(20;5)	(20;5)	(20;5)	(20;5)
	$\tau_{adp} 1$	(7;.2)	(150;10)	(150;50)	(150;50)	(200;50)	(200;50)	(20;5)	(200;5)
	$\tau_m 2$	(20;.5)	(20;5)	(20;5)	(20;5)	(20;5)	(20;5)	(20;5)	(20;5)
	$\tau_{adp} 2$	(100;1)	(150;10)	(150;50)	(150;50)	(200;50)	(200;50)	(20;5)	(200;50)
	$\tau_m 3$	-	(20;5)	(20;5)	(20;5)	(20;5)	(20;5)	(20;5)	(3;1)
	$\tau_{adp} 3$	-	-	-	-	(200;50)	(200;50)	-	-
learning rate		1e-2	1e-2	1e-2	1e-2	1e-2	1e-2	2e-2	1e-2
Loss		NLL	CE	CE	CE	CE	CE	NLL	CE
minibatch size		64	64	32	64	256	256	128	16
epochs		400	20	150	70	300	300	150	200
lr decay		.5per50	.5per20	.5per10	.75per20			.75per50	.5per100
lr decay type		Step	Step	Step	Step	Linear	Linear	Step	Step

Appendix A Table A1: **Details of parameters used for initialization and optimization for each task.** Time constants refer to successive layers in the architecture; the layer output is last numbered layer – the numbers between brackets denote the mean and std used for initializing the time constants with Gaussian distribution $N(\mu;\sigma)$ where the μ is the mean of the time constant and σ is std. LIF non-spiking output layers use only the τ_m parameter. For the Loss, NLL denotes negative log-likehood and CE cross-entropy. The learning rate decay schedules denote the amount of decay (multiplicative factor) and the epoch when the decay is applied (in case of Step decays); for SMNIST and PSMNIST, a linear decay to zero is applied.

Task	Method	Accuracy	Energy$_{nn/s}$	Energy ratio	Error ratio	Efficiency
ECG-qtdb	Adaptive SRNN	***85.9%***	325.7	1x	1x	1x
	LIF SRNN	75.5%	179.9	.55 x	1.67x	.91x
	RELU SRNN	86.4%	5784.6	17.8x	.93x	16.5x
	LSTM	78.9%	20422.8	62.7x	1.43x	89.6x
	GRU	77.3%	15400	47.2x	1.54x	72.7x
	Vanilla RNN	74.8%	9597.6	29.5x	1.71x	50.5x
	Bidirectional LSTM$_{290}$	80.76%	563580	1729.9x	1.31x	2266.2x
SMNIST	Adaptive SRNN	***98.7%***	8250.3	1x	1x	1x
	RELU SRNN	98.99%	487623.8	59.1x	.74x	43.4x
PSMNIST	Adaptive SRNN	***94.32%***	7775.1	1x	1x	1x
	RELU SRNN	93.47%	487623.8	62.7x	1.1x	69.0x
SHD	Adaptive SRNN	***87.81%***	3515.7	1x	1x	1x
	RELU SRNN	88.93%	442097.2	125.8x	.91x	114.5x
	Bidirectional LSTM	87.2%	3468215.52	986.5x	1.05x	1035.8x
	CNN [92]	92.4%	–	–x	–x	–x
SSC	Adaptive SRNN	74.18%	10154.1	1x	1x	1x
	RELU SRNN	74.36%	2373918.	234.0x	.99x	231.7x
	LSTM [92]	73.1%	–	–x	–x	–x
	CNN [92]	77%	–	–x	–x	–x
SoLi	Adaptive SRNN	79.8%	13,804	1x	1x	1x
	Vanilla RNN	63.6%	4101324.8	297.1x	1.80x	524.8x
	GRU	79.20%	6580531.2	476.7x	1.03x	491.1x
	LSTM	79.99%	8360972.8	605.7x	0.99x	599.6x
	RELU SRNN	79.8%	3283950.9	237.9x	1.x	237.9x
TIMIT	Adaptive SRNN	66.13%	10626.8	1x	1x	1x
	RELU SRNN	–%	11861974	175.2x	–x	–x
GSC	Adaptive SRNN	92.12%	4120.3	1x	1x	1x
	RELU SRNN	–%	11861974	167.5x	–x	–x
	CNN [87]	92.4%	80600	19.6x	0.96x	18.8x

Appendix A Table. A2: **Performance and relative energy consumption for various models.** For comparison, the average energy consumption of each input timestep and error rate of Adaptive SRNN was set to unit value. The energy/error ratio is defined as the ratio of the energy/error compared to the adaptive SRNN. We define the efficiency as the product of energy and error ratio. In the ECG task, the Bidirectional LSTM$_{290}$ is a network with 2 bidirectional LSTM with 120 and 40 hidden neurons respectively followed by three dense layer with 100, 20 and 6 hidden neurons. For the ECG tasks, the comparison networks (LSTM, GRU, vanilla RNN and LIF SRNN) share the same architecture with 36 hidden neurons each. In the SHD task, the bidirectional LSTM is a 2-layer bidirectional network with 128 neurons each, followed by a dense layer with 100 neurons. In the SHD and SSC tasks, the temporal bin-width for the LSTM and CNN networks [92] is set to 10ms with a sampling frequency of 100Hz. For SoLi, all networks (vanilla RNN, GRU, and LSTM) use the same network structure as the SRNN, with a recurrent layer with 512 neurons followed by a dense layer with 512 neurons. All CNNs in the table read the whole sequence at once, and their computation cost is computed as the average cost per timestep. Dashes denote where values are not available either because of lacking network architecture details or lack of convergence (ReLu SRNN for TIMIT and GSC).

Appendix B: FPTT theory

For conciseness, we briefly summarize the theory underlying FPTT as developed by Kag et al. [131].

Back-propagation-through-time Back-propagation-through-time (BPTT) uses backpropagation to calculate the gradient of the accumulated loss along the spatial-temporal dimension with respect to the parameters of the recurrent networks. Let us define a recurrent network described by differential equation $(\hat{y}^t, h_t) = NN(x_t, h_{t-1})$ where x_t is the input , $\hat{y}^t$ is the prediction and h_t is the hidden states. The gradient of time t is then computed by considering the effect of the state x_t on all future losses $l^t, l^{t+1}, ...l^T$:

$$\frac{\partial L}{\partial w} = \sum_{t=1}^{T} \frac{\partial l^t}{\partial w} = \sum_{t=1}^{T} \sum_{i=1}^{t} \frac{\partial l^t}{\partial h_i} \frac{\partial h_i}{\partial w} = \sum_{t=1}^{T} (\sum_{i=t}^{T} \frac{\partial l^i}{\partial h_t}) \frac{\partial h_t}{\partial w} \tag{B.1}$$

$$= \sum_{t=1}^{T} (\sum_{i=t}^{T} \frac{\partial l^i}{\partial h_t}) \frac{\partial h_t}{\partial w} = \sum_{t=1}^{T} \{\sum_{i=t}^{T} (\prod_{j=i}^{T-1} \frac{\partial l^{j+1}}{\partial l^j}) \frac{\partial l^i}{\partial h_t}\} \frac{\partial h_t}{\partial w}, \tag{B.2}$$

and a weight is then updated as: $w_{new} \leftarrow w_{old} - \frac{\partial L}{\partial w}$. At the end of training, the loss L will be minimized via optimal solution w^*, where $\frac{\partial}{\partial w} L(w^*) \cong 0$.

For online computation, we will have $w_{t+1} \leftarrow w_t - \sum_{i=1}^{t} \frac{\partial}{\partial w} l^i(\hat{y}^i, y^i, w_t)$ where $l^i(\hat{y}^i, y^i, w_t)$ is the cost of time step i with parameter w_t, $\hat{y}^i$ and y^i are the prediction and target label of the time step i. When the algorithm converges to an optimal solution w^* at time step φ, we will have an optimal solution where:

$$w^* - w_t = -\nabla_w(l^t) = \frac{\partial}{\partial w} l^\varphi(\hat{y}^\varphi, y^\varphi, w^*) - \frac{\partial}{\partial w} l^t(\hat{y}^t, y^t, w_t) \tag{B.3}$$

and, for one step optimization:

$$w_{t+1} - w_t = \nabla_w l^{t+1} - \nabla_w l^t. \tag{B.4}$$

This demonstrates that for any timestep, the change of weight update is proportional to the change of the gradient; this observation (Equation (B.4)) is the foundation of Forward Propagation Through Time.

Forward Propagation Through Time FPTT aims to derive an online weight update mechanism with guaranteed convergence to optimal solution w^*. To have a smooth solution, FPTT learns from the historical information of weight changes by introducing a running mean $\bar{w}_t$ to summarize the historical information of weight evolution:

$$w_{t+1} - w_t = \nabla_w(l^{t+1}) - \nabla_w(l^t) \tag{B.5}$$

$$\Rightarrow \bar{w}_t - w_t \sim \nabla_w(l^{t+1}) - \nabla_w(l^t) \tag{B.6}$$

$$\Rightarrow \nabla_w(l^{t+1}) - \nabla_w(l^t) = \alpha[(\bar{w}_t - w_t) - (w_{t+1} - w_t)] = \alpha(\bar{w}_t - w_{t+1}) \tag{B.7}$$

From this, the convergence-guaranteed loss function for online update is derived, based on Eq. (B.7).

$$\nabla_w(l^{t+1}) - \nabla_w(l^t) = \alpha(\bar{w}_t - w_{t+1}) \Leftrightarrow \nabla_w(l^{t+1}) - \nabla_w(l^t) - \alpha(\bar{w}_t - w_{t+1}) = 0 \tag{B.8}$$

we define the constraint into the function $f(w_{t+1}) = \nabla_w(l^{t+1}) - \nabla_w(l^t) - \alpha(\bar{w}_t - w_{t+1})$. We now consider a convex function $F(w)$ which approached its minimum when $f(w_{t+1}) = 0$; we then have

$$F(w) = \int_w f(w)dw - \text{searching } w_{t+1} \text{ over parameter space} \tag{B.9}$$

$$= l^t(w) + \frac{\alpha}{2}\| w - \bar{w}_t - \frac{1}{2\alpha}\nabla_w(l(w_t))\|^2 \tag{B.10}$$

In this form, Eq B.8 is the first order condition for $F(w)$. So, the weight optimization is to minimize the new objective function

$$w_{t+1} = \arg\min_w l^t(w) + \frac{\alpha}{2}\| w - \bar{w}_t - \frac{1}{2\alpha}\nabla_w(l(w_t))\|^2 \tag{B.11}$$

Bibliography

[1] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998. (Cited on page 1.)

[2] Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022. (Cited on page 1.)

[3] Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. Deepnet: Scaling transformers to 1,000 layers. *arXiv preprint arXiv:2203.00555*, 2022. (Cited on page 1.)

[4] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019. (Cited on page 1.)

[5] Lachit Dutta and Swapna Bharali. Tinyml meets iot: A comprehensive survey. *Internet of Things*, 16:100461, 2021. (Cited on page 1.)

[6] Partha Pratim Ray. A review on tinyml: State-of-the-art and prospects. *Journal of King Saud University-Computer and Information Sciences*, 2021. (Cited on page 1.)

[7] Frederico AC Azevedo, Ludmila RB Carvalho, Lea T Grinberg, José Marcelo Farfel, Renata EL Ferretti, Renata EP Leite, Wilson Jacob Filho, Roberto Lent, and Suzana Herculano-Houzel. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *Journal of Comparative Neurology*, 513(5):532–541, 2009. (Cited on page 1.)

[8] Thao Nguyen, Maithra Raghu, and Simon Kornblith. Do wide and deep networks learn the same things? uncovering how neural network representations vary with width and depth. In *International Conference on Learning Representations*, 2020. (Cited on page 2.)

[9] Lukas Pfahler and Katharina Morik. Explaining deep learning representations by tracing the training process. *arXiv preprint arXiv:2109.05880*, 2021. (Cited on page 2.)

[10] Wulfram Gerstner, Werner M. Kistler, Richard Naud, and Liam Paninski. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, 2014. (Cited on page 2.)

[11] Thomas C Südhof. Towards an understanding of synapse formation. *Neuron*, 100(2):276–293, 2018. (Cited on page 2.)

[12] Henry Markram, Joachim Lübke, Michael Frotscher, and Bert Sakmann. Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps. *Science*, 275(5297):213–215, 1997. (Cited on pages 2, 116, and 118.)

[13] Nicolas Brunel. Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. *Journal of computational neuroscience*, 8(3):183–208, 2000. (Cited on page 2.)

[14] Yuguo Yu, Peter Herman, Douglas L Rothman, Divyansh Agarwal, and Fahmeed Hyder. Evaluating the gray and white matter energy budgets of human brain function. *Journal of Cerebral Blood Flow & Metabolism*, 38(8):1339–1353, 2018. (Cited on pages 2 and 9.)

[15] Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997. (Cited on pages 2, 4, 9, and 29.)

[16] Sander M Bohte, Joost N Kok, and Han La Poutre. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48(1-4):17–37, 2002. (Cited on pages 2, 56, and 77.)

[17] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943. (Cited on pages 2 and 3.)

[18] Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14. IEEE, 2014. (Cited on pages 3, 38, 47, and 49.)

[19] Bojian Yin, Federico Corradi, and Sander M Bohte. Accurate online training of dynamical spiking neural networks through forward propagation through time. *arXiv preprint arXiv:2112.11231*, 2021. (Cited on pages 3, 22, 27, 59, 102, 109, and 113.)

[20] Bojian Yin, Federico Corradi, and Sander M Bohté. Effective and efficient computation with multiple-timescale spiking recurrent neural networks. In *International Conference on Neuromorphic Systems 2020*, pages 1–8, 2020. (Cited on pages 3, 27, 30, 58, 69, 77, 80, and 86.)

[21] Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. Technical report, Stanford Univ Ca Stanford Electronics Labs, 1960. (Cited on pages 3 and 4.)

[22] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961. (Cited on pages 3 and 4.)

[23] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. (Cited on pages 4 and 5.)

[24] Marvin Minsky and Seymour Papert. Perceptrons. 1969. (Cited on page 3.)

[25] Allan Pinkus. Weierstrass and approximation theory. *Journal of Approximation Theory*, 107(1):1–66, 2000. (Cited on page 4.)

[26] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. (Cited on pages 5 and 17.)

[27] SH Shabbeer Basha, Shiv Ram Dubey, Viswanath Pulabaigari, and Snehasis Mukherjee. Impact of fully connected layers on performance of convolutional neural networks for image classification. *Neurocomputing*, 378:112–119, 2020. (Cited on page 5.)

[28] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. (Cited on page 6.)

[29] Colin Lea, Michael D Flynn, Rene Vidal, Austin Reiter, and Gregory D Hager. Temporal convolutional networks for action segmentation and detection. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 156–165, 2017. (Cited on page 6.)

[30] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999. (Cited on pages 6 and 7.)

[31] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. (Cited on page 7.)

[32] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, 62(3):352–364, 2020. (Cited on page 7.)

[33] Tyler W Hughes, Ian AD Williamson, Momchil Minkov, and Shanhui Fan. Wave physics as an analog recurrent neural network. *Science advances*, 5(12):eaay6946, 2019. (Cited on page 7.)

[34] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994. (Cited on pages 7 and 20.)

[35] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR, 2013. (Cited on page 7.)

[36] Haichao Yu, Haoxiang Li, Humphrey Shi, Thomas S Huang, and Gang Hua. Any-precision deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10763–10771, 2021. (Cited on page 8.)

[37] Alison L Barth and James FA Poulet. Experimental evidence for sparse firing in the neocortex. *Trends in neurosciences*, 35(6):345–355, 2012. (Cited on page 9.)

[38] Bo Wang, Wei Ke, Jing Guang, Guang Chen, Luping Yin, Suixin Deng, Quansheng He, Yaping Liu, Ting He, Rui Zheng, et al. Firing frequency maxima of fast-spiking neurons in human, monkey, and mouse neocortex. *Frontiers in cellular neuroscience*, 10:239, 2016. (Cited on page 9.)

[39] Keith B Hengen, Mary E Lambo, Stephen D Van Hooser, Donald B Katz, and Gina G Turrigiano. Firing rate homeostasis in visual cortex of freely behaving rodents. *Neuron*, 80(2):335–342, 2013. (Cited on page 9.)

[40] Matthew E Larkum. Are dendrites conceptually useful? *Neuroscience*, 489:4–14, 2022. (Cited on page 9.)

[41] David Beniaguev, Idan Segev, and Michael London. Single cortical neurons as deep artificial neural networks. *Neuron*, 109(17):2727–2739, 2021. (Cited on pages 10 and 95.)

[42] Wulfram Gerstner, Werner M Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014. (Cited on pages 11 and 13.)

[43] Martin Boerlin, Christian K Machens, and Sophie Denève. Predictive coding of dynamical variables in balanced spiking networks. *PLoS computational biology*, 9(11):e1003258, 2013. (Cited on page 12.)

[44] Hu He, Qilin Wang, Xu Yang, Yunlin Lei, Jian Cai, and Ning Deng. A memory neural system built based on spiking neural network. *Neurocomputing*, 442:146–160, 2021. (Cited on page 13.)

[45] Wulfram Gerstner, Andreas K Kreiter, Henry Markram, and Andreas VM Herz. Neural codes: firing rates and beyond. *Proceedings of the National Academy of Sciences*, 94(24):12740–12741, 1997. (Cited on pages 13 and 85.)

[46] Mark Bear, Barry Connors, and Michael A Paradiso. *Neuroscience: Exploring the Brain, Enhanced Edition: Exploring the Brain*. Jones & Bartlett Learning, 2020. (Cited on page 17.)

[47] Eric R Kandel, James H Schwartz, Thomas M Jessell, Steven Siegelbaum, A James Hudspeth, Sarah Mack, et al. *Principles of neural science*, volume 4. McGraw-hill New York, 2000. (Cited on page 17.)

[48] Timothy P Lillicrap and Adam Santoro. Backpropagation through time and the brain. *Current opinion in neurobiology*, 55:82–89, 2019. (Cited on pages 17 and 19.)

[49] J Stuart et al. Artificial intelligence a modern approach third edition, 2010. (Cited on page 18.)

[50] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015. (Cited on page 18.)

[51] Zoubin Ghahramani. Unsupervised learning. In *Summer school on machine learning*, pages 72–112. Springer, 2003. (Cited on page 18.)

[52] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. (Cited on pages 18 and 119.)

[53] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013. (Cited on page 18.)

[54] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. (Cited on pages 19, 21, 30, 35, and 76.)

[55] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8624–8628. IEEE, 2013. (Cited on page 20.)

[56] Ronald J Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, 2(4):490–501, 1990. (Cited on page 20.)

[57] Corentin Tallec and Yann Ollivier. Unbiasing truncated backpropagation through time. *arXiv preprint arXiv:1705.08209*, 2017. (Cited on pages 20 and 55.)

[58] Ronald J Williams and David Zipser. Experimental analysis of the real-time recurrent learning algorithm. *Connection science*, 1(1):87–111, 1989. (Cited on page 20.)

[59] Asier Mujika, Florian Meier, and Angelika Steger. Approximating real-time recurrent learning with random kronecker factors. *Advances in Neural Information Processing Systems*, 31, 2018. (Cited on page 20.)

[60] Jacob Menick, Erich Elsen, Utku Evci, Simon Osindero, Karen Simonyan, and Alex Graves. Practical real time recurrent learning with a sparse approximation. In *International Conference on Learning Representations*, 2020. (Cited on pages 20 and 62.)

[61] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985. (Cited on page 21.)

[62] Sander M Bohte, Joost N Kok, and Johannes A La Poutré. Spikeprop: backpropagation for networks of spiking neurons. In *ESANN*, volume 48, pages 17–37, 2000. (Cited on pages 22, 29, and 36.)

[63] Sander M Bohte. Error-backpropagation in networks of fractionally predictive spiking neurons. In *International Conference on Artificial Neural Networks*, pages 60–68. Springer, 2011. (Cited on pages 22, 36, 74, and 77.)

[64] Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation*, 30(6):1514–1541, 2018. (Cited on pages 22 and 29.)

[65] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks. *arXiv preprint arXiv:1901.09948*, 2019. (Cited on pages 22, 29, 33, 36, 56, 59, 74, and 77.)

[66] Wei Fang, Zhaofei Yu, Yanqi Chen, Tiejun Huang, Timothée Masquelier, and Yonghong Tian. Deep residual learning in spiking neural networks. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021. (Cited on page 22.)

[67] Wei Fang, Zhaofei Yu, Yanqi Chen, Timothee Masquelier, Tiejun Huang, and Yonghong Tian. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2661–2671, 2021. (Cited on pages 22, 50, 75, 77, 80, 85, 86, 88, 92, and 115.)

[68] Philipp Weidel and Sadique Sheik. WaveSense: Efficient temporal convolutions with spiking neural networks for keyword spotting. November 2021. (Cited on pages 22 and 100.)

[69] Shuo-Yiin Chang, Bo Li, Gabor Simko, Tara N Sainath, Anshuman Tripathi, Aäron van den Oord, and Oriol Vinyals. Temporal modeling using dilated convolution and gating for Voice-Activity-Detection. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5549–5553. ieeexplore.ieee.org, April 2018. (Cited on pages 24, 99, 100, and 105.)

[70] Demis Hassabis, Dharshan Kumaran, Christopher Summerfield, and Matthew Botvinick. Neuroscience-inspired artificial intelligence. *Neuron*, 95(2):245–258, 2017. (Cited on page 28.)

[71] Wulfram Gerstner, Richard Kempter, J Leo Van Hemmen, and Hermann Wagner. A neuronal learning rule for sub-millisecond temporal coding. *Nature*, 383(6595):76–78, 1996. (Cited on page 29.)

[72] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018. (Cited on pages 29, 70, 71, and 72.)

[73] Sumit Bam Shrestha and Garrick Orchard. Slayer: Spike layer error reassignment in time. In *Advances in Neural Information Processing Systems*, volume 31, pages 1412–1421, 2018. (Cited on pages 29, 33, and 36.)

[74] Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, Simon J Thorpe, and Timothée Masquelier. Stdp-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, 99:56–67, 2018. (Cited on pages 29 and 116.)

[75] Pierre Falez, Pierre Tirilly, Ioan Marius Bilasco, Philippe Devienne, and Pierre Boulet. Multi-layered spiking neural network with target timestamp threshold adaptation and stdp. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019. (Cited on pages 29, 116, and 118.)

[76] Timo C Wunderlich and Christian Pehle. Eventprop: Backpropagation for exact gradients in spiking neural networks. *arXiv preprint arXiv:2009.08378*, 2020. (Cited on page 29.)

[77] Guillaume Bellec, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature Communications*, 11(1):1–15, 2020. (Cited on pages 30, 32, 36, 39, 42, 51, 74, 77, 78, 82, 85, 86, 92, 115, and 118.)

[78] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: VGG and residual architectures. *Front. Neurosci.*, 13:95, March 2019. (Cited on page 30.)

[79] Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, 2019. (Cited on pages 30, 38, and 47.)

[80] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018. (Cited on pages 30 and 36.)

[81] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Expected energy-based restricted boltzmann machine for classification. *Neural networks*, 64:29–38, 2015. (Cited on pages 30 and 36.)

[82] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. (Cited on pages 32 and 86.)

[83] Raphael Hunger. *Floating point operations in matrix-vector calculus*. Munich University of Technology, Inst. for Circuit Theory and Signa, 2005. (Cited on pages 33 and 37.)

[84] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002. (Cited on pages 33, 51, and 55.)

[85] Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003. (Cited on pages 33, 34, and 56.)

[86] Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. In *Advances in Neural Information Processing Systems*, pages 787–797, 2018. (Cited on pages 34, 36, and 42.)

[87] Alexander Wong, Mahmoud Famouri, Maya Pavlova, and Siddharth Surana. Tinyspeech: Attention condensers for deep speech recognition neural networks on edge devices. *arXiv preprint arXiv:2008.04245*, 2020. (Cited on pages 37, 50, 51, and 130.)

[88] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. (Cited on pages 37 and 83.)

[89] Percy E Ludgate. On a proposed analytical machine. In *The Origins of Digital Computers*, pages 73–87. Springer, 1982. (Cited on page 38.)

[90] Apeksha Shewalkar, Deepika Nyavanandi, and Simone A Ludwig. Performance evaluation of deep neural networks applied to speech recognition: Rnn, lstm and gru. *Journal of Artificial Intelligence and Soft Computing Research*, 9(4):235–245, 2019. (Cited on page 38.)

[91] Pablo Laguna, Roger G Mark, A Goldberg, and George B Moody. A database for evaluation of algorithms for measurement of qt and other waveform intervals in the ecg. In *Computers in cardiology 1997*, pages 673–676. IEEE, 1997. (Cited on pages 38 and 40.)

[92] Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke. The heidelberg spiking datasets for the systematic evaluation of spiking neural networks. *arXiv preprint arXiv:1910.07407*, 2019. (Cited on pages 39, 41, 42, and 130.)

[93] Saiwen Wang, Jie Song, Jaime Lien, Ivan Poupyrev, and Otmar 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*, pages 851–860, 2016. (Cited on pages 39, 41, 42, and 47.)

[94] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018. (Cited on pages 39, 41, 108, and 111.)

[95] John S Garofolo. Timit acoustic phonetic continuous speech corpus. *Linguistic Data Consortium, 1993*, 1993. (Cited on page 39.)

[96] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128x128 120 db 15μs latency asynchronous temporal contrast vision sensor. *IEEE journal of solid-state circuits*, 43(2):566–576, 2008. (Cited on page 41.)

[97] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, volume 8, pages 18–25, 2015. (Cited on page 41.)

[98] Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5457–5466, 2018. (Cited on page 42.)

[99] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128, 2016. (Cited on page 42.)

[100] Nicolas Perez-Nieves, Vincent CH Leung, Pier Luigi Dragotti, and Dan FM Goodman. Neural heterogeneity promotes robust learning. *bioRxiv*, pages 2020–12, 2021. (Cited on page 42.)

[101] Douglas Coimbra de Andrade, Sabato Leo, Martin Loesener Da Silva Viana, and Christoph Bernkopf. A neural attention model for speech command recognition. *arXiv preprint arXiv:1808.08929*, 2018. (Cited on page 42.)

[102] Thomas Pellegrini, Romain Zimmer, and Timothée Masquelier. Low-Activity supervised convolutional spiking neural networks applied to speech commands recognition. In *2021 IEEE Spoken Language Technology Workshop (SLT)*, pages 97–103, January 2021. (Cited on pages 42 and 43.)

[103] Friedemann Zenke and Tim P. Vogels. The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *Neural Computation*, 0(0):1–27, 2021. (Cited on page 42.)

[104] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6):602–610, 2005. (Cited on page 42.)

[105] Souvik Kundu, Gourav Datta, Massoud Pedram, and Peter A Beerel. Spike-thrift: Towards energy-efficient deep spiking neural networks by limiting spiking activity via attention-guided compression. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3953–3962, 2021. (Cited on pages 47 and 51.)

[106] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, et al. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7243–7252, 2017. (Cited on pages 50, 75, and 84.)

[107] Thomas Pellegrini, Romain Zimmer, and Timothée Masquelier. Low-activity supervised convolutional spiking neural networks applied to speech commands recognition. In *2021 IEEE Spoken Language Technology Workshop (SLT)*, pages 97–103. IEEE, 2021. (Cited on page 50.)

[108] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. 2019. (Cited on pages 50, 59, and 74.)

[109] Friedemann Zenke, Sander M Bohté, Claudia Clopath, Iulia M.Comşa, Julian Göltz, Wolfgang Maass, Timothée Masquelier, Richard Naud, Emre O Neftci, Mihai A Petrovici, Franz Scherr, and Dan F M Goodman. Visualizing a joint future of neuroscience and neuromorphic engineering. *Neuron*, 109(4):571–575, February 2021. (Cited on page 51.)

[110] F. Zenke and E. O. Neftci. Brain-inspired learning on neuromorphic substrates. *Proceedings of the IEEE*, pages 1–16, 2021. (Cited on page 51.)

[111] Joram Keijser and Henning Sprekeler. Interneuron diversity is required for compartment-specific feedback inhibition. *bioRxiv*, 2020. (Cited on page 51.)

[112] Bojian Yin Anca-Diana Vicol and S.M Bohte. Real-time classification of lidar data using discrete-time recurrent spiking neural networks. *IJCNN*, 2022. (Cited on page 53.)

[113] Priyadarshini Panda, Sai Aparna Aketi, and Kaushik Roy. Toward scalable, efficient, and accurate deep spiking neural networks with backward residual connections, stochastic softmax, and hybridization. *Frontiers in Neuroscience*, 14, 2020. (Cited on page 54.)

[114] Wulfram Gerstner. *Biological Learning: Synaptic Plasticity, Hebb Rule and Spike TimingDependent Plasticity*, pages 111–132. Springer US, Boston, MA, 2010. (Cited on page 56.)

[115] André Grüning and Sander M Bohte. Spiking neural networks: Principles and challenges. In *ESANN*. Bruges, 2014. (Cited on page 56.)

[116] Iulia M Comsa, Krzysztof Potempa, Luca Versari, Thomas Fischbacher, Andrea Gesmundo, and Jyrki Alakuijala. Temporal coding in spiking neural networks with alpha synaptic function. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8529–8533. IEEE, 2020. (Cited on page 56.)

[117] Hesham Mostafa. Supervised learning based on temporal coding in spiking neural networks. *IEEE transactions on neural networks and learning systems*, 29(7):3227–3235, 2017. (Cited on page 56.)

[118] Shibo Zhou and Wei Wang. Object detection based on lidar temporal pulses using spiking neural networks. *arXiv preprint arXiv:1810.12436*, 2018. (Cited on pages 56 and 57.)

[119] Friedemann Zenke and Tim P Vogels. The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *Neural Computation*, 33(4):899–925, 2021. (Cited on page 56.)

[120] Michael Himmelsbach, Andre Mueller, Thorsten Lüttel, and Hans-Joachim Wünsche. Lidar-based 3d object perception. In *Proceedings of 1st international workshop on cognition for technical systems*, volume 1, 2008. (Cited on page 56.)

[121] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. Squeezeseg: Convolutional neural nets with recurrent CRF for real-time road-object segmentation from 3d lidar point cloud. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1887–1893. IEEE, 2018. (Cited on page 56.)

[122] Danil V Prokhorov. Object recognition in 3d lidar data with recurrent neural network. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 9–15. IEEE, 2009. (Cited on page 57.)

[123] Wei Wang, Shibo Zhou, Jingxi Li, Xiaohua Li, Junsong Yuan, and Zhanpeng Jin. Temporal pulses driven spiking neural network for time and power efficient object recognition in autonomous driving. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 6359–6366, 2021. (Cited on page 57.)

[124] Shibo Zhou, Ying Chen, Xiaohua Li, and Arindam Sanyal. Deep scnn-based real-time object detection for self-driving vehicles using lidar temporal data. *IEEE Access*, 8:76903–76912, 2020. (Cited on pages 57 and 76.)

[125] Garrick Orchard, Cedric Meyer, Ralph Etienne-Cummings, Christoph Posch, Nitish Thakor, and Ryad Benosman. Hfirst: A temporal approach to object recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(10):2028–2040, 2015. (Cited on page 57.)

[126] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013. (Cited on page 59.)

[127] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012. (Cited on page 59.)

[128] Sander M Bohte, Han La Poutré, and Joost N Kok. Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer rbf networks. *IEEE Transactions on neural networks*, 13(2):426–435, 2002. (Cited on pages 61 and 62.)

[129] Sharan Narang, Erich Elsen, Gregory Diamos, and Shubho Sengupta. Exploring sparsity in recurrent neural networks. *arXiv preprint arXiv:1704.05119*, 2017. (Cited on page 62.)

[130] Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aaron Oord, Sander Dieleman, and Koray Kavukcuoglu. Efficient neural audio synthesis. In *International Conference on Machine Learning*, pages 2410–2419. PMLR, 2018. (Cited on page 62.)

[131] Anil Kag and Venkatesh Saligrama. Training recurrent neural networks via forward propagation through time. In *International Conference on Machine Learning*, pages 5189–5200. PMLR, 2021. (Cited on pages 73, 74, 75, 78, 79, 83, 86, 88, 90, and 131.)

[132] Sander M. Bohté Bojian Yin, Federico Corradi. Accurate online training of dynamical spiking neural networks through forward propagation through time. 2022. (Cited on page 74.)

[133] Bojian Yin, Federico Corradi, and Sander M. Bohté. Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks. *Nat Mach Intell*, 3:905–913, 2021. (Cited on pages 74, 77, 78, 81, 82, 86, 88, and 92.)

[134] Jan Stuijt, Manolis Sifalakis, Amirreza Yousefzadeh, and Federico Corradi. μbrain: An event-driven and fully synthesizable architecture for spiking neural networks. *Frontiers in neuroscience*, 15:538, 2021. (Cited on pages 74, 75, and 115.)

[135] Nicolas Perez-Nieves, Vincent C H Leung, Pier Luigi Dragotti, and Dan F M Goodman. Neural heterogeneity promotes robust learning. *Nat. Commun.*, 12(1):5791, October 2021. (Cited on page 74.)

[136] Joram Keijser and Henning Sprekeler. Interneuron diversity is required for compartment-specific feedback inhibition. *bioRxiv*, 2020. (Cited on page 74.)

[137] A Mehonic and A J Kenyon. Brain-inspired computing needs a master plan. *Nature*, 604(7905):255–260, April 2022. (Cited on page 74.)

[138] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989. (Cited on pages 74 and 76.)

[139] Thomas Bohnstingl, Stanisław Woźniak, Wolfgang Maass, Angeliki Pantazi, and Evangelos Eleftheriou. Online spatio-temporal learning in deep neural networks. *arXiv preprint arXiv:2007.12723*, 2020. (Cited on pages 74, 76, 78, 86, 92, and 115.)

[140] Yuming He, Federico Corradi, Chengyao Shi, Ming Ding, Martijn Timmermans, Jan Stuijt, Pieter Harpe, Ilja Ocket, and Yao-Hong Liu. A 28.2 μw neuromorphic sensing system featuring snn-based near-sensor computation and event-driven body-channel communication for insertable cardiac monitoring. In *2021 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, pages 1–3, 2021. (Cited on page 75.)

[141] Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. Liquid time-constant networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7657–7666, 2021. (Cited on pages 75 and 78.)

[142] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010. (Cited on pages 76, 85, and 87.)

[143] Seijoon Kim, Seongsik Park, Byunggook Na, and Sungroh Yoon. Spiking-yolo: spiking neural network for energy-efficient object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 11270–11277, 2020. (Cited on pages 76, 87, and 95.)

[144] Biswadeep Chakraborty, Xueyuan She, and Saibal Mukhopadhyay. A fully spiking hybrid neural network for energy-efficient object detection. *IEEE Transactions on Image Processing*, 30:9014–9029, 2021. (Cited on page 76.)

[145] Joaquin Royo-Miquel, Silvia Tolu, Frederik ET Schöller, and Roberto Galeazzi. Retinanet object detector based on analog-to-spiking neural network conversion. In *8th International Conference on Soft Computing & Machine Intelligence*, 2021. (Cited on page 76.)

[146] Zicong Jiang, Liquan Zhao, Shuaiyang Li, and Yanfei Jia. Real-time object detection method based on improved yolov4-tiny. *arXiv preprint arXiv:2011.04244*, 2020. (Cited on pages 76, 87, and 92.)

[147] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990. (Cited on page 76.)

[148] Michael C Mozer. Neural net architectures for temporal sequence processing. In *Santa Fe Institute Studies in the Sciences of Complexity-Proceedings Volume-*, volume 15, pages 243–243, 1993. (Cited on page 76.)

[149] James M Murray. Local online learning in recurrent networks with random feedback. *ELife*, 8:e43299, 2019. (Cited on page 76.)

[150] James C Knight and Thomas Nowotny. Efficient GPU training of LSNNs using eprop. In *Neuro-Inspired Computational Elements Conference*, NICE 2022, pages 8–10, 2022. (Cited on page 76.)

[151] Franz Scherr and Wolfgang Maass. Analysis of the computational strategy of a detailed laminar cortical microcircuit model for solving the image-change-detection task. *bioRxiv*, 2021. (Cited on page 78.)

[152] Hongmin Li, Hanchao Liu, Xiangyang Ji, Guoqi Li, and Luping Shi. Cifar10-dvs: an event-stream dataset for object classification. *Frontiers in neuroscience*, 11:309, 2017. (Cited on page 85.)

[153] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020. (Cited on pages 87 and 92.)

[154] Stanisław Woźniak, Angeliki Pantazi, Thomas Bohnstingl, and Evangelos Eleftheriou. Deep learning incorporating biologically inspired neural dynamics and in-memory computing. *Nature Machine Intelligence*, 2(6):325–336, 2020. (Cited on page 92.)

[155] Jacques Kaiser, Hesham Mostafa, and Emre Neftci. Synaptic plasticity dynamics for deep continuous local learning (decolle). *Frontiers in Neuroscience*, 14:424, 2020. (Cited on pages 92 and 118.)

[156] João Sacramento, Rui Ponte Costa, Yoshua Bengio, and Walter Senn. Dendritic cortical microcircuits approximate the backpropagation algorithm. *Advances in Neural Information Processing Systems: NeurIPS*, 31:8721–8732, 2018. (Cited on page 95.)

[157] Matthew E Larkum, Walter Senn, and Hans-R Lüscher. Top-down dendritic input increases the gain of layer 5 pyramidal neurons. *Cereb. Cortex*, 14(10):1059–1070, October 2004. (Cited on page 95.)

[158] Uwe Frey and Richard GM Morris. Synaptic tagging and long-term potentiation. *Nature*, 385(6616):533–536, 1997. (Cited on page 95.)

[159] Diego Moncada, Fabricio Ballarini, María Cecilia Martinez, Julietta U Frey, and Haydée Viola. Identification of transmitter systems and learning tag molecules involved in behavioral tagging during memory formation. *Proceedings of the National Academy of Sciences*, 108(31):12931–12936, 2011. (Cited on page 95.)

[160] Jaldert O Rombouts, Sander M Bohte, and Pieter R Roelfsema. How attention can create synaptic tags for the learning of working memories in sequential tasks. *PLoS computational biology*, 11(3):e1004060, 2015. (Cited on page 96.)

[161] Isabella Pozzi, Sander Bohte, and Pieter Roelfsema. Attention-gated brain propagation: How the brain can implement reward-based error backpropagation. *Advances in Neural Information Processing Systems: NeurIPS*, 33, 2020. (Cited on page 96.)

[162] Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24, 2017. (Cited on page 96.)

[163] Henk Corporaal Federico Corradi Bojian Yin, Qinghai Guo and Sander M. Bohté. Attentive decision-making and dynamic resetting of continual running srnns for end-to-end streaming keyword spotting. *ICONS*, 2022. (Cited on page 97.)

[164] Juntae Kim, Jeehye Lee, and Yoonhan Lee. Generalizing RNN-Transducer to Out-Domain audio via sparse Self-Attention layers. August 2021. (Cited on pages 99, 100, and 105.)

[165] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. Hello edge: Keyword spotting on microcontrollers. November 2017. (Cited on page 100.)

[166] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 369–376, New York, NY, USA, June 2006. Association for Computing Machinery. (Cited on page 100.)

[167] Chunyang Wu, Yongqiang Wang, Yangyang Shi, Ching-Feng Yeh, and Frank Zhang. Streaming transformer-based acoustic models using self-attention with augmented memory. May 2020. (Cited on page 100.)

[168] Emre Yılmaz, Özgür Bora Gevrek, Jibin Wu, Yuxiang Chen, Xuanbo Meng, and Haizhou Li. Deep convolutional spiking neural networks for keyword spotting. In *Interspeech 2020*, ISCA, October 2020. ISCA. (Cited on page 100.)

[169] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio. *SSW*, 125:2, 2016. (Cited on page 100.)

[170] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. (Cited on page 101.)

[171] Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. In S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, and R Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 787–797. Curran Associates, Inc., 2018. (Cited on page 102.)

[172] Kevin Gurney, Tony J Prescott, and Peter Redgrave. A computational model of action selection in the basal ganglia. i. a new functional anatomy. *Biological cybernetics*, 84(6):401–410, 2001. (Cited on page 104.)

[173] Davide Zambrano, Pieter R Roelfsema, and Sander Bohte. Learning continuous-time working memory tasks with on-policy neural reinforcement learning. *Neurocomputing*, 461:635–656, 2021. (Cited on page 104.)

[174] Philipp Weidel and Sadique Sheik. Wavesense: Efficient temporal convolutions with spiking neural networks for keyword spotting. *arXiv preprint arXiv:2111.01456*, 2021. (Cited on page 109.)

[175] Vladimir I Levenshtein et al. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union, 1966. (Cited on page 111.)

[176] Tara Sainath, Ron J Weiss, Kevin Wilson, Andrew W Senior, and Oriol Vinyals. Learning the speech front-end with raw waveform cldnns. 2015. (Cited on page 113.)

[177] Kwantae Kim, Chang Gao, Rui Graça, Ilya Kiselev, Hoi-Jun Yoo, Tobi Delbruck, and Shih-Chii Liu. A 23μw solar-powered keyword-spotting asic with ring-oscillator-based time-domain feature extraction. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pages 1–3. IEEE, 2022. (Cited on pages 113 and 122.)

[178] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in neuroscience*, 13, 2019. (Cited on page 115.)

[179] Yueting Shi, Hai Li, Hehui Zhang, Zhenzhi Wu, and Shiwei Ren. Accurate and efficient lif-nets for 3d detection and recognition. *IEEE Access*, 8:98562–98571, 2020. (Cited on page 115.)

[180] Yujie Wu, Rong Zhao, Jun Zhu, Feng Chen, Mingkun Xu, Guoqi Li, Sen Song, Lei Deng, Guanrui Wang, Hao Zheng, et al. Brain-inspired global-local learning incorporated with neuromorphic computing. *Nature Communications*, 13(1):1–14, 2022. (Cited on page 116.)

[181] David Silver, Anirudh Goyal, Ivo Danihelka, Matteo Hessel, and Hado van Hasselt. Learning by directional gradient descent. In *International Conference on Learning Representations*, 2021. (Cited on page 116.)

[182] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7(1):1–10, 2016. (Cited on pages 116 and 119.)

[183] Stephen Whitelam, Viktor Selin, Sang-Won Park, and Isaac Tamblyn. Correspondence between neuroevolution and gradient descent. *Nature communications*, 12(1):1–10, 2021. (Cited on page 116.)

[184] Amr M AbdelAty, Mohammed E Fouda, and Ahmed Eltawil. Parameter estimation of two spiking neuron models with meta-heuristic optimization algorithms. *Frontiers in Neuroinformatics*, 16, 2022. (Cited on page 116.)

[185] Kristofor D Carlson, Jayram Moorkanikara Nageswaran, Nikil Dutt, and Jeffrey L Krichmar. An efficient automated parameter tuning framework for spiking neural networks. *Frontiers in neuroscience*, 8:10, 2014. (Cited on page 116.)

[186] Junxiu Liu, Xingyue Huang, Dong Jiang, and Yuling Luo. An energy-aware spiking neural network hardware mapping based on particle swarm optimization and genetic algorithm. In *2020 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 11–13. IEEE, 2020. (Cited on page 116.)

[187] NG Pavlidis, OK Tasoulis, Vassilis P Plagianakos, G Nikiforidis, and MN Vrahatis. Spiking neural network training using evolutionary algorithms. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 4, pages 2190–2194. IEEE, 2005. (Cited on page 116.)

[188] Jesus L Lobo, Javier Del Ser, Albert Bifet, and Nikola Kasabov. Spiking neural networks and online learning: An overview and perspectives. *Neural Networks*, 121:88–100, 2020. (Cited on page 116.)

[189] Nicolas Perez-Nieves and Dan Goodman. Sparse spiking gradient descent. *Advances in Neural Information Processing Systems*, 34:11795–11808, 2021. (Cited on pages 117 and 118.)

[190] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014. (Cited on page 119.)

[191] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. *Advances in neural information processing systems*, 29, 2016. (Cited on page 119.)

[192] Ehsan Amid, Rohan Anil, and Manfred Warmuth. Locoprop: Enhancing backprop via local loss optimization. In *International Conference on Artificial Intelligence and Statistics*, pages 9626–9642. PMLR, 2022. (Cited on page 119.)

[193] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *International conference on machine learning*, pages 1627–1635. PMLR, 2017. (Cited on page 119.)

[194] Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. *Advances in neural information processing systems*, 30, 2017. (Cited on page 119.)

[195] Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Accelerating adversarial training via maximal principle. *Advances in Neural Information Processing Systems*, 32, 2019. (Cited on page 119.)

Curriculum Vitae

Bojian Yin

07-03-1993: Born in Shanxi,China

Experience

Sep 2022 – present:	Researcher at *CWI, Amsterdam*
Sep 2018 -Sep 2022:	PhD student at *CWI, Amsterdam*
Sep 2017 -Sep 2018:	Research Assistant at *CWI, Amsterdam*
Mar 2017 -Sep 2017:	Master Thesis Internship at *CWI, Amsterdam*

Education

Sep 2018 -Sep 2022: **Ph.D.**,*Electrical Engineering*
Technische Universiteit Eindhoven
Promoter: Prof. dr. S.M. Bohté and Prof.dr. H. Corporaal
CoPromoter: dr. F. Corradi
Thesis: Efficient and Accurate Spiking Neural Networks

Sep 2015–Sep 2017: **M.S.**, *Artificial Intelligence*
Vrije Universiteit Amsterdam
joint AI program with University of Amsterdam(UvA)
Thesis: An image representation based convolutional network for DNA classification

Sep 2014 -Sep 2015: Master Computational mathematics
North China Electric Power University

Sep 2010 -Sep 2014: **B.S.**, *Information and Computer Science*
Tangshan Normal University
Thesis: Using neural networks for DNA sequence classification