Memory Mechanisms in Spiking Neural Networks

L.D.J. Lammers

Faculty of Aerospace Engineering

TU Delft

Memory Mechanisms in Spiking Neural Networks

by

L.D.J. Lammers

to obtain the degree of Master of Science
at the Delft University of Technology.

Student number:	4394933	
Project duration:	Feb, 2022 – Jul, 2023	
Readers:	J. Hagenaars,	TU Delft, supervisor
	Prof. G. de Croon,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Acknowledgements

I would like to express my gratitude to my master thesis supervisors, Jesse and Guido, for their guidance, constructive feedback, but most of all their support throughout the course of my thesis. Your expertise and commitment have been instrumental in the completion of this thesis. Jesse, your enthusiasm and passion for this field have been contagious, igniting my own excitement for the research. Your ability to simplify complex concepts and communicate them clearly to me has been invaluable to my learning process. I also really appreciate your accessibility. It didn't matter what time of the week I sent you a question, I almost always got a reply in no time. Guido, I never understood how within a few minutes of our meeting you were able to immediately expose cracks in my work, even though you had completely forgotten what I was doing at the start of the meeting. This kept me on my toes and I appreciate the detailed feedback.

L.D.J. Lammers
Rotterdam, July 2023

Abstract

Neuromorphic sensors, like for example event cameras, detect incremental changes in the sensed quantity and communicate these via a stream of events. Desired properties of these signals such as high temporal resolution and asynchrony are not always fully exploited by algorithms that process these signals. Spiking neural networks (SNNs) have emerged as the algorithms that promise to maximally attain these characteristics and are likely the key to achieving a fully neuromorphic computing pipeline. But, this means that if the SNN is to take full advantage, the event stream must be sent directly and unaltered to the SNN, which in turn implies that all temporal integration should occur inside the SNN. Therefore, it is interesting to investigate the mechanisms that achieve this. This thesis does so through evaluating and comparing the performance of different memory mechanisms in SNNs found in the literature, as well as through an in depth analysis of the inner workings of these mechanisms. The mechanisms include spiking neural dynamics (leaks and thresholds), explicit recurrent connections, and propagation delays. We demonstrate our concepts on two small scale generated 1D moving pixel tasks in preliminary experiments first. After that, we extend our research to compare the memory mechanisms on a real-world neuromorphic vision processing task, in which the networks regress angular velocity given event based input. We find that both explicit recurrency and delays improve the prediction accuracy of the SNN, compared to having just spiking neuronal dynamics. Analysis of the inner workings of the networks shows that the threshold and reset mechanism of spiking neurons play an important role in allowing longer neuron timescales (lower membrane leak). Forgetting (at the right time) turns out to play an important role in memory. Additionally, it becomes apparent that optimizing an SNN with explicit recurrent connections or learnable delays does not lead to the formation of robust spiking neuronal dynamics. In fact, spiking neuronal dynamics are largely ignored, as after optimization virtually no input current is integrated onto the membrane potential in these cases. Instead, we consistently find that a recurrent SNN prefers to build a state solely with the explicit recurrent connections, while an SNN with delays prefers to just use the delays. Therefore, our SNNs with explicit recurrent connections and delays are in fact better described as binary activated RNNs and ANNs, respectively.

Contents

List of Symbols

Math symbols

$\rightarrow$	approach
$\approx$	approximately equal
$\leftarrow$	assignment
$*$	convolution
$\doteq$	equality that is true by definition
$\exp(\cdot)$	exponential function
∞	to infinity and beyond
∇	nabla operator
$\neq$	inequality
$\dot{a}$	derivative of a w.r.t. time t
$\mathcal{N}(\mu, \sigma)$	normal distribution with mean μ and standard deviation σ
Δ	difference operator
Θ	Heaviside step function
δ	Dirac delta function

Greek symbols

Δt	simulation time step duration
α	synaptic leak constant
β	membrane leak constant
η	refractory kernel in SRM
θ	neural firing threshold
μ	mean
σ	standard deviation
τ_a	time constant of decay of variable a
ϵ	membrane filter in SRM
$\phi(\cdot)$	neural activation function

Latin symbols

b	bias
c	correlation coefficient
C	output feature map of convolution
d	delay
U_{rest}	neural membrane potential in rest
H	height
$I(t)$	synaptic current
P	event polarity
$S(t)$	spike train
t	time
$U(t)$	neural membrane potential
W	width
w	weight or synaptic efficacy

Sub- and superscripts

i	postsynaptic neuron index
j	presynaptic neuron index

(l)	index of neural network layer
pre, *post*	pre- and postsynaptic contribution subscripts

List of Abbreviations

A

a.u.	arbitrary units
AER	address-event representation
ANN	artificial neural network

B

BPTT	back-propagation-through-time

C

CD	coincidence detector
CPU	central processing unit

D

DL	deep learning
DVS	dynamic vision sensor

E

EDP	energy-delay-product

F

FNN	feedforward neural network

G

GPU	graphics processing unit
GRU	gated recurrent unit

I

IF	integrate-and-fire
indRNN	independently recurrent neural network

L

LIF	leaky integrate-and-fire
LSTM	long-short-term-memory

M

MAC	multiply-and-accumulate
MNIST	modified National Institute of Standards and Technology

N

N-MNIST	neuromorphic modified National Institute of Standards and Technology

P

PSP	post-synaptic potential

R

ReLU	rectifying linear unit
ResNets	residual networks
RNN	recurrent neural network

S

sEMD	spiking elementary motion detector
SG	surrogate gradient
SHD	spoken Heidelberg digits
SLAM	simultaneous localisation and mapping
SLAYER	spike layer error reassignment
SpiNNaker	Spiking Neural Network Architecture
SRM	spike response model
SRNN	spiking recurrent neural network
STBP	spatiotemporal backpropagation
STDP	spike-timing-dependent plasticity

T

TDE	time difference encoder
TTFS	Time To First Spike

List of Figures

List of Tables

List of Listings

1 Introduction

In recent years, the field of neuromorphic computing has gained significant attention as researchers strive to create machines that can handle data faster and more efficiently than current computing systems. The traditional Von Neumann architecture, which is the accepted standard for computing machines, has significant differences when compared to the human brain in terms of organization, power requirements, and processing capabilities (Schuman et al., 2022). This has led to the question of whether it is possible to design alternative architectures that are inspired by the brain and potentially even superior to a biological brain.

One promising approach in this field is the use of Spiking Neural Networks (SNNs), which are a family of algorithms that mimic the biological behavior of neurons and synapses. These algorithms have shown to be capable of producing competitive performance while significantly improving computational efficiency, particularly when used in settings with time-varying signals and in combinationwith dedicated hardware (Davies et al., 2021; Göltz et al., 2021; Hagenaars et al., 2021; Yin et al., 2020). As the integration of algorithms and hardware continues to improve, the potential for SNNs to achieve even greater performance and efficiency remains enormous.

One area of particular interest is the development of a neuromorphic pipeline that mimics the biological way of perceiving. Take for example human visual perception. The retina, a key organ in the visual system, turns the spatiotemporal information in the visual scene's light into spike trains and patterns through adaptive filtering and sampling methods to increase coding efficiency (Posch et al., 2014). These spike trains are then transferred to the visual cortex, where cognitive perception occurs. To replicate this process in a neuromorphic pipeline, it is crucial to transfer the spike trains from a silicon retina to a spiking neural network (SNN) in an unaltered and one-by-one manner, and allow the network to handle the temporal integration of signals.

Because nature demonstrates that most temporal integration of input signals should occur inside the neural network, it is important to investigate how to manipulate SNNs to extract spatiotemporal features of data streams accurately and efficiently. To that end this research will address which architectures, neuron models, learning procedures and other characteristics promote or prevent the formation of robust temporal integration of information.

1.1 Research opportunities & research question

Various approaches to incorporating memory in SNNs are found in the existing literature. Many techniques are designed to solve one specific task and the models differ significantly in neuron model, learning procedure, network size and many other parameters. Until now, no systematic and comprehensive research has been carried out in order to fairly compare these approaches and find out which architectures, neurons models, learning procedures have a positive (or negative) impact on temporal integration of information inside an SNN. This thesis aims to bridge this knowledge gap. Therefore, the main research question of this thesis is:

Which spiking neural network (SNN) architectures, neuron models, learning procedures, and potentially other configurations promote or prevent the formation of robust temporal integration of information?"

Integration of information inside an SNNrequires the saving (buffering) of spikes from previous states on hardware for later use. This buffering of spikes can, especially when implemented on real-time neuromorphic hardware devices, incur a significant memory and computing overhead. It would be interesting which memory mechanism are more affected or less affected by this. Therefore, the first sub-question is:

To what extent introduce different memory mechanisms in SNNs an overhead in terms of memory usage and computing power?

Moreover, the temporal integration of information requires that input signals propagate the network at different rates. Information fromone input time step has to come together with information from another time step in the same node in the SNN for it to be successful. This implies that potentially significant latency can arise during inference. Therefore, the second sub-question is:

Which memory mechanisms, and to what extent are different memory mechanisms, in SNNs accompanied by latency during inference?

Finally, it is interesting to compare memory mechanisms in SNNs (where possible) to equivalentmemory mechanismimplementations in non-spiking neural networks. The comparison can be done using various criteria such as accuracy on a certain task, energy efficiency or length ofmemory. Therefore, the third sub-question is:

How does incorporating spiking neuronal dynamics into a neural network compare to having no spiking dynamics?

1.2 Structure of this work

This thesis consists of four parts. Part Part I contains a standalone scientific paper that presents the main contributions of this thesis.

Part II gives an in-depth account of relevant literature on the topics of neuromorphic computing, spiking neural networks and memory mechanisms in SNNs. In Chapter 2, neuromorphic computing is introduced and in particular the differences and advantadges compared to traditional computing approaches are highlighted. Subsequently, Chapter 3 elaborates on SNNs. Among others, different neuron models, learning approaches and performance on neuromorphic hardware are discussed. Finally, Chapter 4 examines the different methods for integrating information inside SNN that are found in the existing literature.

Next, Part III discusses the preliminary experiments performed to gain a deeper understanding of the topic. These experiments consist of testing various ANN and SNN models with different forms of internal memory on two artificially created tasks that require memory to solve. Chapter 5 contains the methodology of the preleminary experiments where the various models and two tasks are described in detail. In Chapter 6 presents the results of the preliminary experiments and also discusses them. Finally, Chapter 7 concludes with the main takeaways of the preliminary experiments.

Finally, Part IV contains the appendices. Appendix A.1 contains the network parameters as used for the preliminary experiments, Appendix A.2 a very basic PyTorch implementation of a SNN and Appendix

A.3 the results of a small ablation study to find the optimal settings for gradient clipping.

I

Scientific Paper

An Investigation of Different Memory Mechanisms in Spiking Neural Networks in the Context of Event-Based Vision

L.D.J. Lammers[1], J.J. Hagenaars[2], G.C.H.E. de Croon[2]

***Abstract*— Neuromorphic vision sensors detect brightness changes in individual pixels asynchronously and with high temporal resolution. To maximally attain these desirable signal characteristics, brightness changes are passed directly as 'events' to spiking neural networks (SNNs) for processing. Since these events arrive asynchronously and with low latency, the SNN has to integrate information from these events over time. In this article, we provide a systematic performance comparison and analysis of different memory (integration) mechanisms in SNNs. We distinguish three classes of mechanisms with which we augment a base neural network architecture: leaks, explicit recurrent connections, and synaptic delays. We compare these mechanisms on a neuromorphic vision processing task, in which the network regresses angular velocity given event based input. We find that all three integration mechanisms provide enough temporal integration to achieve performance competitive with previous work. Additionally, we find that the use of explicit recurrent connections and learnable axonal delays strongly inhibits the formation of robust spiking neuronal dynamics during optimization, to the point where the SNN approaches a binary activation network. Also, the importance of a threshold with a membrane potential reset mechanism emerges, even for conventional artificial neural networks (ANNs). Adding a thresholded ReLu with reset mechanism to a leaky feedforward ANN not only improves speed of convergence, but also improves prediction accuracy.**

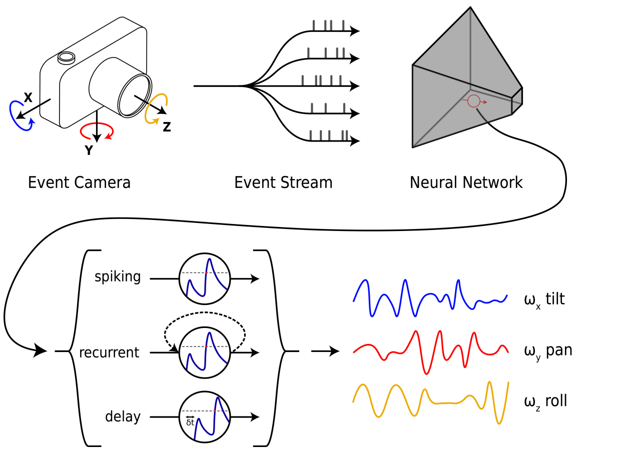


Fig. 1: Schematic representation of our neuromorphic computing pipeline. Events from the event camera enter the encoder network without any temporal encoding in the input representation. The base encoder network can be augmented with three different memory mechanisms (spiking neurons, explicit recurrent connections or propagation delays) that introduce information integration inside the network.

I. Introduction

Researchers in the fields of neuroscience, machine learning, and robotics draw substantial inspiration from biological species such as winged insects. The ability of these small animals to perform highly complex tasks, such as visual perception and rapid movement in cluttered environments, with extremely limited computational resources showcases their efficiency. This discipline of attempting to compute in an efficient manner that closely resembles the biological way of interpreting and processing information is known as *neuromorphic computing* [1, 2].

Efforts to construct an end-to-end neuromorphic computing pipeline focus primarily on three aspects: (1) acquiring signals with neuromorphic characteristics such as sparsity and spike-based information encoding, (2) developing algorithms for processing these signals in a brain-like manner, and (3) creating hardware that supports these neuromorphic characteristics. The event camera is an example of an innovation aimed at achieving the first aspect. Event cameras detect per-pixel brightness *changes* asynchronously and output these as a stream of events, rather than continuously reporting a value at a fixed frequency [3]. High temporal resolution (on the order of μs), very large dynamic range (140dB vs. 60dB for CMOS cameras), low power consumption, and high pixel bandwidth (on the scale of kHz) are favourable properties that result in reduced motion blur for event cameras compared to traditional cameras [3, 4].

Due to the unique characteristics of event-driven signals, such as sparsity and asynchrony, conventional signal processing methods often cannot be used without compromising these hard-earned characteristics. Spiking neural networks (SNNs), a family of algorithms employing computational primitives that mimic the biological behaviour of neurons and synapses, have emerged as a highly promising alternative. SNNs have been shown to substantially improve computational efficiency and latency, particularly when used in conjunction with time-varying signals and dedicated neuromorphic hardware [5]–[8].

Despite the promise of more efficient computing and the inherent suitability for time-dependent signals, SNNs often are not implemented in a manner that actually takes advantage of these features. Typically, when processing signals with spatiotemporal dependencies, these dependencies are encoded in the event representation prior

[1]Lammers is a MSc student in Aerospace Engineering, Delft University of Technology, Netherlands.

[2]Supervisor at Faculty of Aerospace Engineering, Delft University of Technology, Netherlands.

to entering the (spiking) network [9]–[13], permitting memoryless networks to perform such tasks. This method disregards, at least in part, the biological motivation underlying event-based perception [14], while also introducing latency in the communication between sensor and neural network. In order to realize a true neuromorphic pipeline, spike trains originating from an event-based sensor should be transferred as-is and one-by-one to the SNN, and temporal integration of signals should be left to the network. To accomplish this, it is necessary to investigate how to manipulate SNNs in order to extract spatio-temporal features of data streams accurately and efficiently.

In this article, we evaluate and compare three distinct ways of introducing memory into an SNN. These mechanisms include the inherent spiking neuronal dynamics (originating from leaks and thresholds), explicit recurrent connections and learnable spike propagation delays. In addition to SNNs, non-spiking equivalent networks are also trained to function as a reference (explained in more detail in Sec. III-D). Whereas a significant fraction of SNN research focusses on problems with limited temporal complexity like classification and segmentation, we deliberately target an alternative application with more temporal complexity by training our memory mechanisms to regress angular velocity from event-based data. In contrast to classification and segmentation, where the class can sometimes be determined on the basis of only a single input and the predicted value remains constant over time, regression on event-based input signals requires much more temporal integration as well as predicting a continuously changing value. We use a public dataset with real-world motion sequences [15] that have ground-truth angular velocity information, allowing supervised learning.

In summary, this article has two main contributions. First, we present a thorough comparison between different memory mechanisms in SNN in terms of performance, as well as an in depth analysis of network parameters including membrane voltage leaks, recurrent connection weights and delay values. Second, we report the first implementation of an SNN with learnable delays to successfully perform a relatively complex regression task.

II. Related Work

In the field of neuromorphic computation, spiking neural networks have become a popular research topic, and the literature offers a variety of architectures that can integrate information inside the network. Almost all mechanisms can be classified into one of three categories: (1) networks that exclusively utilize spiking neuronal dynamics, (2) networks with explicit recurrent connections, and (3) networks with propagation delays. Here, we will only discuss works where one of the previously mentioned mechanisms is optimized through training. In addition, with *spiking neuron dynamics* we denote that it concerns feedforward (i.e., without recurrency or delays) SNN architectures that *exclusively* rely on the dynamics of their spiking neurons for temporal information integration.

A. Spiking Neuron Dynamics

With the introduction of surrogate gradient backpropagation [16], neuronal parameters and synaptic connectivity could be optimized simultaneously. Optimizing neuronal parameters represented a significant departure from previous modelling standards, where the error signals were usually computed on the timing of spikes. The importance of optimizing spiking neuron parameters was highlighted in [17]. Perez-Nieves *et al.* found that increasing heterogeneity (providing each neuron with a unique membrane and synaptic time constant) in SNNs improved performance across a variety of tasks, but particularly for tasks with a more complex temporal structure.

A number of studies focus specifically on exploiting the inherent spiking neuronal dynamics to extract information from spatiotemporal dependencies in input signals. Due to the fact that only a small proportion of these works concentrate on regression tasks, we also discuss some other notable work focussing on other tasks. In [18], Ponghiran *et al.* demonstrate the utility of spiking dynamics for learning long sequences. By improving the inherent recurrence dynamics, they achieve a $2\times$ reduction in learnable parameters compared to using LSTMs or GRUs, while performance is similar. In another work, Pellegrini *et al.* [19] show that a feedforward network of dilated convolutional spiking layers can perform speech command recognition on par with (non-spiking) deep neural networks (DNNs), while exhibiting sparse activity. In contrast to the majority of works that apply SNNs to speech tasks, where non-trainable mechanisms are required to first encode the speech input features into a neural encoding, their approach applies the first convolution layer directly to real-valued speech features and is trainable just like the rest of the network. The authors in [20] focus on an application that appears perfectly suitable for SNNs due to the importance of minimal energy consumption. They train a feedforward SNN to accurately recognize human activity based on time series data from wearable sensors while reducing energy significantly. Finally, a work by Viale *et al.* is certainly noteworthy as they not only utilize spiking neuronal dynamics to detect vehicle lanes from event-based camera input, but also successfully implemented their SNNs on neuromorphic processor Loihi [21]. Their Loihi implementation demonstrates accuracy comparable to the SOTA while consuming as little as $1W$ of power.

One of the earliest (and only) works that exclusively exploits the intrinsic temporal dynamics of spiking neurons for regression in the continuous time domain is [22]. The authors trained a simple, strictly feedforward convolutional SNN to estimate angular velocity ego-motion using a generated event-camera dataset. They discovered that with spiking neuronal dynamics alone, angular velocity could be regressed with a reasonable degree of accuracy. Early works that tackle larger scale, dense visual regression tasks include [23, 24]. Rançon *et al.* [24] propose StereoSpike, a fully spiking deep neural network architecture for depth prediction, while Hagenaars *et al.* [23] train non-recurrent (as well as recurrent) SNN architectures to regress dense optical flow. Finally, the EVSNN model presented in [25] for event-based video reconstruction demonstrates for the first time that feedforward SNN can also perform large-scale regression tasks such as event-based video reconstruction. Compared to ANN-based models, this model obtains comparable performance on the IJRR [15], MVSEC [26], and HQF [12] datasets while consuming 19.36 times less energy.

B. Recurrent Connections

In addition to the inherent form of memory of spiking neurons, temporal integration can also be introduced explicitly through recurrent connections. Non-spiking recurrent neural networks (RNNs) were introduced in the late 1980s [27]–[29]. Aside from the neuron type, recurrent ANNs and recurrent SNNs (SRNNs) operate similarly. The only distinction is that during subsequent time steps, spikes are fed back into the spiking neurons of the same layer rather than floats. One of the earliest publications to effectively implement recurrency in an SNN in conjunction with backpropagation through time (BPTT) is [30]. More recent works such as [31] and [8, 32] for the first time successfully leverage surrogate gradients in order to train their recurrent SNNs, although they limited themselves to classification and recognition tasks. This observation is actually quite prevalent, as a large proportion of the most widely-read literary works focus on these types of tasks. [33]–[37].

Not many works use spiking neuron in combination with explicit recurrent connection. A notable early implementation of recurrent connections in a deep SNN in order to solve a complex continuous time regression problem however is [23]. Hagenaars *et al.* use spiking variants of two common optical flow estimation methods from events to regress optical flow without any temporal encoding in the input representation. They show that performance is comparable to equivalent non-spiking variants at an estimated $25\times$ increase in energy efficiency. In [38], the authors extend the classical LSTM formulation to the spiking domain. The performance of this SLSTM was analysed by employing a history-dependent isotropic hardening model in which various load vectors were accurately regressed. The spiking LSTM reduced energy consumption by $238\times$ during inference on the Loihi processor.

C. Delays

Numerous studies have demonstrated that synaptic delays are present in biological neurons and that they contribute to the richness of neuronal dynamics [39]–[41], so it is not surprising that research in the neuromorphic computing domain has laid emphasis on these mechanisms as well. In SNNs, the learning of delays is typically done via one of either two methods. We refer to these methods as *delay selection* and *delay shift*. The former is a mechanism in which a presynaptic and a postsynaptic neuron are connected via a connection consisting of sub-connections, each having a distinct delay and weight. By performing gradient descent on the weights, sub-connections with appropriate delays are strengthened, while those producing undesirable output spikes are reduced. Bohte *et al.* [42] who proposed the SpikeProp algorithm were the first to present a framework that achieves delay selection. Numerous researchers have utilized this algorithm to learn delays in a wide variety of small scale scenarios since [43]–[47]. However, delay selection has been applied to complex tasks as well, although not in a supervised learning context. Paredes-Vallés *et al.* [48], for example, used it to successfully regress optical flow from events in an unsupervised manner.

Alternatively, we have delay shift, a method that optimizes propagation delays via direct backpropagation of errors on delays. For this to work, during training larger segments of a time sequence must be entered into the SNN at a time. Only then there is an exact defined mathematical relation between points that bridges time steps in the input signal (achieved through convolving with a time shifted Dirac pulse). This is required in order to compute partial derivatives and realize error credit assignment. Important is that during inference this is not a requirement because once the delays are learned, sequential input is no problem and low latency inference can be realized. Shrestha and Orchard [49] presented a framework which accomplishes exactly that, which they called Spike Layer Error Reassignment (SLAYER). Their approach is capable of learning simultaneously the weights, leaks and axonal delay parameters in an SNN, and established a new SOTA on multiple tasks including image and audio challenges. Axonal delays can be viewed as a special case of synaptic delays, where all synapses downstream to neuron i share the same delay. In other words, all postsynaptic neurons of neuron i receive spikes at the same (delayed) moment in time. Only four other works learn axonal delays directly and are all implemented using the SLAYER framework. Sun et al. [50] integrate learnable axonal delays into the spiking neuron model with a static delay cap which they later extend to an adaptive training scheduler to adjust the delay cap [51]. In [52] axonal delays are learned without constraint. Each of these implementations have been limited to classification tasks. A fourth work also focusses on classification but is notable for a different reason. In [53], the SLAYER framework is modified such that downstream synapses no longer share the same delay, but each have their own delay value. Through learning only synaptic delays (i.e., no weight learning) in a feedforward SNN, they achieve performance comparable to the conventional training of synaptic weights.

III. Methodology

In this section, we describe our approach in detail. Subsection III-A discusses how the event stream is partitioned to an input representation. Then, in III-B, we describe the (spiking) neuron models used. Afterwards, the base neural network architecture that is used throughout this research is presented in subsection III-C. The different

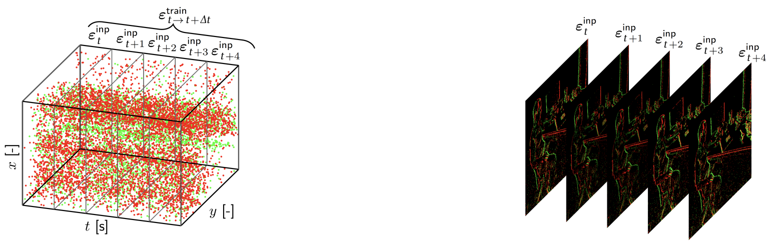

$\varepsilon^{\text{train}}_{t\to t+\Delta t}$
$\varepsilon^{\text{inp}}_{t}$
$\varepsilon^{\text{inp}}_{t+1}$
$\varepsilon^{\text{inp}}_{t+2}$
$\varepsilon^{\text{inp}}_{t+3}$
$\varepsilon^{\text{inp}}_{t+4}$
x [-]
t [s]
y [-]

Fig. 2: Left: Schematic representation of the output event stream of the event camera. Discrete partitions in the time dimension are created every Δt seconds. Right: The resulting frames after the events within each partition are accumulated. Images are a courtesy of [23].

memory mechanisms are introduced in subsection III-D. Finally, the methodology is concluded with an elaboration on the used dataset in subsection III-E and a description of the training procedure in subsection III-F.

A. Events to Input Representation

The sensor of an event camera consists of a grid of independent pixels that respond to changes in brightness. More precisely, they respond to variations in their log photocurrent $L = \log(I)$. An event $e_k = (\mathbf{x_k}, t_k, p_k)$ is generated at pixel $\mathbf{x_k} = (x_k, y_k)$ and time t_k immediately after the brightness change since the last event at that pixel,

$$\begin{aligned}\Delta L\left(\mathrm{x}_{\mathbf{k}}, t_k\right) &= L\left(\mathrm{x}_{\mathbf{k}}, t_k\right) - L\left(\mathrm{x}_{\mathbf{k}}, t_k - \Delta t_k\right)\\ &= p_k C,\end{aligned}$$

exceeds a predefined threshold $\pm C$, where $C > 0$, Δt_k is the time since the last event at the pixel in question and the polarity $p_k \in \{+1, -1\}$ is the sign of the brightness change [4].

Since the purpose of this study is to examine temporal integration capabilities inside SNNs, it is crucial that time information is not encoded in the input representation. This is however precisely what many learning-based models currently proposed for event-based regression do, allowing memoryless artificial neural networks (ANNs) to take advantage of time dependencies and to produce precise estimations.

Our goal is for spiking neural networks (SNNs) to receive spikes directly at event locations and extract temporal information by itself. Due to digital hardware limitations, we cannot enter individual events into the network as they may be only $\sim 1\,\mu\text{s}$ apart. Hence, we opt for a second-best alternative. As shown in Figure 2, we use a representation, inspired by [23], consisting solely of per-pixel and per-polarity event counts. This representation contains partitions of the event sequence that are consecutive and non-overlapping, with a length of 1 ms per partition. Additionally, the maximum count per pixel is limited to 1 to assure minimal temporal encoding.

B. Neuron Models

For our spiking neurons we use the widely used Leaky Integrate-and-Fire (LIF) neuron model or the Spike Response Model (SRM). The SRM is only used in the networks that learn axonal delays because the SLAYER framework [49] we use for learning axonal delays only features the SRM. For our non-spiking neurons we draw inspiration from [23].

LIF: Although the LIF neuron is not of the highest possible biological realism, its simplicity allows for efficient simulation, while offering enough computational versatility for most applications. The LIF model, models the biological neuron as a simple electrical circuit having a membrane potential U that is excited by an input current I. In discretized form the dynamics are described as:

$$I_i^{(l)}[n] = \sum_j W_{ij}^{(l)} S_j^{(l-1)}[n] + \sum_r W_{ir}^{(l)} S_r^{(l)}[n-1] \qquad (1)$$

$$U_i^{(l)}[n] = (1 - S_i^{(l)}[n-1])\alpha U_i^{(l)}[n-1] + (1-\alpha) I_i^{(l)}[n] \quad (2)$$

where subscripts j and r denote presynaptic neurons from the previous layer and current layer (in case of recurrency), respectively, i denotes postsynaptic neurons and superscript (l) denotes the layer index. The membrane leak is denoted by α, spikes by $S \in \{0, 1\}$ and W_{ij} and W_{ir} represent the weights of the feedforward and recurrent connections, respectively. Furthermore, the LIF neurons fire when their threshold ϑ is crossed from below according to

$$S_i^{(l)}[n] = \Theta\left(U_i^{(l)}[n] - \vartheta\right), \qquad (3)$$

where Θ represents the Heaviside function. We use a per neuron learnable threshold, as well as a hard reset mechanism [54], meaning that the membrane potential returns to zero after a spike.

As surrogate for the derivative of the spiking function we use the derivative of the inverse tangent $\sigma' = 1/(1 + \gamma x^2)$ [55], where γ defines the width of the surrogate gradient and x the difference between the membrane potential and the threshold $U - \theta$. We use $\gamma = 10$.

SRM: The networks that employ learnable axonal delays are build using the lava-dl implementation of SLAYER [49]. This platform uses the Spike Response Model (SRM) instead of the LIF model. In the SRM, the course of the membrane potential is described by

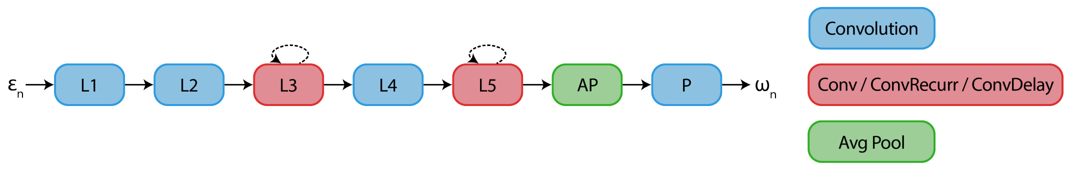


Fig. 3: Diagram of the base neural network architecture utilized in this study. The network receives events from an event camera (input size = 240 × 180), and outputs the rotation rates around the three axes x, y and z. All evaluated variants inherit from this architecture, differing only in the neuron model, convolutional recurrent or delay layers, or both.

TABLE I: Hyperparameters of the (spiking) neural networks.

	Conv 1	**Conv 2**	**Conv 3**	**Conv 4**	**Conv 5**	**Avg Pool**	**Pred**
Kernel Size	4×4	3×3	3×3	3×3	3×3	4×4	—
Channels	16	32	32	64	64	—	—
Stride	3	2	1	2	1	3	—
Padding	0	0	same	0	same	—	—

$$U_i^{(l)}(t) = \sum_j W_{ij}^{(l-1)} \left(\varepsilon * S_j\right)(t) + \left(\nu * S_i\right)(t) \tag{4}$$

where j denotes presynaptic neurons, i denotes postsynaptic neurons, $\varepsilon(t) = {}^{t}/\tau_s \exp\left(1 - {}^{t}/\tau_s\right)\Theta(t)$ is a filter kernel that describes the response to an incoming spike and $\nu(t) = -2\vartheta \exp\left(1 - {}^{t}/\tau_r\right)\Theta(t)$ is a kernel that describes the refractory response to an outgoing spike. The time constants τ_s and τ_r are learnable parameters. The lava-dl implementation of the SLAYER framework only allows for shared membrane leakage and axonal delay per convolutional channel.

The SLAYER framework uses a decaying exponential as surrogate gradient (SG) that is defined as $\sigma' = (1/\alpha)\exp(-\beta|x|)$. By setting $\alpha = 1$ and $\beta = 2$, the decaying exponential and the derivative of the inverse tangent of width $\gamma = 10$ become practically identical.

Non-spiking Neuron Model: For a fairer comparison between spiking and non-spiking neural networks, our non-spiking neurons receive a leaky state. We adopt the definition in [23], where the leaky artificial neuron is analogous to the dynamics of the spiking neuron, except that no reset mechanism is used and a non-spiking activation function is applied. By replacing the spiking activation with ReLu activation, the activation of any neuron becomes

$$y_i^k = \mathrm{ReLU}\left(\alpha y_i^{k-1} + (1-\alpha)\sum_j W_{ij}^{\mathrm{ff}} I_i^k\right), \tag{5}$$

where j and i denote presynaptic and postsynaptic neurons respectively, α is the neuron leak, k the time step, W^{ff} are the feedforward weights that are multiplied with the input current I, and ReLU the rectifying linear unit.

C. Architecture

To ensure a fair comparison between memory mechanisms, we designed a base neural network architecture of which we create variants having either one or multiple memory mechanisms available to it. The base architecture ConvANN is a feedforward convolutional neural network inspired by the network proposed in [22]. The base network, illustrated in Figure 3, consists of 5 convolutional layers, followed by an average pooling layer and a fully connected prediction layer that outputs 3 values, one for the rotation rate around each of the Cartesian axes. The blue blocks in Figure 3 perform spatial downsampling, while doubling the number of channels in each layer. The red block perform spatial downsampling but keep the number of channels the same to allow for easy implementation of recurrent connections. Table I provides the exact per layer hyperparameters.

In the most basic (non-spiking) form, the network does not possess any integration capabilities. Therefore, this variant should not be able to regress angular velocities and by consequence is suitable as a baseline to check that no temporal information is encoded into the input. From there the base architecture can be augmented with spiking neurons in all layers, creating a convolutional spiking neural network ConvSNN. In addition, the network can be augmented further by introducing learnable axonal delays or explicit recurrent connections in layers 3 and 5 (red blocks in Fig. 3).

D. Memory Mechanisms

We evaluate three different memory mechanisms. The first mechanism is the spiking neuronal dynamics. Due to having an internal state that is not just dependent on present synaptic inputs, but also on previous states, we refer to leaks as *implicit recurrency*. Implicit recurrency introduces 'memory' into the SNN because it integrates previous synaptic input. The parameters α and θ in equations 2 and 3, ultimately determine the shape of membrane potential and thus the spiking activity [57].

In addition to introducing implicit recurrency in the form of spiking neurons, it is also possible to introduce temporal dependencies *explicitly* by adding recurrent connections. A (spiking) neural network with recurrent connections introduces memory in the form of internal hidden states

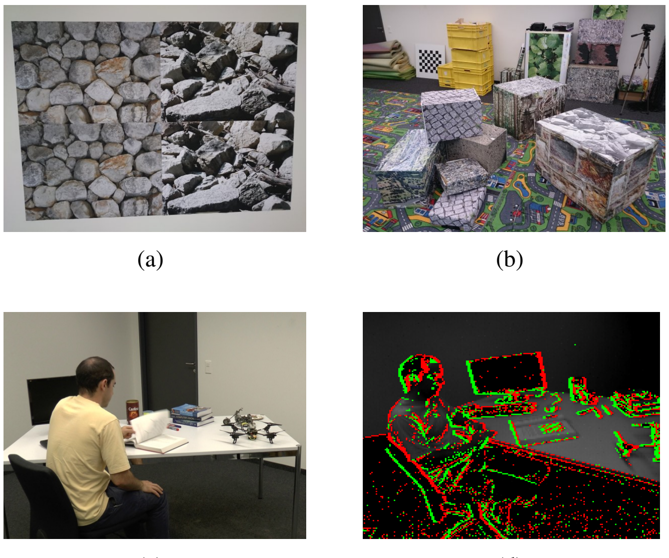


Fig. 4: (a) The `poster` scene (b) The `boxes` scene (c) The `dynamic` scene (d) The `dynamic` scene with events overlaid [56].

which are updated while time-stepped input feeds in. The difference with implicit recurrency is that output spikes not just feed back into the same neuron, but feed back into all the neurons in the layer.

The third integration mechanism is the propagation delay. For delay shift we learn axonal delays, meaning that all presynaptic connections share the same delay (not to be confused with a synaptic delay, where each synapse has its own propagation delay). For delay selection we learn the weights of synaptic delays. The propagation delays introduce memory into the SNN by letting inputs of different time steps catch up within the network.

E. ECD Datasets

To assess the accuracy of the memory mechanisms, we train and test them on three distinct sequences: `poster`, `boxes`, and `dynamic rotation` [15]. The sequences `poster` and `boxes` are used for training, while the sequence `dynamic` is used for testing. The `poster` scene consists of a textured wall poster, the `boxes` scene of crates on a carpet, and the `dynamic` scene of a desk containing objects and a person moving them. In addition to the event stream, all sequences include ground truth pose measurements from a motion capture system (mocap), as well as gyroscope data from the IMU of the Dynamic and Active-pixel Vision Sensor (DAVIS) [58]. The motion capture system has a sampling frequency of 200 Hz and the IMU of 1 kHz. The sequences were manually recorded. Each sequence is one minute long and comprises approximately 150 million events. Each sequence begins with rotations around each camera axis, followed by rotations in all three degrees of freedom. In addition, as the sequence progresses, the motion accelerates.

F. Training Procedure

Our framework is implemented in PyTorch. All networks, except for those learning axonal delays, are trained directly using PyTorch Autograd with an SG. The axonal delay networks are implemented utilizing the SLAYER framework, which is based on PyTorch and indirectly employs the Autograd functionality.

For training, we use the Adam optimizer to jointly update the synaptic weights, membrane leaks and neuron thresholds with a constant learning rate of 0.0005 and a batch size of 8. Networks are trained until convergence. Initialization of the neuron leaks is done by applying the sigmoid $\alpha = \frac{1}{1+\exp(-a)}$ to values a, drawn from a normal distribution centred around -3. This results in small initial leak and thus in little initial 'memory', encouraging the spiking neurons to learn such behaviour instead of giving it from the start. In addition, during every forward pass the leaks are squished between 0 and 1 using the sigmoid function in order to prevent instability [55]. The thresholds θ are clamped to $[0.05, \rightarrow)$ as implemented in [23].

IV. Results and Discussion

In this section, we evaluate the performance and the inner workings of the memory mechanisms. Training is done on the `poster_rotation` and `boxes_rotation` ECD sequences [15], while performance is evaluated on the `dynamic_rotation` dataset (the most diverse and realistic scene). In addition to comparing our networks with each other, we also compare performance with the event-based SNN architecture for angular velocity regression in [22] and the ego-motion estimation by contrast maximization algorithm proposed in [4].

A. Performance Evaluation

First, based on Table II we conclude that our input representation indeed does not encode any temporal dependencies (in a single frame), as the memoryless ConvANN does not perform better than a naive mean prediction. As for the other networks, all have converged. Our spiking networks show similar or better accuracy to the SNN from [22] even though that network is directly trained on the `dynamic_rotation` dataset, while our networks have never seen the dataset before testing. The non-spiking equivalent networks achieve smaller error although never having seen the test data before. The highest accuracy is clearly achieved by the delay selection networks. This is expected because the network has more learnable parameters than the others (not one but six synapses with different delays for all neurons in the 3^{rd} and 5^{th} layer). The networks having explicit recurrent connections (ConvSRNN and LeakyConvRNN) achieve the second-highest accuracy. Their performance is followed by the networks employing delays (DelaySNN and DelayANN). The feedforward ConvSNN having just neuronal dynamics to achieve temporal integration performs least well.

Secondly, comparing the performance of the SNNs with their non-spiking equivalents, we conclude that the SNNs

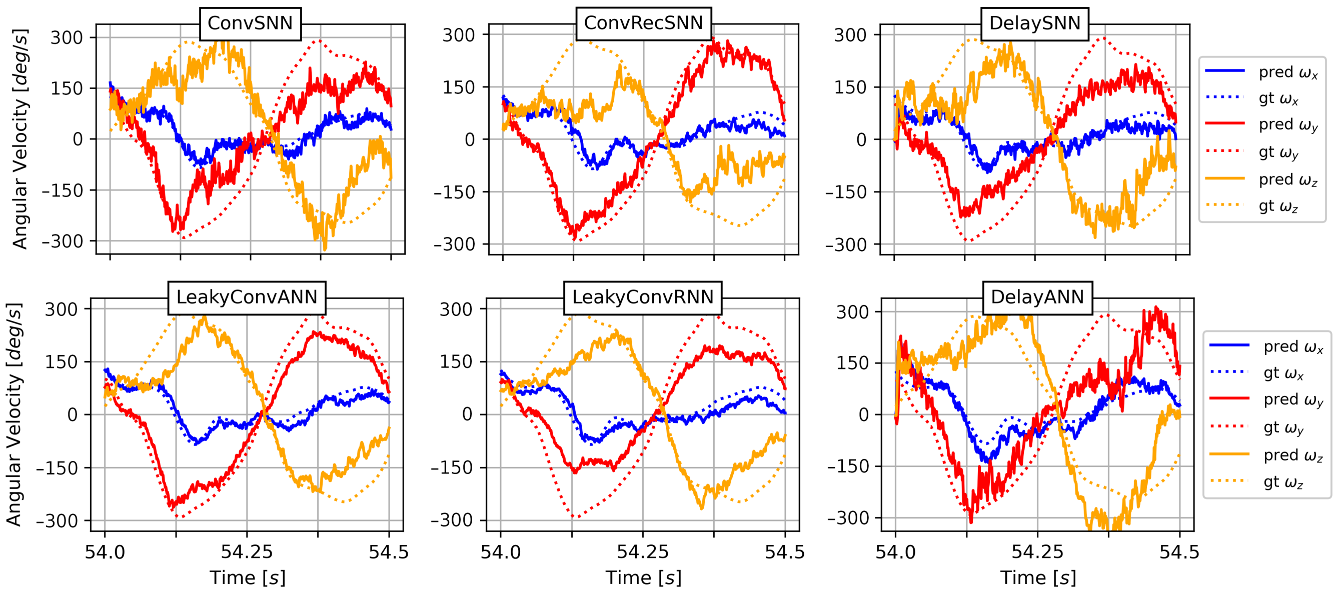


Fig. 5: Qualitative results of the SNN (top row) and ANN (bottom row) on one of the most challenging parts of the `dynamic` test sequence, where large pan and roll motion occur simultaneously.

TABLE II: Performance of the proposed ANN and SNN architectures on the `dynamic_rotation` dataset, as well as state-of-the-art methods. The highest accuracies are in bold.

	`dynamic_rotation`	
	$\varepsilon_{\mathrm{rms}}$ [deg/s]	$\varepsilon_{\mathrm{rel}}$ [−]
Naive mean	126.3	—
ConvANN	126.4	1
ConvRNN	48.0	0.42
ConvSNN	63.0	0.55
LeakyConvANN	50.9	0.44
ConvSRNN	56.2	0.50
LeakyConvRNN	44.3	0.39
DelayShiftSNN	58.2	0.51
DelayShiftANN	67.0	0.58
DelaySelectSNN	**35.8**	**0.29**
DelaySelectANN	32.3	0.27
SNN [22]	53.8	—
CM [59]	9.68	—

in all three cases perform slightly worse. This observation is consistent with previous work [23, 31, 60]. The reason that spiking seems to perform consistently less well than non-spiking highly likely has to do with the neural activation function right before the prediction layer. Because spiking activation is binary, the solution space of the SNN is finite. In fact the SNN can output only N^2_{pred} distinct values, as it can switch each prediction layer weight either 'on' or 'off. The ANN in contrast has an infinite solution space as the continuous activation allows for an infinite combination of prediction layer weights. This fundamental difference between spiking and non-spiking likely becomes more apparent when regressing continuous numerical values than for example when attempting a discrete task like classification. This difference between spiking and non-spiking is also clearly visible in the qualitative results in Figure 5, in which the spiking networks (upper row) give much noisier output. To test whether the prediction layer is responsible for the noisy output, we also trained an SNN with a leaky prediction layer. Theoretically, this could result in a smoother output, as fractions of preceding outputs are also included in new outputs. The evaluation demonstrated however that the leaks of the output neurons do not grow larger than $\alpha = 0.3$, which is not large enough to notice a substantial difference in smoothness. Initializing the leaks of the prediction layer near $\alpha = 0.8$ also was not helpful as it actually resulted in the leaks decreasing during optimization, as well as a serious deterioration in performance. Further details can be found in Appendix V.

B. *Analysis of Neural Parameters*

As explained in section III-F, we initialize the leaks α of our spiking neurons close to 0, corresponding to little to none initial memory thereby challenging the spiking neurons to learn such behaviour instead of giving it from the start (see Equation 2). To investigate whether this has indeed happened, histograms of the leaks per layer before training and after training are plotted (Fig. 6). The top row in Figure 6, corresponding to the leaks of the (feedforward) ConvSNN, clearly shows that the leaks initially (blue bins) are distributed around 0 in each layer. After training, we see that the ConvSNN, which is completely dependent on spiking neuronal dynamics for information integration, has indeed transformed a large part of its decay values towards

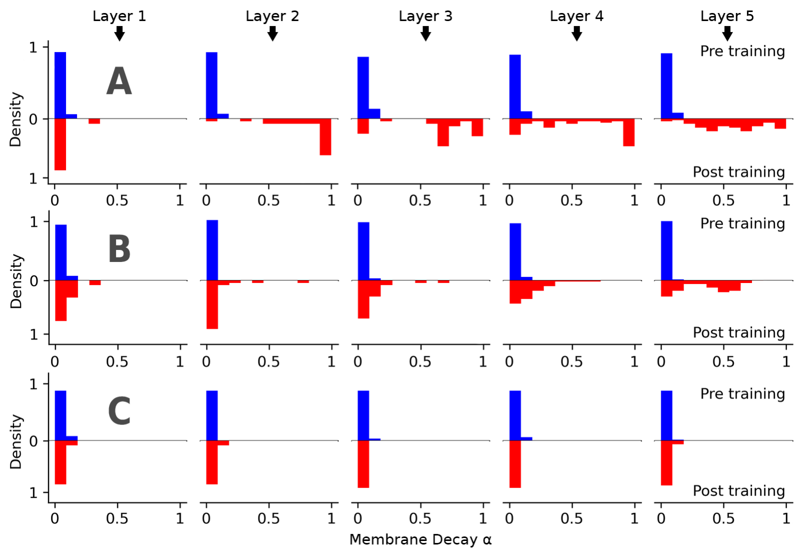


Fig. 6: Histograms of the distribution of the membrane decay parameter α per layer, before (blue) and after training (red). **A** are the distributions of the decay of the ConvSNN, **B** the distributions of the LeakyConvANN and **C** the distributions of the ConvRecSNN.

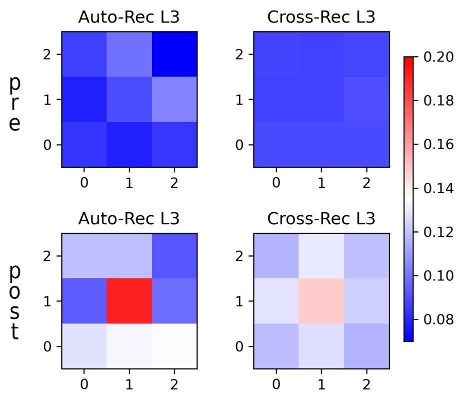


Fig. 7: Top row: average size of the convolutional kernels before training. On the left kernels that connect to neurons to themselves (auto-recurrent connections). On the right kernels that only connect neurons to other neurons (cross-recurrent connections). The bottom row: the average of the same kernels as on top, but after training.

1, as the red bins are no longer distributed around zero. Especially layer 2 (the second plot from the left) appears to be responsible for a lot of integration, based on the large peak close to 1. In later layers, decay rates are distributed more uniformly, some choosing for almost no integration while others a lot. This makes sense as angular velocity estimation is only possible by concentrating input information of different time steps to the same point in time in the network. This requires different spiking neuron timescales.

If we examine the distribution of the leaks of LeakyConvANN (middle row of Figure 6), the non-spiking counterpart of ConvSNN, then we notice that the distribution looks somewhat different. The α values do not approach 1, but we see that for the non-spiking variant the maximum α values approach ≈ 0.5. This difference can be explained by the difference in neural activation. The non-spiking neurons do not possess a reset mechanism to 'clear' their memory after a spike is transmitted, whereas spiking neurons do. To prevent a neuron from becoming worthless after a number of inputs because it is blocked with an accumulation of old inputs, the leaks do not exceed about 0.5. Our hypothesis that the lack of a reset mechanism forces this distribution of leaks is tested by training the leaky ANN with a thresholded ReLU and a reset mechanism (details in Appendix V). With a reset mechanism, the α values of the ANN do indeed move towards 1 and it even turns out that this neuron model achieves the highest accuracy of all networks: $\varepsilon_{\text{rmse}} = 40.5$deg/s (not considering the delay selection networks because they have more learnable parameter). This highlights nicely that forgetting is an important aspect of memory.

C. *Membrane Voltage Decay with Recurrency*

Next we investigate the composition of the parameters in the SNN with explicit recurrent connections (ConvSRNN) after training. From Table II we deduce that the accuracy of the SRNN is greater than that of the feedforward SNN (56.2 vs 63 deg/s), indicating that the recurrent connections augment temporal integration capabilities. The decay rates of the membrane potentials of the spiking neurons are visualized in the bottom row of Figure 6. What these histograms show is that nothing changes at all to the learnable neuronal parameters during the training process. The recurrent connections did not augment the temporal integration capabilities of the SNN, but completely substituted the spiking neuronal dynamics. Even layers without recurrent connections (L1, L2 and L4) do not integrate any input spikes. In this form, the network is better described as a binary activation network with recurrent connections than as a spiking neural network.

To gain insight into how the explicit recurrent connections take over the job of the spiking neuronal dynamics, we analyse the recurrent weight kernels. Similar behaviour to spiking neural dynamics, would mean that the recurrent weights connecting the spiking neurons to itself have a relatively large weight, while the other weights are relatively small. Figure 7 shows the mean weight size of the weights connecting neurons to themselves (which we call *auto-recurrent connections*) and of weights connecting to others (which we call *cross-recurrent connections*). The top two plots correspond to pre-training, while the bottom two corresponds to post-training. From the fact that the top two plots are approximately uniformly blue, we deduce that all weights are (as expected) uniformly initiated. From the fact that the middle weight in the plot on the bottom left is the darkest red, we deduce that it is the largest on average after training. The weights of auto-recurrent connections are therefore larger than those of cross-recurrent connections. This is not to say that feeding back output to other neurons in the same layer does not happen. After all, the other weights are also non-zero. Yet, it seems that by feeding neural output relatively strongly back to themselves, the neurons no longer need spiking neuronal dynamics. The membrane voltage decay are simply kept near 0 and the auto-recurrent connections build a state as a substitute.

To test whether, when using recurrent connections, the SNN does not want to use the dynamics of the spiking neurons at all, or if this only occurs when α is initialized near 0, we also trained the network with uniform initialization between 0 and 1 of decay (see Appendix V for details). We find that learning becomes extremely unstable with uniform initialization and that there is a significant reduction in performance. Despite the fact that there are further possibilities to tinker with the initialization of the leaks, thresholds and weights, with which perhaps a little better performance can be achieved, this clearly indicates that SNNs do not easily combine explicit recurrent connections with spiking neuronal dynamics.

D. Axonal Delay vs. Membrane Leak

To investigate what happens to the delays and leaks during training and whether there is a correlation between the two, scatter plots of these two parameters were created. The plots are depicted in Figure 8. The blue data points represent the conditions prior to training, while the red data points represent the situation after training. What is immediately noticeable is that the leaks hardly change during training when we are learning delays, just like we have seen with recurrent connections. In fact, the leaks grow smaller through learning, which indicates even less input current is integrated onto the membrane potential after learning than before. The axonal delays in contrast, have changed significantly during training: from uniform initialization between 0 and 1 millisecond to a maximum of approximately 6 ms after training. Not all delays have increased, however. Some delays actually return to 0. This is in fact not very surprising because eventually it's important to have input elements arriving at different times to catch up with one another inside the network, in order to achieve temporal integration of information. This requires some delays to be short and others to be long. Lastly, comparing the delays in layer 3 with the delays in layer 5, we notice that the delays get slightly larger deeper in the network. This indicates that the network prefers to focus more on extracting spatial features in shallower layers to then integrate the feature maps of different input time steps deeper in the network.

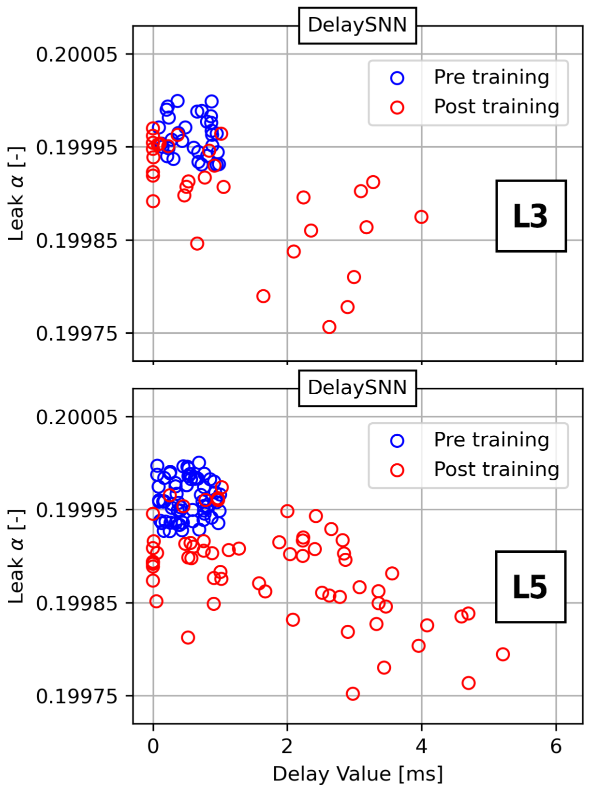


Fig. 8: Scatter plots of the delays vs the leaks before training (blue) and after training (red). The upper plot corresponds to the delays and leaks layer 3 (L3), while the bottom plot corresponds to the delays and leaks in layer 5 (L5).

V. Conclusions

In this paper, we evaluated the performance of three different memory mechanisms for SNNs on an angular velocity regression task given events from an event camera. The evaluated mechanisms include spiking neuronal dynamics (implicit recurrency), recurrent connections (explicit recurrency) and propagation delays. We find that both explicit recurrency and delays improve the prediction accuracy of the SNN, compared to having just spiking neuronal dynamics. Analysis of the inner workings of the networks shows that the threshold and reset mechanism of spiking neurons play an important role in allowing longer neuron timescales (lower membrane leak). Forgetting (at the right time) turns out to play an important role in memory. Additionally, it becomes apparent that optimizing an SNN with explicit recurrent connections or learnable delays does not lead to the formation of robust spiking neuronal dynamics. In fact, spiking neuronal dynamics are largely ignored, as after optimization virtually no input current is integrated onto the membrane potential in these cases. Instead, we consistently find that a recurrent SNN prefers to build a state solely with the explicit recurrent connections, while an SNN with delays prefers to just use the delays. Therefore, our SNNs with explicit recurrent connections and delays are in fact better described as binary activated RNNs and ANNs, respectively.

References

[1] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A Survey of Neuromorphic Computing

and Neural Networks in Hardware," May 2017.

[2] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, P. Date, and B. Kay, "Opportunities for neuromorphic computing algorithms and applications," *Nature Computational Science*, vol. 2, pp. 10–19, Jan. 2022.

[3] M.-H. Tayarani-Najaran and M. Schmuker, "Event-Based Sensing and Signal Processing in the Visual, Auditory, and Olfactory Domain: A Review," *Frontiers in Neural Circuits*, vol. 15, p. 610446, 2021.

[4] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, "Event-based Vision: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, pp. 154–180, Jan. 2019.

[5] M. Davies, A. Wild, G. Orchard, Y. Sandamirskaya, G. A. F. Guerra, P. Joshi, P. Plank, and S. R. Risbud, "Advancing Neuromorphic Computing With Loihi: A Survey of Results and Outlook," *Proceedings of the IEEE*, vol. 109, pp. 911–934, May 2021.

[6] P. Blouw, X. Choo, E. Hunsberger, and C. Eliasmith, "Benchmarking Keyword Spotting Efficiency on Neuromorphic Hardware," Apr. 2019.

[7] G. Tang, A. Shah, and K. P. Michmizos, "Spiking Neural Network on Neuromorphic Hardware for Energy-Efficient Unidimensional SLAM," Sept. 2019.

[8] B. Yin, F. Corradi, and S. M. Bohté, "Effective and Efficient Computation with Multiple-timescale Spiking Recurrent Neural Networks," June 2020.

[9] C. Lee, A. K. Kosta, A. Z. Zhu, K. Chaney, K. Daniilidis, and K. Roy, "Spike-FlowNet: Event-based Optical Flow Estimation with Energy-Efficient Hybrid Neural Networks," Sept. 2020.

[10] C. Lee, A. K. Kosta, and K. Roy, "Fusion-FlowNet: Energy-Efficient Optical Flow Estimation using Sensor Fusion and Deep Fused Spiking-Analog Network Architectures," Mar. 2021.

[11] F. Paredes-Vallés and G. C. H. E. de Croon, "Back to Event Basics: Self-Supervised Learning of Image Reconstruction for Event Cameras via Photometric Constancy," Apr. 2021.

[12] T. Stoffregen, C. Scheerlinck, D. Scaramuzza, T. Drummond, N. Barnes, L. Kleeman, and R. Mahony, "Reducing the Sim-to-Real Gap for Event Cameras," in *Computer Vision – ECCV 2020* (A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, eds.), Lecture Notes in Computer Science, (Cham), pp. 534–549, Springer International Publishing, 2020.

[13] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis, "EV-FlowNet: Self-Supervised Optical Flow Estimation for Event-based Cameras," in *Robotics: Science and Systems XIV*, June 2018.

[14] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbruck, "Retinomorphic Event-Based Vision Sensors: Bioinspired Cameras With Spiking Output," *Proceedings of the IEEE*, vol. 102, pp. 1470–1484, Oct. 2014.

[15] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza, "The Event-Camera Dataset and Simulator: Event-based Data for Pose Estimation, Visual Odometry, and SLAM," *The International Journal of Robotics Research*, vol. 36, pp. 142–149, Feb. 2017.

[16] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks," *IEEE Signal Processing Magazine*, vol. 36, pp. 51–63, Nov. 2019.

[17] N. Perez-Nieves, V. C. H. Leung, P. L. Dragotti, and D. F. M. Goodman, "Neural heterogeneity promotes robust learning," *Nature Communications*, vol. 12, pp. 1–9, Oct. 2021.

[18] W. Ponghiran and K. Roy, "Spiking Neural Networks with Improved Inherent Recurrence Dynamics for Sequential Learning," Sept. 2021.

[19] T. Pellegrini, R. Zimmer, and T. Masquelier, "Low-Activity Supervised Convolutional Spiking Neural Networks Applied to Speech Commands Recognition," in *2021 IEEE Spoken Language Technology Workshop (SLT)*, pp. 97–103, Jan. 2021.

[20] Y. Li, R. Yin, H. Park, Y. Kim, and P. Panda, "Wearable-based Human Activity Recognition with Spatio-Temporal Spiking Neural Networks," Nov. 2022.

[21] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, vol. 38, pp. 82–99, Jan. 2018.

[22] M. Gehrig, S. B. Shrestha, D. Mouritzen, and D. Scaramuzza, "Event-Based Angular Velocity Regression with Spiking Networks," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4195–4202, May 2020.

[23] J. Hagenaars, F. Paredes-Vallés, and G. de Croon, "Self-Supervised Learning of Event-Based Optical Flow with Spiking Neural Networks," Tech. Rep. arXiv:2106.01862, arXiv, Oct. 2021.

[24] U. Rançon, J. Cuadrado-Anibarro, B. R. Cottereau, and T. Masquelier, "StereoSpike: Depth Learning With a Spiking Neural Network," *IEEE Access*, vol. 10, pp. 127428–127439, 2022.

[25] L. Zhu, X. Wang, Y. Chang, J. Li, T. Huang, and Y. Tian, "Event-based Video Reconstruction via Potential-assisted Spiking Neural Network," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3584–3594, June 2022.

[26] A. Z. Zhu, D. Thakur, T. Ozaslan, B. Pfrommer, V. Kumar, and K. Daniilidis, "The Multi Vehicle Stereo Event Camera Dataset: An Event Camera Dataset for 3D Perception," *IEEE Robotics and Automation Letters*, vol. 3, pp. 2032–2039, July 2018.

[27] J. L. Elman, "Finding Structure in Time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[28] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986.

[29] M. I. Jordan, "Serial order: a parallel distributed processing approach. Technical report, June 1985-March 1986," Tech. Rep. AD-A-173989/5/XAB; ICS-8604, California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science, May 1986.

[30] D. Huh and T. J. Sejnowski, "Gradient Descent for Spiking Neural Networks," June 2017.

[31] Y. Xing, G. Di Caterina, and J. Soraghan, "A New Spiking Convolutional Recurrent Neural Network (SCRNN) With Applications to Event-Based Hand Gesture Recognition," *Frontiers in Neuroscience*, vol. 14, 2020.

[32] B. Yin, F. Corradi, and S. M. Bohté, "Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks," *Nature Machine Intelligence*, vol. 3, pp. 905–913, Oct. 2021.

[33] W. Wang, S. Hao, Y. Wei, S. Xiao, J. Feng, and N. Sebe, "Temporal Spiking Recurrent Neural Network for Action Recognition," *IEEE Access*, vol. 7, pp. 117165–117175, 2019.

[34] W. Zhang and P. Li, "Spike-Train Level Backpropagation for Training Deep Recurrent Spiking Neural Networks," Nov. 2019.

[35] Y. Ma, E. Donati, B. Chen, P. Ren, N. Zheng, and G. Indiveri, "Neuromorphic Implementation of a Recurrent Neural Network for EMG Classification," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 69–73, Aug. 2020.

[36] A. Bittar and P. N. Garner, "A surrogate gradient spiking baseline for speech command recognition," *Frontiers in Neuroscience*, vol. 16, 2022.

[37] A. Bittar and P. N. Garner, "Surrogate Gradient Spiking Neural Networks as Encoders for Large Vocabulary Continuous Speech Recognition," Feb. 2023.

[38] A. Henkes, J. K. Eshraghian, and H. Wessels, "Spiking neural networks for nonlinear regression," Oct. 2022.

[39] B. Katz and R. Miledi, "The measurement of synaptic delay, and the time course of acetylcholine release at the neuromuscular junction," *Proceedings of the Royal Society of London. Series B. Biological Sciences*, vol. 161, pp. 483–495, Feb. 1965.

[40] C. Wang, B. G. Cleland, and W. Burke, "Synaptic delay in the lateral geniculate nucleus of the cat," *Brain Research*, vol. 343, p. 236, Sept. 1985.

[41] F. Minneci, R. T. Kanichay, and R. A. Silver, "Estimation of the time course of neurotransmitter release at central synapses from the first latency of postsynaptic currents," *Journal of Neuroscience Methods*, vol. 205, pp. 49–64, Mar. 2012.

[42] S. M. Bohte, J. N. Kok, and H. La Poutré, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, pp. 17–37, Oct. 2002.

[43] S. Ghosh-Dastidar and H. Adeli, "Improved spiking neural networks for EEG classification and epilepsy and seizure detection," *Integrated Computer-Aided Engineering*, vol. 14, pp. 187–212, Jan. 2007.

[44] S. McKennoch, D. Liu, and L. Bushnell, "Fast Modifications of the SpikeProp Algorithm," in *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pp. 3970–3977, July 2006.

[45] S. B. Shrestha and Q. Song, "Adaptive learning rate of SpikeProp based on weight convergence analysis," *Neural Networks: The Official Journal of the International Neural Network Society*, vol. 63, pp. 185–198, Mar. 2015.

[46] I. Sporea and A. Grüning, "Supervised Learning in Multilayer Spiking Neural Networks," *Neural Computation*, vol. 25, pp. 473–509, Feb. 2013.

[47] Y. Xu, X. Zeng, L. Han, and J. Yang, "A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks," *Neural Networks: The Official Journal of the International Neural Network Society*, vol. 43, pp. 99–113, July 2013.

[48] F. Paredes-Vallés, K. Y. W. Scheper, and G. C. H. E. de Croon, "Unsupervised Learning of a Hierarchical Spiking Neural Network for Optical Flow Estimation: From Events to Global Motion Perception," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, pp. 2051–2064, Aug. 2020. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[49] S. B. Shrestha and G. Orchard, "SLAYER: Spike Layer Error Reassignment in Time," Tech. Rep. arXiv:1810.08646, arXiv, Sept. 2018.

[50] P. Sun, L. Zhu, and D. Botteldooren, "Axonal Delay As a Short-Term Memory for Feed Forward Deep Spiking Neural Networks," Tech. Rep. arXiv:2205.02115, arXiv, Apr. 2022.

[51] P. Sun, E. Eqlimi, Y. Chua, P. Devos, and D. Botteldooren, "Adaptive Axonal Delays in feedforward spiking neural networks for accurate spoken word recognition," Feb. 2023.

[52] S. B. Shrestha, L. Zhu, and P. Sun, "Spikemax: Spike-based Loss Methods for Classification," in *2022 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7, July 2022.

[53] E. W. Grappolini and A. Subramoney, "Beyond Weights: Deep learning in Spiking Neural Networks with pure synaptic-delay training," June 2023.

[54] E. Ledinauskas, J. Ruseckas, A. Juršėnas, and G. Buračas, "Training Deep Spiking Neural Networks," June 2020.

[55] W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang, and Y. Tian, "Incorporating Learnable Membrane Time Constant To Enhance Learning of Spiking Neural Networks," pp. 2661–2671, 2021.

[56] S. Lee and H. J. Kim, "Low-Latency and Scene-Robust Optical Flow Stream and Angular Velocity Estimation," *IEEE Access*, vol. 9, pp. 155988–155997, 2021.

[57] J. Göltz, L. Kriener, A. Baumbach, S. Billaudelle, O. Breitwieser, B. Cramer, D. Dold, A. F. Kungl, W. Senn, J. Schemmel, K. Meier, and M. A. Petrovici, "Fast and energy-efficient neuromorphic deep learning with first-spike times," May 2021.

[58] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, "A 240 × 180 130 dB 3 µs Latency Global Shutter Spatiotemporal Vision Sensor," *IEEE Journal of Solid-State Circuits*, vol. 49, pp. 2333–2341, Oct. 2014.

[59] G. Gallego and D. Scaramuzza, "Accurate Angular Velocity Estimation With an Event Camera," *IEEE Robotics and Automation Letters*, vol. 2, pp. 632–639, Apr. 2017.

[60] A. Kugele, T. Pfeil, M. Pfeiffer, and E. Chicca, "Efficient Processing of Spatio-Temporal Data Streams With Spiking Neural Networks," *Frontiers in Neuroscience*, vol. 14, 2020.

A Ablation Study Influence Reset Mechanism

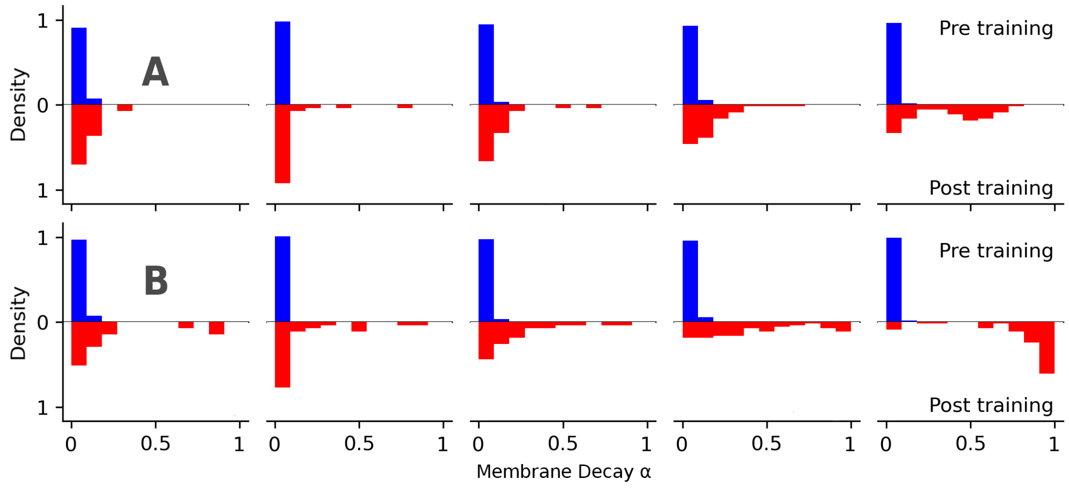


Fig. 9: **A** Distribution of the membrane potential decays in the LeakyConvANN per layer, after training. **B** Distribution of the membrane potential decays per layer, after training when a thresholded ReLu with a reset mechanism is added.

To test our hypothesis that the decays in the non-spiking variant of the ConvSNN do not come close to $\alpha = 1$ because it does not have a threshold with reset mechanism, we train a variant of the LeakyConvANN with a thresholded ReLu and reset. The decay distribution per layer without thresholded or reset are shown in **A** in Figure 9. The decay distribution per layer with a thresholded ReLu and reset are shown in **B** in Figure 9. It is very clear that with a thresholded ReLu and reset the decay values do go towards 1 during training because in the bottom figure red bins are located near $\alpha = 1$. What is also very interesting is that with a thresholded ReLu and reset mechanism the performance of the LeakyConvANN increases considerably: 40.5 deg/s with vs. 50.9 deg/s without. This highlights elegantly that forgetting is an important aspect of memory.

B Ablation Study Decay Initialization Recurrent SNN

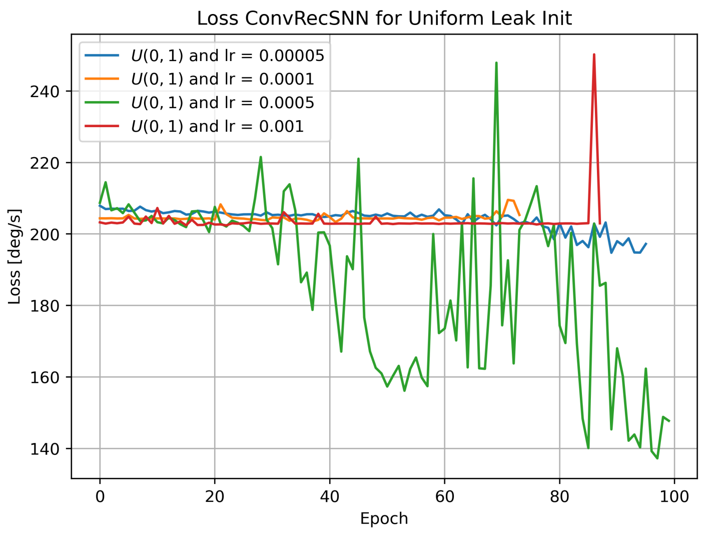


Fig. 10: Loss curves for uniform initialization of the membrane potential decay rates for different learning rates.

To determine if, when using recurrent connections, the network does not want to use the dynamics of the spiking neurons at all, or if this only occurs when alpha is initialized near zero, we also run the network with uniform initialization of decays between 0 and 1. This shows that training is becoming increasingly unstable. All trials except for the green curve stop training early because of numerical stability issues arising. The green run (lr = 0.0005) does not come close to the performance of decay initialization near 0.

C Ablation Study Prediction Layer in SNN

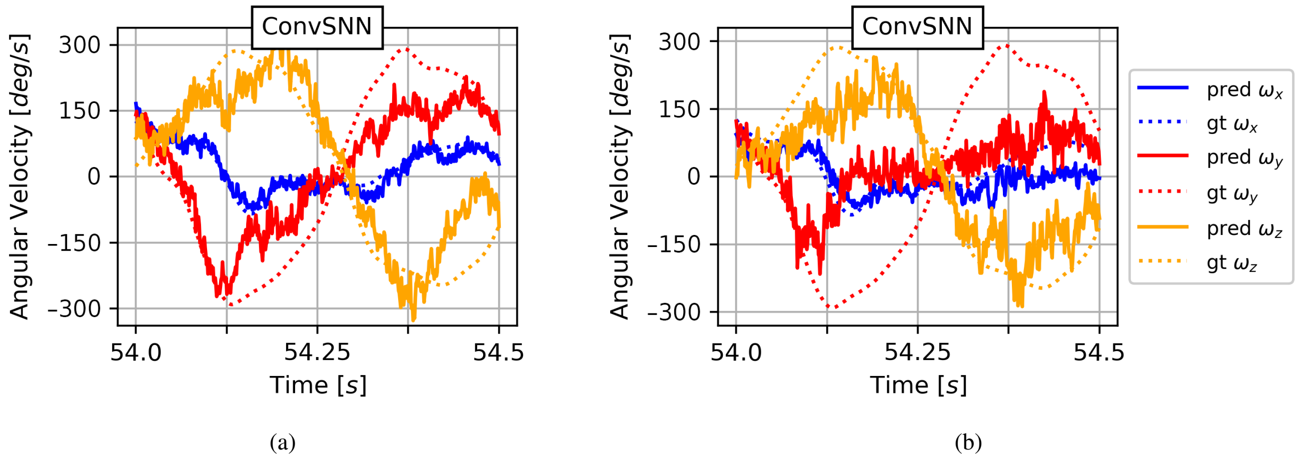


Fig. 11: (a) Qualitative results of ConvSNN with a regular, stateless prediction layer. (b) Qualitative results of ConvSNN with a leaky prediction layer.

It became apparent from the qualitative results that the predictions of the spiking networks are noisier than those of the non-spiking networks. It is believed that this is due to the fact that spiking networks have fewer possible outputs than non-spiking networks because they can only switch prediction weight ’on’ or ’off’ by spiking or not. A float output cannot be used to select a fraction of the prediction layer weight. We train an SNN with a ‘leaky’ prediction layer to determine if this improves the smoothness of the angular velocity prediction. The result is depicted in Figure 11. The left plot has not used a leaky prediction layer, the right one has. Initializing the prediction layer leaks close to 0 results in leaks no larger than $\alpha = 0.3$ (remember $\alpha \to 1$ corresponds to more memory), which based on the noisy signal in the right plot is clearly not enough to produce a smoother angular velocity estimation. Initialization of the prediction layer leaks much closer to 1 also does not improve the smoothness as it deteriorates performance very substantially.

II

Literature Study

2 Neuromorphic Computing

Developing a machine that can process information faster than humans has been a major driver in computing for decades, and the Von Neumann architecture has become the undisputed standard for this type of machine. Comparisons between this architecture and the human brain, however, reveal major discrepancies in organisational structure, power requirements and processing capability. This raises the question whether it is possible to create alternative architectures based on neurological concepts that are similar or even superior to a biological brain.

The first chapter of this literature review is devoted to the scientific advancements in the field of neuromorphic computing. Neuromorphic computing is, simply said, a subfield of computer science that focuses on attempting to compute in a manner that closely resembles the biological way of interpreting and processing information. In order to clearly describe all aspects of neuromorphic computing, the subject is broken down into three components: event-based sensing, event-based signal processing and neuromorphic hardware. The chapter begins with a brief introduction to neuromorphic computing. After that, in section 2.2 bio-inspired (or event-based) sensing methods are discussed. Section 2.3 addresses methods for processing data streams acquired by event-based sensors. Finally, section 2.4 concludes with a review of some neuromorphic hardware implementations, specifically hardware designed to facilitate spiking neural networks.

2.1 Introduction to Neuromorphic Computing

It was Mead (1990) who came up with the term neuromorphic computing in the late 1980s. Initially, it referred to brain-inspired computing solutions that combined analogue and digital components. As the field evolved however, the term neuromorphic has come to embrace a much broader range of hardware implementations. In this literature review the definition of Schuman et al. (2022) is followed and neuromorphic computers are defined as non-Von Neumann computers whose design and operation are inspired by the brain and consist of neurons and synapses with communication though spikes.

Where computers based on the Von Neumann architecture comprise separate central processing units (CPUs) and memory units, neuromorphic computers aim at bringing the memory and computational units much closer together. They do not store data and instructions in memory, but the neurons and synapses comprising them take up this role. Neuromorphic computing programs are often task-specific and are determined by the topology of the neural network and its parameters. Von Neumann computing programs in contrast, are defined by explicit instructions and often have more general applicability. In addition, whereas Von Neumann computers encode numerical data as binary values, neuromorphic computers accept input as spikes whose timing, shape, and magnitude can encode numerical data. Figure 2.1 provides a side by side comparison between the traditional Von Neumann architecture and the neuromorphic architecture. Four fundamental differences between Von Neumann machines and neuromorphic computers can be identified:

1. **Closely integrated processing and memory**: Although some metaphors suggest that in

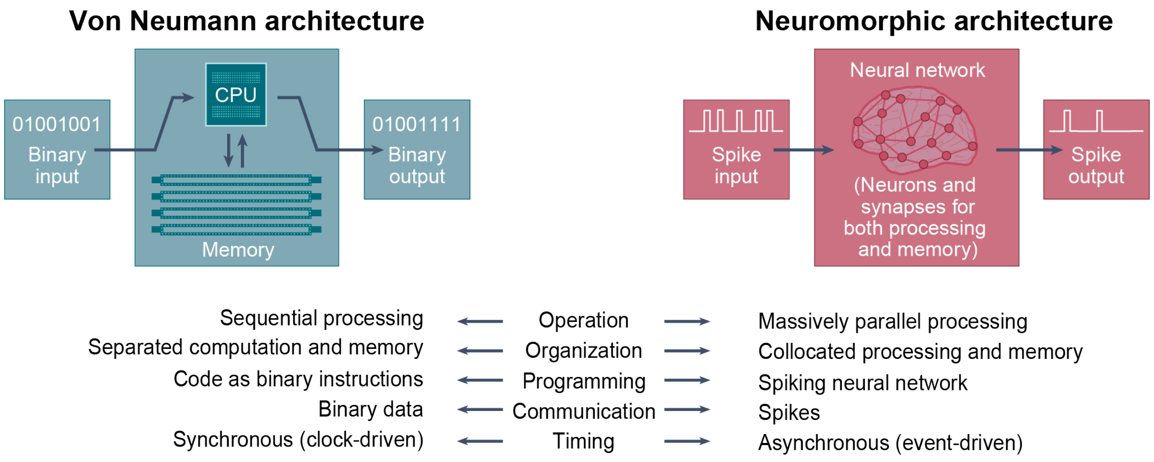

Von Neumann architecture
Neuromorphic architecture
01001001
Binary input
CPU
Memory
01001111
Binary output
Spike input
Neural network
(Neurons and synapses for both processing and memory)
Spike output
Sequential processing ← Operation → Massively parallel processing
Separated computation and memory ← Organization → Collocated processing and memory
Code as binary instructions ← Programming → Spiking neural network
Binary data ← Communication → Spikes
Synchronous (clock-driven) ← Timing → Asynchronous (event-driven)

Figure 2.1: Side-by-side comparison of the Von Neumann architecture with the neuromorphic architecture. Illustration adapted from Schuman et al. (2022).

neuromorphic computers the neurons serve as processing units (CPUs) and the synapses serve as the memory units, this is often not true. In most implementations, neurons and synapses both compute and store values. This so-called colocation of processing and memory avoids the Von Neumann bottleneck caused by the separation of the two, which restricts the maximum achievable throughput.

2. **Parallel operation**: Neuromorphic computers are highly parallel by nature. Numerous neurons and synapses can compute simultaneously, not least because of the factors mentioned in item one.

3. **Inherent scalability**: Neuromorphic computing systems are scalable. That is to say, it is possible to increase the number of neurons and synapses by merely adding more physical chips, so becoming a single large chip. This feature is advantageous since it allows hardware to be sized according to the requirements of a specific task. Implementations of this have been published using SpiNNaker (Furber et al., 2014; Mayr et al., 2019) and Loihi (Davies et al., 2018).

4. **Event-driven computation**: Neuromorphic computers are known for their event-driven computation and sparse activity. This means that information is only processed when input spikes are received and no redundant computations are made.

As a result of event-driven computation, massively parallel processing and interneuron communication by spikes, neuromorphic computers offer the prospect of very low power operation. It is this prospect that is one of the most appealing elements of neuromorphic computers. One can imagine that low power operation is advantageous for almost all applications, especially for robotic devices equipped with batteries, such as drones. In addition to these clear advantages, neuromorphic computing has attracted a great deal of attention in general due to the coming end of Moore's law, the increasing power demands associated with Dennard scaling[1], and the narrow CPU-to-memory bandwidth (Von Neumann bottleneck) (Schuman et al., 2017). Instead of attempting to continue extending and optimising an intrinsically constrained design, a point has come that it is more beneficial, at least for some purposes, to investigate alternative architectures.

The explosive growth of machine learning is also a major driver in the increasing interest in neuromorphic computing (Bouvier et al., 2019). Since their introduction almost 40 years ago, machine learning algorithms have become progressively more complex, to the point where the field is now commonly referred to as 'deep learning' due to the many layers often comprising them. Because of the vast number of parallel computations required, these algorithms perform particularly poorly on traditional

[1]A scaling law stating that the power density of transistors remains constant as their size decreases. Since 2004, however, this is no longer the case, as current and voltage could no longer continue to decrease without compromising the reliability of integrated circuits (Bohr, 2007).

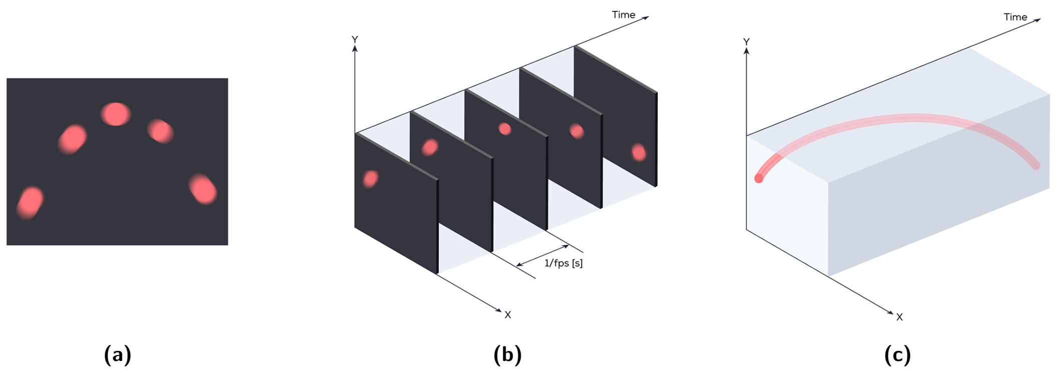


Figure 2.2: **(a)** Trajectory of a fast moving ball. **(b)** Measurement of the same moving ball using a frame-based sensor **(c)** Measurement of the same moving ball using an event-based sensor. Illustration retrieved from www.prophesee.ai.

Von Neumann architectures in terms of both time and energy consumption. Invention of hardware dedicated to efficiently enabling these types of algorithms would be a significant advancement for machine learning.

Efforts to construct an end-to-end neuromorphic computing pipeline have in common that they aim to solve three important challenges. The first challenge is finding a way of acquiring input signals that exhibit neuromorphic characteristics such as sparsity and spike-based information encoding. The second is the creation of capable, yet efficient, algorithms with brain-like processing properties. The third and final challenge is developing dedicated neuromorphic hardware having the five above-mentioned fundamentals properties.

The invention of event-based sensors presented the solution to the first challenge associated with constructing a fully neuromorphic pipeline and is discussed in more detail in the next section. The rise of the SNN meant a brain-like computing algorithm was found that embodied the desired characteristics. SNNs are the central topic of this literature review and are discussed at length in Chapter 3. The final important element, neuromorphic hardware devices, are discussed in the latter section of this chapter.

2.2 Event-based Sensing

A key principle of neuromorphic computing is event-based sensing. Event-based sensing is inspired by the ability of sensory neurons in the nervous system to only react to *changes* in the sensed quantity rather than continuously reporting its value at a fixed frequency (Tayarani-Najaran & Schmuker, 2021). Figure 2.2 illustrates the difference between neuromorphic sensing and traditional sensing. Although the strategy of periodic sampling at a constant rate has proven to be highly effective, it has some inherent flaws. First, it is constrained to band limited signals in practice due to aliasing, an issue which can render different signals indistinguishable when sampled. Secondly, sampling signals on a periodic basis may be energy inefficient when signals change only sporadically. And thirdly, the sampling interval imposes an intrinsic limitation on the minimum feasible latency. These problems are not encountered with event-based sensing it is a sampling technique in which the signal dictates the sampling. When a signal crosses a certain threshold, a sensing element such as a pixel or a filter bank element emits an event. This kind of sampling is often referred to as 'send-on-delta' or 'event-driven sampling' (Vasyutynskyy & Kabitzsch, 2010). Although event-driven sensing was pioneered in vision, it has now expanded to other sensory systems such as auditory and olfactory. The same event-based sensing principles apply to the visual, auditory, and olfactory sensory systems: when a change in the sensed quantity exceeds a threshold, an event is emitted. For this reason, only event-based vision will be used to illustrate the operation of event-based sensing. This technique is the most comprehensible and has the best coverage in the literature. Reiterating theories for the auditory and olfactory systems would contribute minimally to this literature review and is therefore not explicitly included.

2.2.1 Event-based Vision

Event-based cameras, or sometimes called dynamic vision sensors (DVSs), are bio-inspired sensors that function quite differently compared to traditional frame cameras. Instead of taking images at a constant rate, they detect per-pixel brightness changes asynchronously and output a stream of events of the brightness changes. High temporal resolution (on the order of μs), very large dynamic range ($140dB$ vs. $60dB$), low power consumption, and high pixel bandwidth (on the scale of kHz) are favourable properties that result in reduced motion blur for event cameras compared to traditional cameras. Therefore, event cameras have a great deal of potential for robotics and computer vision in conditions that are difficult for conventional cameras, such as low-latency, high speed, and high dynamic range.

Event Generation

An event camera consists of a grid of independent pixels that respond to changes in brightness. More precisely, they respond to variations in their log photocurrent $L = \log(I)$. An event $e_k = (\mathbf{x_k}, t_k, p_k)$ is generated at pixel $\mathbf{x_k} = (x_k, y_k)$ and time t_k immediately after the brightness change since the last event at that pixel,

$$\Delta L\left(\mathbf{x_k}, t_k\right) = L\left(\mathbf{x_k}, t_k\right) - L\left(\mathbf{x_k}, t_k - \Delta t_k\right), \tag{2.1}$$

exceeds a predefined threshold $\pm C$,

$$\Delta L\left(\mathbf{x_k}, t_k\right) = p_k C, \tag{2.2}$$

where $C > 0$, Δt_k is the time since the last event at the pixel in question and the polarity $p_k \in \{+1, -1\}$ is the sign of the brightness change (Gallego et al., 2019).

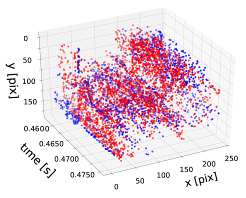
Event Representation

Quite often neuromorphic computing attempts to emulate the biological approach to problem-solving. This also applies to the transmission of sensor events to the processing hardware. As we shall see, precisely mimicking the biological approach results in difficulties. Take for instance the optic nerve linking the retina to the visual cortex in the brain. It is composed of axons of the approximately one million retinal ganglion cells, which are considered the retina's spiking output cells (Posch et al., 2014). Transferring this to an artificial vision system would mean that every pixel of an image sensor should have its own wire for transmitting data out of the array. Given the constraints imposed by chip connectivity and manufacturing technologies, it is evident that this strategy is not realistic. Nevertheless, an alternative approach has been found by inventing an encoding scheme to transmit the events.

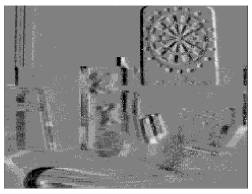
The basics of the encoding scheme are the same for all forms of neuromorphic sensing. Once an event is generated by detecting a threshold crossing, it is emitted as a data structure containing at least two pieces of information: an address and a time. The first piece, the address, can for example be the coordinates of the pixel that emitted the event, but also the index of the filter bank that emitted the event. The second piece of information is the time. It corresponds to the precise timing that the event was generated. This protocol is typically referred to as address-event representation (AER) (Tayarani-Najaran & Schmuker, 2021). Depending on the type of neuromorphic sensing, the event representation can vary slightly. In the case of event-based vision for example, a third piece of data is included in the event representation: the polarity. An individual event is then represented as the pack of data $e_k = (\mathbf{x_k}, t_k, p_k)$. In addition, in a real-time setting, time can be excluded from the event representation as event addresses are transmitted directly to a processing algorithm.

2.3 Methods for Event Processing

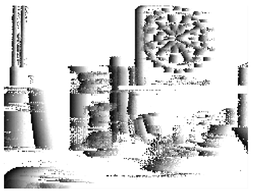
Due to the unique properties of event-driven signals, traditional signal processing techniques cannot be applied. This leaves the options of creating entirely new techniques or modifying current methods to make them compatible with these asynchronous events. Regardless of which option is selected, a



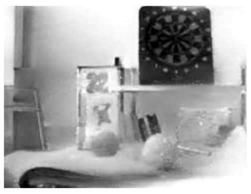
(a)



(b)



(c)



(d)

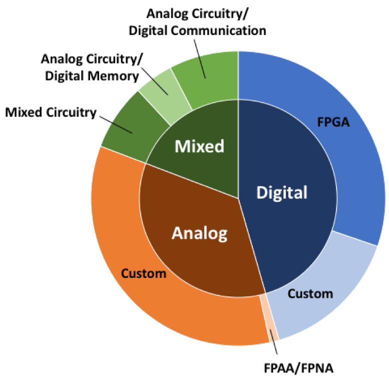
Figure 2.3: Several (pre-processed) event representations **(a)** Individual events in space-time, coloured according to pixel polarity (blue is positive, red is negative) **(b)** Brightness increment event frame ($\Delta L(\mathbf{x})$). A 2D frame is created by counting polarity pixel-wise. **(c)** Time surface, in which each pixel corresponds to the last timestamp of brightness change (darker is more recent). **(d)** Reconstructed intensity image. Grid-like representations are compatible with conventional computer vision methods. Images retrieved from (Gallego et al., 2019).

general approach framework can be provided.

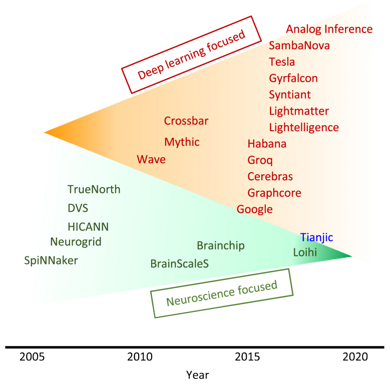
Event-driven signals are typically processed in three stages: preprocessing (input adaptation), core-processing (feature extraction and analysis), and post-processing (output creation) (Gallego et al., 2019). The first two stages, preprocessing and core-processing, are strongly correlated. That is to say, the preprocessing approach is often determined based on the core-processing algorithm to be employed. In turn, the available hardware for running the core-processing algorithm often influences the choice of core-processing algorithm. For example, when one wants to employ traditional frame-based computer vision tools, image frames are expected as an input. In this case a form of preprocessing is required that can generate frames from the event stream (Fig. 2.3a). Such techniques are widely described in literature and popular approaches include event accumulation (Fig. 2.3b) (Kogler et al., 2009; Liu & Delbruck, 2018), time surfaces (Fig. 2.3c) (Benosman et al., 2014; Lagorce et al., 2017; Sironi et al., 2018) and intensity reconstruction (Fig. 2.3d).

Preprocessing the event stream can have a negative impact on the time resolution of the measured signals and partly disregards the biological motivation underpinning event-based vision. However, it also provides a number of advantages. A reason to convert to frames, for instance, is that a single event contains only little information and is susceptible to noise. Processing several events simultaneously increases the signal-to-noise ratio and reduces computing effort when no hardware is available to benefit from the sparsity of the measured data. Another reason might be to facilitate the application of traditional computer vision algorithms that require frames as input. Despite the fact that the advantages of event cameras may not be fully realised when employing pre-processed frames, improvements are still frequently achieved compared to conventional cameras, for instance in terms of time resolution. Even when using spiking neural networks as the core processing method, it might be advantageous to apply some type of event accumulation. When simulating an SNN on hardware that cannot utilise properties as sparsity, for example, it can significantly reduce computational load.

As stated earlier, it is not ideal to use a variety of techniques to convert an event stream to frames, especially when the goal is to create a completely neuromorphic computing pipeline. As Hagenaars et al. (2021) formulated it quite nicely: “in order to realise the full potential of neuromorphic pipelines, we must move towards an event-based communication and processing paradigm in which single events are passed as-is between an event-based sensor/camera and the neuromorphic processor running an SNN, without any processing or accumulation of any kind in between”. One would be mistaking however to think that all neural network implementations in literature realise this objective. In practice, a significant amount of research over the past two decades in one form or another encoded spatio-temporal information in the event representation (Lee et al., 2021; Lee et al., 2020; Paredes-Vallés & de Croon, 2021; Stoffregen et al., 2020; Zhu et al., 2018)



(a)



(b)

Figure 2.4: (a) A breakdown of the distribution of neuromorphic computing hardware implementations. Image retrieved from Schuman et al. (2017) **(b)** Timeline of the introduction of prominent neuromorphic chips for architectures focusing on neuroscience (spiking) and deep learning (non-spiking). This graph excludes spiking chips still in development, such as those developed by SynSense, Imec, and Innatera. Image retrieved from Kendall and Kumar (2020).

2.4 Neuromorphic Hardware

Despite the promise of low power computation with SNNs, a significant discrepancy exists between most simulations of SNNs and their actual implementation on modern computing hardware (not even considering the gap with the biological brain). Simulating SNNs on Von Neumann machines is challenging due to the fact that asynchronous network activity demands asynchronous, quasi-random access to synaptic weights. (Pfeiffer & Pfeil, 2018). This mode of operation is incompatible with characteristics such as centrally located memory, a very limited CPU-to-memory bandwidth, and a minimally parallel CPU design. In order to decrease the gap between simulations of SNNs and biological SNNs, numerous SNN-optimised neuromorphic hardware solutions have been created over the past two decades. Before discussing a few prominent examples themselves, we introduce a common high level taxonomy for neuromorphic hardware systems.

Most taxonomies categorise neuromorphic hardware systems at a high level as analogue, digital, or analogue/digital hybrids. Because not everything is as straightforward as it may appear, it is worthwhile to consider the features of analogue and digital systems in relation to neuromorphic systems. Digital systems often rely on Boolean logic-based gates, such as AND, OR, and NOT, for computing, whereas analogue systems utilise the natural physical characteristics of electronic components (Schuman et al., 2017). The biological brain, for instance, is an entirely analogue system. Digital systems utilise discrete values, whereas analogue systems employ continuous values. Moreover, digital systems are often (but not always) synchronous or clock-based whereas analogue systems are typically (but not always) asynchronous. These properties of digital and analogue systems frequently do not carry over one-to-one to neuromorphic hardware implementations, as even digital systems are frequently event-driven and analogue systems utilise clocks for synchronisation.

From Figure 2.4a we deduce that most neuromorphic hardware implementations are either analogue or digital. In addition, the literature shows that digital implementations are the most prevalent in the field of SNN. The three most prominent ones are SpiNNaker, TrueNorth, and Loihi. They will be discussed in the following subsection. In addition to a taxonomy such as analogue vs digital, Figure 2.4b demonstrates that a differentiation may be formed depending on the purpose of the neuromorphic device. With the emergence of (non-spiking) deep neural networks over the past two decades, the introduction of chips with a focus on deep architectures has increased dramatically. In this literature study, however, neuromorphic chips designed specifically for deep nets are disregarded, and this serves just to demonstrate that neuromorphic hardware is not solely designed to accelerate SNNs.

2.4.1 SpiNNaker

In 2005, Spiking Neural Network Architecture (SpiNNaker) (Furber et al., 2014) was launched as one of the first large-scale neuromorphic systems. SpiNNaker uses general-purpose CPU cores to simulate biologically realistic models of the human brain. The processors are compact integer cores though, that were originally designed for mobile and embedded applications. These embedded processors utilise simple microprocessors to manage electrical functions allowing them to consume less power than conventional processors. SpiNNaker implements neurons as software running on the cores, sacrificing hardware acceleration to enhance model flexibility. This is different from implementations such as TrueNorth and Loihi, which embed simplified neuron models in customised transistor circuits. SpiNNaker can therefore be considered a somewhat more generalised neurocomputer, as it is not limited to modelling neural networks alone (Furber, 2016). The design of SpiNNaker was primarily driven by two factors: scalability and energy efficiency. It was determined that scalability is necessary since the brain (particularly the human brain) is essentially a computer with a very large number of components. In order to approximate its behaviour therefore, a neuromorphic system should embody the principle of scalability was the idea. The second important consideration, energy efficiency, followed naturally from the first. To allow very large scalability, the system must be extremely efficient in order to remain economical. When all cores are fully loaded, SpiNNaker consumes no more than $1W$ of power, hence it can indeed be said that a considerable efficiency is achieved compared to Von Neumann machines. It should be noted though that more recent neuromorphic hardware implementations are significantly more energy-efficient. With regard to scalability, it can also be stated that the goal was achieved as it is possible to create an almost arbitrarily large surface of tiled SpiNNaker chips (Furber, 2016).

2.4.2 TrueNorth

The IBM TrueNorth chip (Merolla et al., 2014) is the product of the DARPA SYNAPSE project, which sought to produce a dense, energy-efficient architecture able to serve a variety of cognitive applications (Furber, 2016). A TrueNorth chip accommodates 4096 neurosynaptic cores made up of 256 neurons with 256 synaptic inputs each. The chip is completely digital. Furthermore, TrueNorth is partially asynchronous and partially synchronous in the sense that some activities do not coincide with the clock, but the clock controls the system's fundamental time step. One TrueNorth chip consumes $72mW$ of energy at maximum load (Kendall & Kumar, 2020). The TrueNorth chip is backed by a software simulator that can precisely predict the hardware's performance. This software also facilitates learning, as on chip learning is not supported. TrueNorth, like SpiNNaker, supports the stacking of chips to construct larger systems. A circuit board with 16 chips, comprising 16 million neurons and 4 billion synapses, has been built.

TrueNorth and SpiNNaker are excellent illustrations of the extreme design decisions that can be made for digital hardware implementations. TrueNorth has opted for a design with fixed leaky integrate-and-fire neurons, minimal customizable connectivity and no possibility of on-chip learning. It is highly optimised for the selected network model and architecture. SpiNNaker in contrast is incredibly adaptable in terms of neuron model, synapse model, and learning algorithm. All can be programmed to the demands of a particular task. By permitting this however, the system loses some of its energy efficiency (Schuman et al., 2017).

2.4.3 Loihi

More recently, Intel Labs introduced a neuromorphic chip called Loihi (Davies et al., 2018). Loihi is also a fully digital chip and aims to provide operational flexibility for the evaluation of large scale SNNs (Bouvier et al., 2019). It functions at the intersection of biological realism and machine learning with SNNs. It combines on-chip learning using several learning rules, complex neuron models, synaptic delays, and multiple information coding methods. At maximum power, Loihi consumes just $0.45W$. Each of the cores individually iterates across each of its neuron groups to carry out communication. Each time a core reaches a firing state, the core creates a spike message that is sent to the cores that are connected to it. All cores must complete their iterations within the same discrete timestamp. To guarantee that all spikes reach their destinations before the procedure is repeated, Loihi sends a synchronisation message that eliminates any spikes in transit. Then, it sends a timestep-advance notice to all the cores to update their timestamp by one step to $t + 1$, allowing them to update their internal states (Davies et al., 2018). In 2020 Intel announced the arrival of Pohoiki Springs, a neuromorphic processor consisting of 768 Loihi chips and 100 million neurons (100 billion synapses), which Intel claims has the neural capacity of a small mammal's brain. This device should be able to function with less than $500W$. A figure that is, by the way, still considerably larger than that of the human brain ($\pm 20W$ for 86 billion neurons and 800 trillion synapses (Balasubramanian, 2021)) (Gallego et al., 2019).

Though certain applications can always point to the use of specific neuromorphic hardware, all in all Loihi appears to be the most versatile and capable chip yet for learning and inference with SNNs. Loihi has demonstrated to perform well on numerous applications including event-based vision (Dupeyroux et al., 2020; Massa et al., 2021), event-based olfaction (Imam & Cleland, 2020), closed loop robotic control (DeWolf et al., 2020), simultaneous localisation and mapping (SLAM) (Kreiser et al., 2020; Tang et al., 2019) and other applications (Davies et al., 2021).

3 Spiking Neural Networks

Neuromorphic computing has become a popular field of research. In contrast to von Neumann devices, brain-inspired processors strive to bring memory and computational components closer together to evaluate machine learning algorithms efficiently. Spiking neural networks (SNNs), a family of algorithms employing computational primitives that imitate the biological behaviour of neurons and synapses, have recently emerged as a very promising processing algorithm to run on these neuromorphic hardware systems. So far, SNNs have shown that they can produce competitive performance while significantly improving computational efficiency, especially when used in a setting with time-varying signals and in combination with dedicated hardware. Nonetheless, as the integration of algorithms and hardware continues to improve, the potential for SNNs to achieve even greater performance and efficiency remains enormous.

This chapter will cover in detail all the components that go into constructing a neural network with spiking activity. Section 3.1 begins with a brief overview of the evolution of neural networks. After that, Section 3.2 discusses the computational primitives that comprise a SNN. How spiking neurons communicate information with each other and how they receive input information is covered in Section 3.3. In Section 3.4, two prominent neuron models are explained that feature in the literature quite often. Finally, the chapter concludes with a description of different learning strategies employed by SNNs in Section 3.5.

3.1 History of Neural Networks

Spiking neural networks can be considered the third and latest generation of neural network models, according to the taxonomy introduced by Maass (1997) (Fig. 3.1a). His way of distinguishing generations of neural networks is based on the computational primitives from which a network is built. According to Maass, the first generation of neural networks utilized threshold gates, also known as perceptrons, which are based on McCulloch-Pitts[1] neurons. The output of these neurons is either 1 or 0 depending on whether the weighted sum of their inputs is greater than, or less than a certain threshold. These neural networks are limited to digital output, but are nevertheless capable of modelling every Boolean function, using just a single hidden layer.

The second generation of neural networks employ neurons that apply a non-linear continuous activation function to a sum of weighted inputs, hence producing a continuous set of possible output values. Networks of these kinds of neurons are referred to as artificial neural networks (ANNs). Two very important characteristics of ANNs are their ability to (1) be universal non-linear approximators (Hornik et al., 1989) and (2) the possibility to train them. The latter means that without specific prior knowledge of how the target function should be described, it is possible to iteratively program them by back-propagating error gradients, in order to successfully approach the function (Werbos, 1990).

[1]McCulloch and Pitts, 1943.

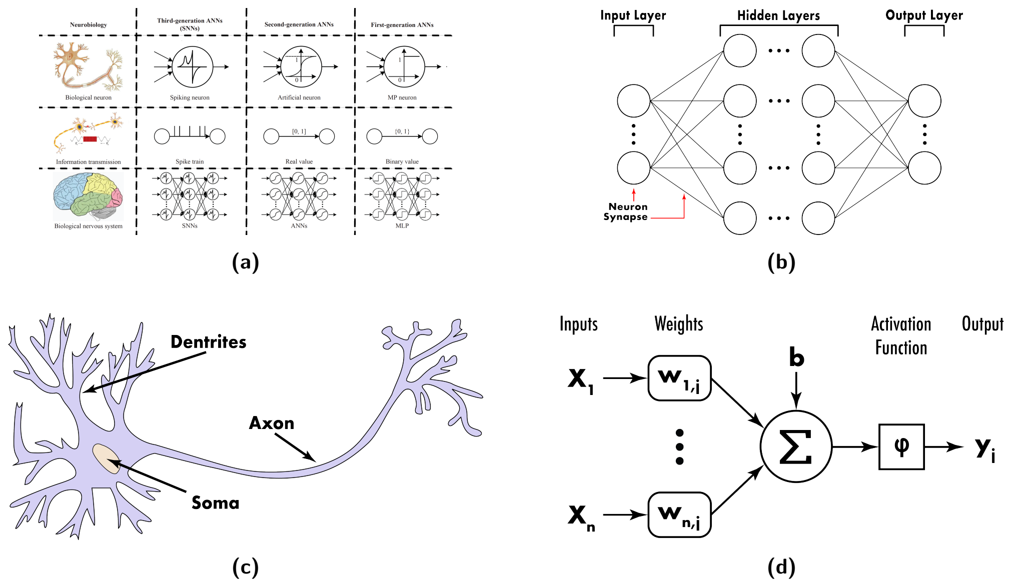


Figure 3.1: (a) Overview of the three generations of artificial neurons. Illustration adapted from Wang et al. (2020). **(b)** Schematic representation of a fully connected, feedforward neural network. **(c)** Simplyfied schematic representation of the biological neuron. Dentdrites receive inputs from upstream neurons. The input voltages are integrated onto the membrane potential of the soma. If the threshold is exceeded, the neuron 'fires' and the output is delivered to downstream neurons through the axon. **(d)** Schematic representation of the conventional artificial neuron. The neuron sums all weighted input signals, adds a bias term to it and finally applies a non-linear activation function.

The most basic type of artificial neural network is the fully connected (FNN). Unless otherwise specified, future references to ANNs will refer to this most basic form (fully connected FNN). ANNs are composed of computational units, called neurons (or perceptrons), which are grouped in layers. All neurons in one layer are connected to all neurons in the following layer via synapses. This characteristic is referred to as being 'fully connected'. Figure 3.1b depicts a schematic representation of the architecture of the FNN.

Each synapse is connected to precisely one presynaptic and one postsynaptic neuron. Before summing all contributions, the postsynaptic neuron multiplies all presynaptic activations by their respective synaptic weight (also known as synaptic efficacy). A bias term b is then added, causing the neuron to equate to

$$y_j = \sum_{i=1}^{n} w_{i,j} \cdot x_i + b_j. \tag{3.1}$$

A computational schematic of the just described artificial neuron, together with its biological counterpart are shown in the bottom row of Figure 3.1. Note that up until now, all calculations have been linear. A network built of such computational units would never be able to approximate any non-linear function. To overcome this limitation, a non-linear activation function is applied to the neuron's output y_j. The neural activation that propagates to the subsequent layer is then described by

$$x^{l+1} = \phi(y^l), \tag{3.2}$$

where ϕ is some non-linear activation function. Common examples of activation functions are the sigmoid function, the rectifying linear unit (ReLU) and the inverse tangent. For a biological interpretation of the second generation of neural networks, the activation function can be seen as transforming the output y_j into a proportional output that approximates the average firing rate of the postsynaptic neuron. This innovation of applying an activation function not only made approximating non-linear functions possible, but could also be used to ensure neural activation to be bounded (e.g., sigmoid, arctan). Furthermore, the interpretation that neuronal activations of neurons of the second generation resemble firing rates in biological neurons, made them much more biologically plausible than the first

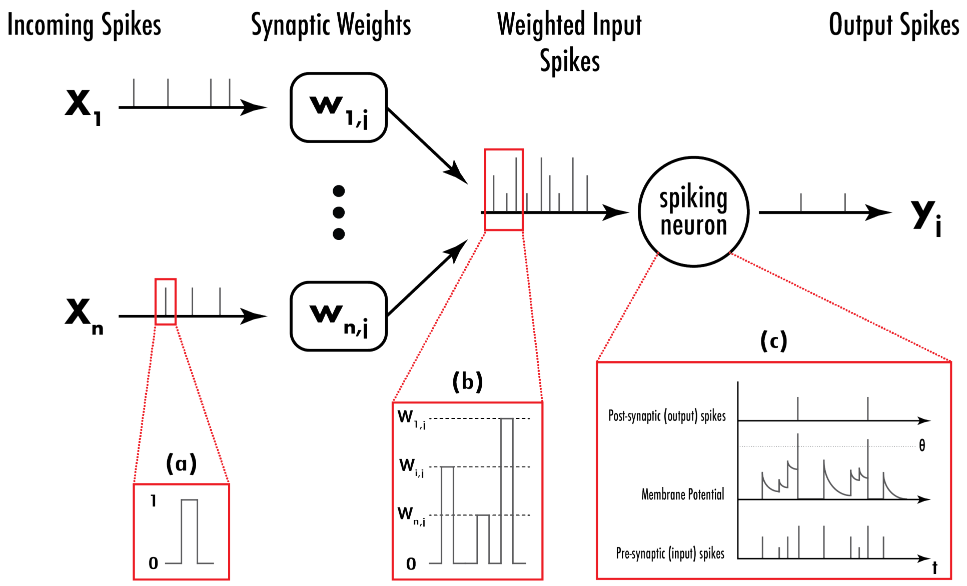


Figure 3.2: Schematic representation of the spiking neuron. **(a)** Spikes, originating from presynaptic neurons are multiplied by their synaptic weight. **(b)** The contributions of all synapses are summed in order to compute the PSP **(c)** In the spiking neuron, the PSP is integrated onto the membrane potential. Every time the membrane potential exceeds the threshold θ, an output spike is emitted and the membrane potential is reset.

generation of neurons, which are limited to binary activation (Maass, 1997).

Throughout the 1980s, ongoing research into the functioning of the human brain revealed an increasing amount of evidence that undermined the biological plausibility of the second generation. Experiments on the visual cortex of the brain revealed that the encoding of analogue variables by firing rates in the brain appeared highly improbable, particularly for extremely rapid computations (Perrett et al., 1982; Rolls & Tovee, 1994; Thorpe & Imbert, 1989). In addition, research conducted in the 1990s revealed that the majority of biological neuron systems most likely encode information using the precise timing of *action potentials* (also known as spikes). These neurobiological discoveries eventually led to the creation of the third generation of neural network models, which employ spiking neurons as computational primitives.

3.2 Spiking Neural Networks

Spiking neural networks are simply networks consisting of spiking neurons. Their architecture is typically very similar to the architecture of networks of the second generation. However, the manner in which information flows through the network differs significantly. Spiking neurons behave as integrate-and-fire units with a binary output, making them actually more comparable to the first generation of neurons than to the second. The spiking neuron, however, is dynamic and has an internal state that varies over time. The behaviour of SNNs can be explained as follows. A synapse receives an action potential, or spike, coming from a pre-synaptic neuron. Depending on the strength of the connection, as defined by the synaptic weight, the synapse will generate a post-synaptic potential (PSP) that excites the membrane of its post-synaptic neuron. The membrane potential of the neuron is a function of the PSPs of all connected synapses and changes over time. When the membrane potential exceeds a certain threshold, the neuron is said to fire (i.e., it emits an action potential). The behaviour of the spiking neuron is visualised in Figure 3.2.

3.3 Information Encoding Using Spiking Neurons

The human brain is an extremely powerful, yet very energy efficient computing system. Communication between neurons in the brain is made possible by transmittance of action potentials (spikes). In an

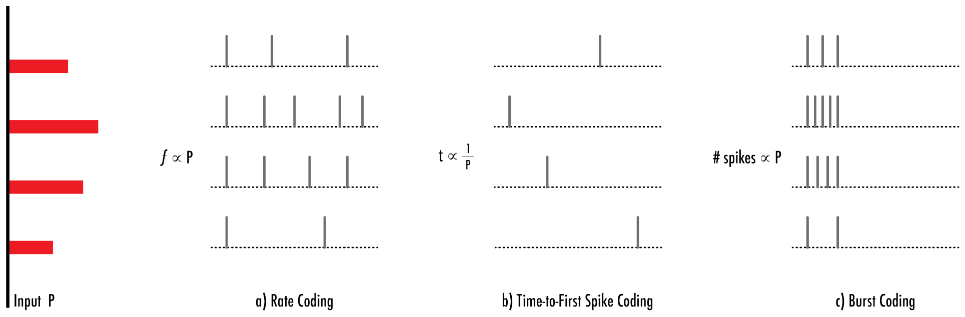


Figure 3.3: On the left, in red, four activations of varying intensity are shown. (a) Rate coding encodes activation with Poisson spike trains. Even though the exact timing between spikes is variable, the overall spike frequency remains constant. The frequency increases with the magnitude of the activation. (b) TTFS coding encodes all information in the arrival time of the first spike. Typically, the greater the activation, the shorter is the time to first spike. (c) Burst coding encodes information in a single burst of spikes. The greater the activation, the greater the number of spikes in the burst, which reduces the inter-spike interval.

effort to benefit from the mechanisms demonstrated by nature, researchers have attempted to imitate it. The mimicking of this brain communication scheme, where information is represented using a spike pattern, is referred to as *neural coding*. Extensive research has been conducted on neural coding, and numerous coding schemes have been proposed to represent complex information in spikes (Guo et al., 2021). Of these proposed neural coding schemes, the three most commonly encountered schemes will be discussed in this literature study. They are rate coding, temporal coding and burst coding. Figure 3.3 depicts a schematic of the three neural coding schemes.

Rate coding uses mean firing rates to encode information and has been the dominant paradigm in neuroscience and ANNs for a long time because of its robustness and simplicity. Poison spike trains are usually employed to accomplish rate coding. For poison spike trains, the precise timing between two spikes is random, but the average frequency over a period is constant. At least that is when a constant activation is converted to rate code. The average frequency of a Poisson spike train is proportional to the strength of the activation. Greater activation corresponds to a higher average frequency. Rate coding has been experimentally demonstrated in most sensory systems such as the visual and motor cortex (Guo et al., 2021). Despite the success of rate coding, increasing experimental data suggested that a definition of firing rate, based on temporal averaging, may be too basic to adequately represent brain activity. One of the most important arguments is that reaction times in behavioural experiments are often too quick to permit lengthy averaging of spike-rates. For example, in less than 400 milliseconds, humans can recognize and respond to visual scenes. If upstream neurons computed an average firing rate in order to decode the message of presynaptic neurons, response times would be significantly longer (Gerstner & Kistler, 2002). To explain fast responses of the biological brain, neuroscientist suggested that an alternative mechanism must be at play and temporal coding was suggested.

Temporal coding encodes information using the precise timing of spikes. Most often Time To First Spike (TTFS) is applied because it is in line with the aim to mimic rapid responses, though also schemes such as rank order between spikes (Thorpe & Gautrais, 1998) and relative spike latency (Gollisch & Meister, 2008) have been proposed. A TTFS coding scheme permits the transfer of information to a target neuron with the arrival of the first spike. In TTFS coding, the arrival time of the first spike is typically inversely proportional to the activation. That is, the greater the activation the shorter is the time to the first spike. This method allows for extremely rapid responses, even within milliseconds. Numerous studies have confirmed that the arrival of the first spike plays an essential role in various regions of the nervous system, including the retina and auditory systems (Heil, 1997; Johansson & Birznieks, 2004). In addition, TTFS coding has been demonstrated to reduce the amount of required spikes and increase speed during inference for SNNs (Guo et al., 2021).

The third form of neuronal coding is burst coding. As its name suggests, burst coding encodes information in a single burst of spikes. Burst coding, like TTFS coding, enables rapid information transmission.

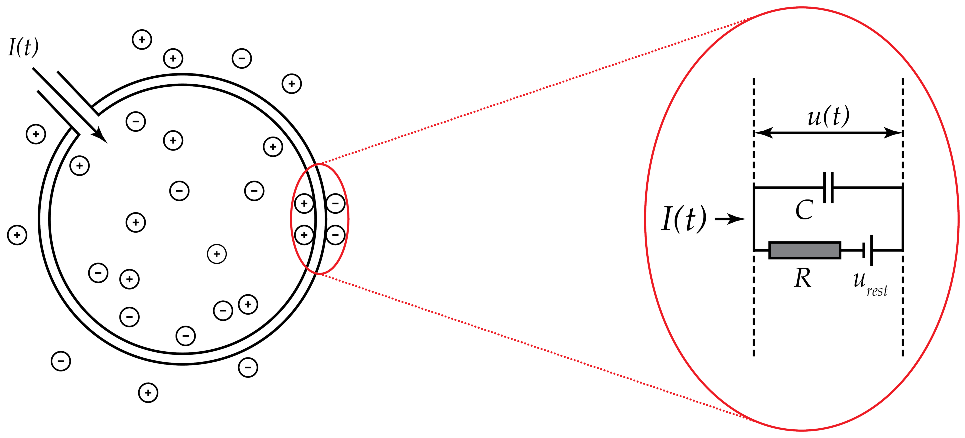


Figure 3.4: Schematic representation of how an electrical circuit models the biological neuron.

Sending a burst of spikes as opposed to a single spike improves the reliability of communication between neurons. Both the number of spikes in a burst and the inter-spike interval (ISI) in a burst can carry information. Burst coding has been observed in numerous nervous system regions, including the thalamus cortex and the hippocampus (Zeldenrust et al., 2018).

3.4 Neuron Models

So far, it has been demonstrated that spiking neurons are computational primitives that generate action potentials whenever the membrane potential of the neuron exceeds its threshold from below. Not the shape, but the timing of the spikes carries the information. This general description of the spiking neuron still leaves room for many conceptual variations. These variations particularly differ in degree of biological realism and complexity. On the less complex end of the spectrum are the neuron models commonly employed in spiking neural networks, because a less complex model requires less computational resources to simulate, while still offering adequate versatility for a host of applications. Over the past 20 years, primarily two spiking neuron models have gained true prominence in the realm of SNNs. These are the interate-and-fire (IF) model and the spike response model (SRM), although these models, under some circumstances, come down to one and the same as we will see later in this section. The remainder of this subsection's explanations and derivations will be based on Gerstner's (2014) excellent book on artificial spiking neurons. Accordingly, his book is recommended to the reader with a deeper level of interest.

3.4.1 Integrate-and-Fire Model

The dynamics of the integrate-and-fire neuron model are characterised by two distinct components. The first component is a differential equation which represents the membrane potential's progression in time. The second component is a threshold-based spike-generation mechanism.

For this derivation we will model the biological neuron as a simple electrical RC-circuit. We start by identifying the parameters that characterise the membrane potential in time. The instantaneous membrane potential of neuron i we call U_i. Furthermore, we model synaptic inputs as currents $I(t)$. When no input currents are present, the neuron's membrane potential remains at its resting value, U_{rest}. However, if the neuron does receive a synaptic input current $I(t)$ from an upstream neuron, the membrane potential U_i is perturbed from its resting value U_{rest}.

To derive a mathematical expression for the relationship between the input current $I(t)$ and the instantaneous membrane voltage $U_i(t) - U_{rest}$, basic electrical laws can be applied to the RC-circuit analogue of the biological neuron. As we can see in Figure 3.4, the biological neuron is enclosed by a cell membrane. The cell membrane can be considered an effective insulator. When a neuron is fed with a current $I(t)$, the additional electrical charge will charge the cell membrane. Consequently, the membrane functions as a capacitor with a capacity of C. Due to the fact that the membrane is not an ideal insulator, some charge will escape through it. The membrane is therefore also characterised by a finite leak resistance R. These observations lead to a leaky integrate-and-fire (LIF) model characterised by an electrical circuit consisting of a capacitor C in parallel with a resistor R that is powered by a current $I(t)$.

The law of current conservation allows us to separate the driving current $I(t)$ into its components flowing through the resistor (I_R) and the capacitor (I_C). Using Ohm's law we can write the resistive current as $I_R = U_R/R$, where $U_R = U - U_{rest}$ and using the definition of the capacity $C = q/U$, the capacitive current can be written as $I_C = dq/dt = Cdu/dt$. Therefore, the circuit's current can be expressed as

$$I_i(t) = \frac{U_i(t) - U_{\text{rest}}}{R} + C\frac{\mathrm{d}U_i}{\mathrm{d}t} \tag{3.3}$$

Multiplying Equation 3.3 by R and setting the RC-circuit's time constant to $\tau_{mem} = RC$, yields the formal differential definition of the LIF neuronal dynamics.

$$\tau_{\text{mem}}\frac{\mathrm{d}U_i}{\mathrm{d}t} = -\left(U_i(t) - U_{\text{rest}}\right) + RI_i(t) \tag{3.4}$$

Important to note is that not only the membrane potential of the neuron adheres to its own temporal dynamics, but so does the synaptic current. Modelling their temporal evolution as an exponentially decaying current, following each presynaptic spike, is the most frequently taken approach. Because differential equations are involved, it is convenient to represent a spike train as the sum of Dirac delta functions

$$S_j^{(l)}(t) = \sum_s \delta(t - t_j^s), \tag{3.5}$$

where s runs over the firing times of neuron j in layer l. Using this notation and assuming that synaptic currents add linearly, the dynamics of the synaptic current can be expressed as

$$\frac{\mathrm{d}I_i^{(l)}}{\mathrm{d}t} = -\underbrace{\frac{I_i^{(l)}(t)}{\tau_{\text{syn}}}}_{\text{exp. decay}} + \underbrace{\sum_j W_{ij}^{(l)} S_j^{(l-1)}(t)}_{\text{feedforward}} \tag{3.6}$$

where the sum is computed over all presynaptic neurons j and $W_{ij}^{(l)}$ are the weights of the synaptic connections from neurons j to neuron i.

Having derived equation 3.4 and 3.6, we must not forget that when the membrane potential of a neuron exceeds the firing threshold θ, it sends spikes to downstream neurons. The membrane voltage U_i is reset to the resting potential U_{rest} after each spike. However, this mechanism is not yet included in our definition. Only the subthreshold dynamics of the LIF neuron are described by Equation 3.4 and 3.6. That is, the neuron's dynamics in the absence of spiking output.

To capture the complete dynamics of the LIF neuron, we must extend our definition and incorporate the reset of the membrane potential in it. The reset of the membrane potential after an output spike is considered instantaneous. This enables us to introduce the reset term in equation 3.4 via an additional term that instantaneously reduces the membrane potential by $\vartheta - U_{\text{rest}}$ whenever the neuron emits a spike. The dynamics of the LIF neuron's membrane potential are then described by

$$\frac{\mathrm{d}U_i^{(l)}}{\mathrm{d}t} = -\frac{1}{\tau_{\text{mem}}}\left(\left(U_i^{(l)}(t) - U_{\text{rest}}\right) + RI_i^{(l)}(t)\right) + S_i^{(l)}(t)\left(U_{\text{rest}} - \vartheta\right). \tag{3.7}$$

For purposes of machine learning, it is often necessary to approximate the solutions of equations 3.6 and 3.7 in discrete time. The output spike train $S_i^{(l)}$ of neuron i, in layer l, at time step n, can be expressed as a nonlinear function of the membrane potential $S_i^{(l)}[n] \equiv \Theta\left(U_i^{(l)}[n] - \vartheta\right)$, where Θ is the

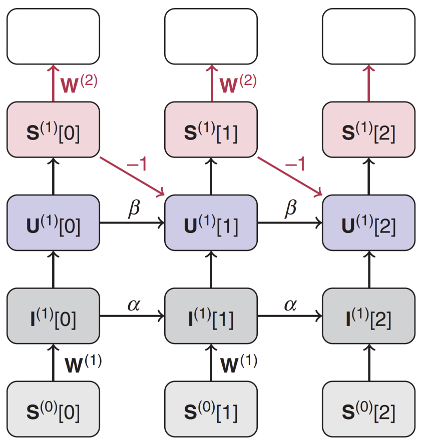


Figure 3.5: Computational graph of SNN. Eeach column depicts one simulation timestep. Information flows through the graph from the bottom to the top. Synaptic currents $I^{(1)}$ decay by α each timestep and feed into the membrane potentials $U^{(1)}$. The membrane potentials decay by β. The thresholding function generates spikes $S^{(1)}$ which causally effect the network states and reset the membrane potential. Illustration is adapted from Neftci et al. (2019).

Heaviside stepfunction and ϑ is the firing threshold. For readibility and clarity, but without sacrificing generality, we set $U_{\text{rest}} = 0$, $R = 1$ and $\vartheta = 1$. Using a sufficiently small simulation time step Δ_t, the solution of equation 3.6 is well approximated by

$$I_i^{(l)}[n+1] = \alpha I_i^{(l)}[n] + \sum_j W_{ij}^{(l)} S_j^{(l-1)}[n], \tag{3.8}$$

where $\alpha \equiv \exp\left(-\left(\Delta_t/\tau_{\text{syn}}\right)\right)$ is the decay strength. Moreover, for positive values of τ_{syn} applies that $0 < \alpha < 1$ and $S_j^{(l)}[n] \in \{0, 1\}$. The time step is indicated with n to highlight the discrete dynamics. The discrete time dynamics of the membrane potential are written As

$$U_i^{(l)}[n+1] = \beta U_i^{(l)}[n] + I_i^{(l)}[n] - S_i^{(l)}[n] \tag{3.9}$$

where $\beta \equiv \exp\left(-\left(\Delta_t/\tau_{\text{mem}}\right)\right)$. The computational graph of this model is shown in Figure 3.5.

3.4.2 Spike Response Model

Neuronal dynamics for LIF neurons have been described thus far in terms of a system of differential equations. In a second frequently employed model, the parameters are substituted for functions of time, generally referred to as filters (or kernels). The SRM neuron can be regarded as a membrane filter coupled with a function that describes the shape of an output spike (the refractory kernel) (Figure 3.6). Optionally, a time dependent threshold can be introduced. The SRM has more complex sub-threshold behaviour than the LIF model since it can account for refractoriness and adaptation (Gerstner et al., 2014).

The membrane potential of a neuron is described by the spike response model using a single variable u. When the neuron receives no input signals, it remains at its resting potential u_{rest}. The filter $k(s)$ describes the response of the neuron's membrane potential to a brief current pulse. Due to the linearity of the SRM neuron's subthreshold regime, the voltage response to a time dependent stimulating current I^{ext} may be expressed as $h(t) = \int_0^\infty \varepsilon(s) I^{ext}(t-s) ds$. Furthermore, spike firing is controlled by a threshold mechanism. An output spike is produced when the membrane potential exceeds the firing threshold. The manner in which an output spike affects the membrane potential is determined by the shape of the after-potential ν (refractory kernel). Let's consider a neuron that has fired previous spikes at times $t^{(f)} < t$. The coarse of the membrane potential is described by

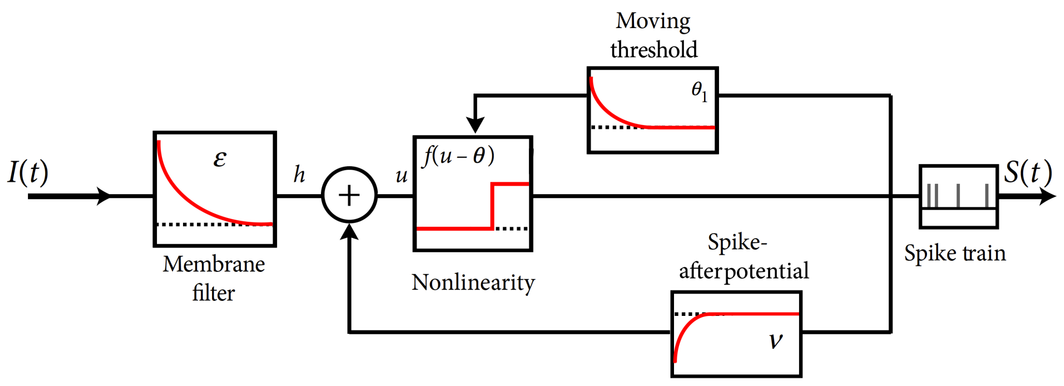


Figure 3.6: Computational graph of the SRM. The input current $I(t)$ is filtered by de membrane filter ε and gives the PSP. When the threshold θ is crossed from below, the neuron fires. Spikes feed back into the system by causing a spike afterpotential ν (refractory response) and by temporarily increasing the threshold by θ_1 (temporary adaptation of the threshold is an optional feature of the SRM that is not required).

$$u(t) = u_{\text{rest}} + \int_0^\infty \varepsilon(s)\, I^{ext}(t-s)ds + \sum_f \nu(t-t^{(f)}). \tag{3.10}$$

The sum includes all previous firing times $t^{(f)}, t = 1, 2, 3, ...$ of the neuron in question. Writing the output spike train as a sum of Dirac delta functions $S(t) = \sum_f \delta(t - t^{(f)})$, the second term of Equation 3.10 can also be written as a convolution

$$u(t) = u_{\text{rest}} + \int_0^\infty \varepsilon(s)\, I^{ext}(t-s)ds + \int_0^\infty \nu(s)S(t-s)ds. \tag{3.11}$$

Realising that the stimulating current I^{ext} is a sum of weighted spike trains originating from upstream neurons and setting $u_{rest} = 0$, it is easy to see that the membrane potential is simply the sum of all PSPs and refractory responses

$$u(t) = \sum_i w_i\,(\varepsilon * S_i)(t) + (\nu * S)(t). \tag{3.12}$$

Finally, the neuronal firing is mathematically defined as the moment when the membrane potential $u(t)$ crosses the threshold $\theta(t)$ from below, according to

$$t = t^f \quad \Leftrightarrow \quad u(t) = \vartheta(t) \text{ and } \frac{\mathrm{d}[u(t) - \vartheta(t)]}{\mathrm{d}t} > 0. \tag{3.13}$$

3.5 Learning in SNN

Synaptic plasticity is the ability of synapses to modify their synaptic efficacy (strength of connection) and is believed to be the underlying mechanism for learning and memory in the biological brain. Actually, numerous forms of synaptic plasticity coexist inside the brain, differing primarily in terms of the duration of the process, but also in the conditions necessary for modification. To date however, the precise link between synapse characteristics at the microscopic level and functional outcomes at the macroscopic scale, remains disputed (Taherkhani et al., 2020).

This biological mechanism of altering synaptic efficacies (weights) in order to learn, was carried over to SNNs. And just like in the biological brain, for SNNs, various techniques exist to update weights in order to learn certain tasks too. In general, the learning strategies can be divided into three categories: supervised learning, unsupervised learning, and reinforcement learning. Because the only form of learning that will be utilized in this thesis is supervised learning, unsupervised learning and reinforcement learning will only be discussed briefly. Following that, supervised learning will be covered in depth.

3.5.1 Unsupervised Learning

Unsupervised learning approaches are considered biologically plausible learning approaches and are often implemented with exactly that in mind. Unsupervised learning progresses according to local activities, and local activities have no concept of the problem to be solved, nor do they have any concept of whether a change is ”good” or ”bad” (Taherkhani et al., 2020). Adaptation to local activity is all that is involved in the process of learning. Unsupervised learning relies exclusively on the data’s properties. The network discovers patterns in the data without any external guidance for the desired goal. By altering the strength of synaptic connections, the learning process results in the rearrangement of connections within a neural network and under some conditions, the creation of new functions, such as input clustering, pattern recognition, source separation and dimensionality reduction (Ponulak & Kasiński, 2011). Clustering for example, in which data is organized into clusters on the basis of a particular attribute of the data, is a classic example where unsupervised learning is suited for. The reasoning is that pieces of data that are comparable with regard to the given attribute should be grouped together. In 1949, it was Donald Hebb who posed for the first time the subject of how synapses should alter their efficacy in order to learn or store information (Hebb, 1949). His work has led to the proliferation of many unsupervised techniques. Of these techniques the most widely used for learning in SNNs is spike-timing-dependent plasticity (STDP) (Caporale & Dan, 2008). According to STDP, a synaptic weight is increased (potentiated) if a presynaptic spike directly precedes a postsynaptic spike. Alternatively, a synaptic weight is reduced (depressed) when a presynaptic spike occurs immediately after a postsynaptic spike. A benefit of unsupervised learning is the reduced computing workload, as the procedure avoids repeated iterations through a training dataset and backpropagation of errors, which are usually necessary for supervised learning algorithms such as gradient descent. Furthermore, the data does not need to be labeled, which increases the usable amount of data tremendously.

3.5.2 Reinforcement learning

In addition to supervised and unsupervised learning approaches, there is also reinforcement learning, whose concept can be effectively illustrated by considering how animals learn. Animals acquire new skills or behaviour not just through direct instruction, but mostly by exploring possible actions in the presence of reward signals. In a process of trial and error, desired behaviours are reinforced with positive reward signals, whilst undesirable behaviours are punished with negative reward signals. Such a learning scenario has been effectively applied to the field of machine learning and is known as reinforcement learning (Ponulak & Kasiński, 2011). Reinforcement learning differs from classical Hebbian (unsupervised) learning in that classical Hebbian learning focuses solely on pre- and postsynaptic activity while ignoring the possible significance of neuromodulators like dopamine, which is known to also influence synaptic plasticity (Frémaux & Gerstner, 2016). Reinforcement learning approaches incorporate the role of neuromodulators by modulating the outcome of classical Hebbian learning rules (e.g., STDP) with a reward term.

3.5.3 Supervised learning

In contrast to approaches like STDP, which strive for a high level of biological plausibility, supervised learning techniques are typically less concerned with biological plausibility. To train deep SNNs, supervised training methods with spikes often employ different variations of backpropagation. Although some alternative supervised approaches exist, which now and then demonstrate competitive performance on conventional tasks and benchmarks, the focus of this literature review is on methods involving some form of backpropagation of error gradients and others are left out.

When backpropagating errors in SNNs, the credit (error) assignment in SNNs offers a significant obstacle that must be overcome, as a result of the nondifferentiability of the spiking nonlinearity. Recall that the credit assignment problem in backpropagation with gradient descent is solved by offering explicit expressions for weight updates through the chain rule for derivatives. This “chain” of derivatives contains the derivatives of the neural activation function as a multiplicative factor. For a spiking neuron, this activation function is $S_i^{(l)}[n] \equiv \Theta\left(U_i^{(l)}[n] - \vartheta\right)$. The derivative of this expression is zero everywhere, except at $U = \vartheta$, where it is poorly defined. This characteristic of the derivative of spiking nonlinearity prevents the gradient from ”flowing” and at first glance appears to renders LIF neurons unsuitable for gradient-based optimisation (Neftci et al., 2019).

According to Neftci et al. (2019), the most common methods, used to overcome the discontinuous spiking nonlinearity, can be classified in the following four categories: 1) biologically motivated local learning rules; 2) converting conventionally trained rate-based ANNs to SNNs; 3) smoothing the network model to be continuously differentiable; or 4) defining a surrogate gradient (SG) as a continuous relaxation of the real gradients. The first approach has previously been covered. The second approach technically falls outside the scope of this study because it does not combine spiking and backpropagation directly. However, because this is (or has been) a frequently used technique and because it will be addressed in the following section, it will be discussed briefly. After that, this section's primary focus will be on supervised techniques from the third and fourth categories: smoothing techniques and SG techniques.

Conversion of ANN to SNN

By adjusting the weights and properties of the spiking neurons, conventionally trained ANNs can be turned into SNNs to bypass issues associated with learning in spiking networks. The objective is to obtain the same input-to-output mapping as the original ANN with a deep SNN. Almost all conversion methods adhere to the concept of rate-coding, in which analog neuron activations are transformed into the firing rates of spiking neurons (Pfeiffer & Pfeil, 2018). The primary advantage of the conversion method is that the entire toolbox of deep learning can be utilized, allowing state-of-the-art deep networks for classification tasks to be turned into SNNs without much difficulty (Hu et al., 2018). Converting conventional networks to SNNs is not completely without problems, though. Some ANNs are difficult to convert to SNNs because negative activations in ANNs and some non-linear operations in ANNs are challenging to transfer. In addition, SNNs which have been converted from ANNs using rate code tend to be inefficient in terms of number of spikes being produced, because multiple spikes are required to encode a single real-valued activation, resulting in less energy-efficient behaviour. Lastly, ANN-to-SNN coversion cannot learn to exploit spatio-temporal features in spikes trains originating from for example event-based sensors, because they are limited to mean rate codes, which correspond to real-valued activations (Pfeiffer & Pfeil, 2018).

Smoothing Approaches

Smoothed SNNs are distinguished by their formulation, which assures well-behaved gradients that are immediately suitable for optimisation by modifying the network model in its entirety. This differs from SGs, which are only employed during the backward pass and do not alter the network model itself.

Between the years 2000 and 2010, when research into supervised learning algorithms for SNNs was relatively young, several gradient-based supervised learning methods were developed that operate only for single layer networks (e.g. Ponulak & Kasiński, 2010; Urbanczik & Senn, 2009). Since the focus of this thesis is on multilayer (deep) spiking networks, these methods are omitted and only approaches that apply gradient-based supervised learning in multiple layer networks will be covered.

Bohte et al. (2002) were the first to suggest a back-propagation training algorithm for multilayer feed forward SNNs using a smoothing approach, called SpikeProp. They used the SRM for their neurons. By linearizing the relationship between post-synaptic input and subsequent output spike time, they were able to avoid the discontinuity caused by thresholding. SpikeProp constructs an error signal used for backpropagation by defining

$$E = \frac{1}{2} \sum_{o=1}^{N_o} \left(t_o^a - t_o^d\right)^2, \tag{3.14}$$

where t_o^a and t_o^d are the actual and desired spike times of the o^{th} output neuron, respectively. Equation 3.14 shows that SpikeProp requires a target spike train to compute an error signal and therefore to ultimately produce a temporal pattern for a given input. An important characteristic to note that makes this feasible is that all neurons are restricted to firing only a single spike. Although this formulation can produce well-defined gradients in some instances, it could come with certain limitations. For example, since each hidden unit is required to generate exactly one spike per trial, it is difficult to determine the firing time for inactive neurons. Furthermore, such an activity requirement may conflict with power efficiency, where it is often advantageous to have only a fraction of the neurons active for a particular task (Neftci et al., 2019). Even though Booij and tat Nguyen (2005) presented an extension of SpikeProp to multiple spike firings, SpikeProp has not yet been applied to challenges at the scale of modern deep learning applications (Pfeiffer & Pfeil, 2018).

Gardner et al. (2015) Gardner introduced a novel supervised learning approach called MultilayerSpiker that can train multilayer feed-forward SNNs to map between spatiotemporal input and output spike pattern. MultilayerSpiker belongs to a subset of smoothing methods which are considered probabilistic. Stochasticity effectively smoothes out the discontinuity of the nonlinearity, hence enabling the definition of a gradient on expectation values. This particular learning rule optimizes the log-likelihood of producing a desired output spike train. Even in the presence of input noise, the learning method is effective and generalizes well to an example dataset.

Lee et al. (2016) were the first to present a spike-based backpropagation rule capable of training deep SNNs for classification tasks given only labels and no target spike trains. Their solution is to execute stochastic gradient descent on real-valued membrane potentials. Low-pass filtering is applied to overcome discontinuities during the times of spikes, making it possible to compute a gradient. In contrast to the approach of Bohte et al. (2002), the approach of Lee et al. (2016) employs rate code and is not limited to a maximum of one spike per neuron. This approach, helped by a range of different optimisation techniques, produces state-of-the-art performance for deep SNNs on tasks like MNIST [2] and its event-based equivalent N-MNIST (Wang et al., 2020).

Mostafa (2018) proposed a method that shows parallels to the SpikeProp algorithm in certain respects. Whereas SpikeProp assumes that connections between neurons consist of several sub-connections, each with its own delay and weight, Mostafa's method employs a more traditional network model that does not rely on pre-specified delays to translate input spike times into output spike times. In this network, non-leaky integrate-and-fire neurons and exponentially decaying synaptic current kernels were used. The desired output is determined by the timing of the first spike. Mostafa defined the input-output relationship of a network as locally linear after transforming the time variable. This formulation, unlike SpikeProp, results in an analytical relationship between input and output spike times. Furthermore, it allows the network to be trained via gradient descent directly.

SG Approaches

Surrogate gradient approaches offer an alternate strategy for solving the challenges associated with the discontinuous nonlinearity. In stead of changing the model formulation in its entirety as in smoothed approaches, a surrogate gradient is introduced *only* during the backward pass in these methods. The SG provides a means to relate the outgoing spike to an internal membrane potential value. Surrogate gradients are constructed in neuronal simulations with a discrete time grid, much like traditional recurrent neural networks. This also avoids the need to provide a particular coding scheme for spike propagation in the network's layers.

Figure 3.7b depicts the loss function of a classifier spiking network (2 input, hidden, and output neurons) as a function of the hidden layer weights from pre-optimisation to post-optimisation. The grey line represents the true loss and is characterised by plateaus where the gradient is zero, which is troublesome for gradient descent-based optimisation. As depicted in Figure 3.7c, a nonzero surrogate gradient is introduced to substitute the actual gradient.

The SG approximates the true gradient while retaining all desirable qualities, including continuity and finiteness. A natural way to consider the surrogate gradient is to imagine it as the gradient of a virtual

[2]MNIST is a large database of handwritten digits that is frequently used to train image processing tools.

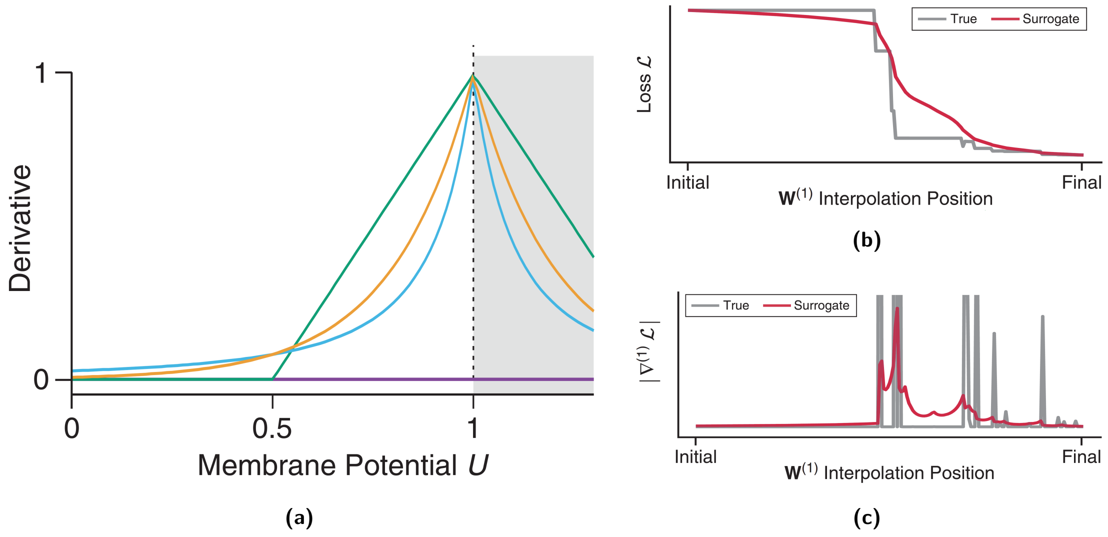


Figure 3.7: **(a)** Common surrogate derivatives. The derivative of the stepfunction (purple) is zero everywhere except at 0, where it is not well defined. The green, the blue and the orange lines are a piecewise linear function, the derivative of the fast sigmoid and an exoponential function, respectively. The axis are rescaled for better illustration and the grey area corresponds to $U > \theta$. **(b)** True loss (grey) and interpolated loss (red) over the hidden weights of a SNN classifier with 2 input, hidden and output neurons. **(c)** Absolute value of gradient (grey) and surrogate gradient (red) with respect to hidden weights. Illustration are adapted from Neftci et al., 2019

loss function (red line in Figure 3.7b) which, in contrast to the gradient of the true loss function, is nonzero. It is important to point out that in practise, this surrogate loss function is not computed because gradients are derived using approximations of gradients in the original network.

Several studies have employed surrogate gradients to train SNNs. From tackling small-scale toy problems with fractionally predictive neurons (Bohte, 2011), to training convolutional SNNs on challenging neuromorphic and vision benchmarks (Amir et al., 2017; Esser et al., 2016; Orchard et al., 2015), to training recurrent SNNs on temporal problems requiring working memory (Bellec et al., 2018; Shrestha & Orchard, 2018).

The first SG implementation was presented by Bohte (2011), in which he approximated the spiking nonlinearity as a quadratic function of the form $x(t)^{\alpha}$ for $x(t) > 0$, yielding a rectifying linear unit as the surrogate gradient. This concept is comparable to the proposed technique for optimising binary NNs. A disadvantage of the rectifying linear unit is that the surrogate gradient still vanishes in the subthreshold regime, restricting its applicability to neurons that are not inactive.

Zenke and Ganguli (2018) solved this limitation by proposing the derivative of the fast sigmoid to serve as the surrogate gradient. Alternatively, Shrestha and Orchard (2018) used an exponential function to achieve excellent performance on a variety of benchmarks. More variants for the SG exist, however without mentioning them all, it can be stated that all well-functioning SGs increase monotonically towards the firing threshold, as depicted in Figure 3.7a. Nevertheless, for a long time there was no systematic research on the importance of SG choice and design parameters.

It were Zenke and Vogels (2021) who presented exactly that study. They systematically evaluated the performance of SGs by repeatedly training the same network on the same task while varying the surrogate gradient. The authors discovered that surrogate gradient learning is particularly insensitive to SG shape variations. In contrast, improper scale selection negatively impacted learning performance. The choice of scale for a derivative can significantly impact learning performance.

3.6 Influence of Learning Strategy on Neuromorphic Hardware Performance

Training deep ANNs (second generation) with massive pre-collected datasets and the backpropagation algorithm has proven to be extremely capable of approximating high-dimensional nonlinear models. Given the success of deep learning (DL), it might seem appropriate to begin SNN research there. Unfortunately, it turns out that this is not necessarily the case. There is a common misconception that SNN chips are developed to accelerate traditional deep learning models, such as MobileNets and ResNets, in an effort to exceed GPUs in terms of energy efficiency and speed on these workloads. On the basis of this (incorrect) assumption, the attention turns to comparing efficiency and latency in terms of the number of multiply-and-accumulate (MAC) operations required in ANNs, versus the number of ”synaptic operations” in equivalent SNNs. First of all, this disregards the fact that the SNN in this context performs redundant operations to compute the internal dynamics of the spiking neurons, which are pointless because the ANN equivalent does not employ temporal dynamics. And secondly, a single MAC operation nowadays requires less energy on a dedicated ANN accelerator than a synaptic operation on a neuromorphic chip, such as Loihi. The reason for this is operating cost associated with supporting sparse network architectures. Typically, approximating a single MAC in an SNN involves multiple synaptic operations spaced out across time (using rate coding), causing even greater energy consumption and delay. In summary, it is not desirable to simply copy well-functioning algorithms from the deep learning literature while searching for algorithms that are suited for spiking chips. The ANN literature can act as a useful starting point for the search for effective SNN algorithms, but it only represents a small subset of the algorithmic realm accessible to neuromorphic devices (Davies et al., 2021). Much of this algorithmic domain is still to be explored, but research suggests that it is effective to design algorithms that take full advantage of the dynamic features of the spiking neurons. In addition, it appears to be more favourable to use them for dynamical tasks as opposed to static ones like classification.

Davies et al. (2021) employ a different taxonomy for SNN learning approaches than was previously utilised in this literature study. They differentiate between *online* learning approaches and *offline* learning approaches. Although this formulation differs from ours (supervised vs. unsupervised), both taxonomies reconcile well in the majority of cases. Online learning approaches learn as data enters the network and adjust parameters after each data sample (in real-time and in place). Offline learning approaches employ a static dataset for learning (often fed to the network in batches). When implemented on neuromorphic hardware, online approaches deploy the SNN on the hardware first and then perform learning on-chip, whereas offline approaches learn the model parameters off-chip first and then transfer them to the hardware. The on-chip learning in online approaches is carried out using on-chip plasticity features and are therefore almost always unsupervised approaches (Sec. 3.5.1). Offline learning approaches include ANN-to-SNN conversion approaches (Sec. 3.5.3) and direct approaches (Sec. 3.5.3) that apply backpropagation through time on a simulated SNN.

To determine how learning approaches function on neuromorphic hardware, we turn to the thorough study conducted by Davies et al. (2021). They examined the ’time-to-solution’ (inference delay) and energy consumption of all benchmarked deep SNN topologies and workloads utilising offline-conversion, offline-direct, and online approaches for the Loihi chip. In every instance, the precision of the trained SNNs match the precision of the reference implementations on conventional hardware. Figure 3.8 highlights the outcomes of their evaluation of eight workloads ranging from single-core to multi-chip. The grey dashed line indicates the energy-delay-product (EDP). The region above or below this dashed line represents a performance increase or decrease on Loihi relative to the reference architecture, respectively. Some tasks are compared to multiple reference architectures and are indicated by distinct markers in the figure.

3.6.1 ANN-to-SNN Conversion

The majority of the markers below the EDP line in Figure 3.8 are coloured red, indicating an offline-conversion learning strategy. In addition, the size of these red markers suggests that a considerable number of Loihi cores were utilised. Recall that conversion approaches often convert continuous artificial

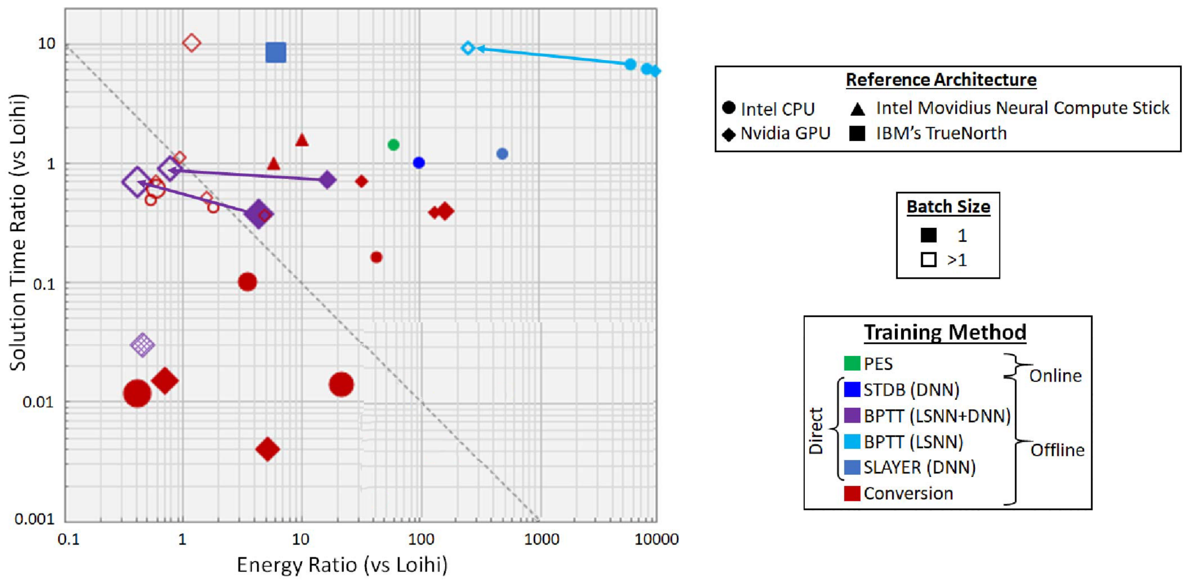


Figure 3.8: Energy and time-to-solution ratio between eight different tasks implemented on Loihi and to the same implementation on a reference architecture. Accuracy of trained SNN models implemented on Loihi match the accuracy of the implementation on reference architectures. Dotted line represents the EDP. Marker shape corresponds to reference architecture, marker color to learning approaches used and marker size to the number of Loihi cores required. Loihi always uses batchsize one, but reference architectures are distinguished between one and more than one. Arrows indicate the movement of a marker when batchsize one is substituted for a larger batch size. The closer to the top right, the beter EDP is achieved on Loihi.

neural activations in an ANN to a spike rate in a SNN. What happends is that for tasks (in this case audio keyword spotting (Blouw et al., 2019), CIFAR image classification (Liu et al., 2022) and image segmentation (Patel et al., 2021)) that ANNs process as single static input vectors through a number of dense operations, are processed by a SNN as a series of sparse computations over several time steps. Additional bits of precision require exponentially more encoding time when a spike rate is used to indicate neural activation, because the spike rate can only be estimated more precisely by extending the interval over which it is computed. Thus, when computing time increases, so does energy usage. Despite the fact that at least half of the conversion methods are below the EDP line, operating on Loihi nevertheless increases efficiency in the majority of instances (up to 100x). For smaller workloads, Loihi's delay is comparable to that of the reference architecture. In contrast, inference on Loihi takes substantially longer when the workload is quite large, particularly when it spans multiple chips. This is the result of congestion in the interconnections between chips, which have a bandwidth that is up to thirty times narrower than the bandwidth of on-chip connections.

3.6.2 Direct Deep SNN Training

Direct training of SNNs eliminates the need to explicitly select a particular coding scheme and naturally leads more to a situation in which spike-timing dictates the flow of information, instead of specific schemes like rate coding or TTFS. Spike-timing promotes lower latency and better energy efficiency and is therefore of particular interest when input information is contained in the relative timing between spikes, as is the case for data streams produced by event-based neuromorphic sensors. In addition, precise spike timing is generally more efficient than rate-coding since it requires fewer spikes, which reduces latency and energy consumption. It is even possible to design a loss function in such a way as to minimise latencies and spike counts.

Directly trained SNNs benchmarked on Loihi include networks trained with spike layer error reassignment (SLAYER) (Shrestha & Orchard, 2018), spatiotemporal backpropagation (STBP) (Wu et al., 2018), and a mix of BPTT with deep rewiring suggested in (Bellec et al., 2018), which builds recurrent long short-term SNNs (LSNNs). Since the centre of gravity of the cold-colored markers is located significantly further to the upper right than that of the red-colored markers in Figure 3.8, it demonstrates that direct approaches do indeed lead to a substantially greater reduction in EDP than conversion approaches. Even when compared to neuromorphic architectures such as IBM's TrueNorth, SLAYER exhibits a 50-fold lower EDP on Loihi. When BPTT is applied to an LSNN, an even greater EDP decrease is realised. 60,000 times less than a GPU for batchsize 1 and 2500 times less than a GPU for batchsizes bigger than 1, for which they are better suited. Applying BPTT to substantially larger collections of LSNNs that are interconnected with feedforward sub-networks is the only task where EDP reduction is notably less than the others. Due to interchip congestion and inefficient rate codes emerging from the feedforward portions of the network, this application which requires 2320 Loihi cores, loses a substantial amount of performance gains. Despite the only slight improvements, this implementation is the largest network successfully deployed on Loihi to date that outperforms conventional topologies.

4

Memory Mechanisms in SNNs

In the case of human visual perception, the retina turns spatio-temporal information contained in the visual scene's light into spike trains and patterns by employing adaptive filtering and sampling methods to increase coding efficiency (Posch et al., 2014). Retinal ganglion cells then transfer the spike trains to the visual cortex, where cognitive perception occurs. This implies that when attempting to realise a neuromorphic pipeline that mimics the biological way of perceiving, spike trains coming from a silicon retina (the sensory organ) should be transferred as-is and one-by-one to the SNN, and leave all temporal integration of signals to the network in order to perceive. To achieve this, it is important to investigate how to manipulate spiking NNs to extract spatio-temporal features of data streams accurately and efficiently. *Which architectures, neuron models, learning procedures, etc., promote or prevent the formation of robust temporal integration of information?* This chapter examines the available literature on this topic. Consideration is given not just to implementations of SNNs, but also to implementations of ANNs of which the concepts show promise that it could be carried over to SNNs.

This chapter will provide a comprehensive overview of the approaches described in the literature that introduce temporal integration of input signals in SNNs. Section 4.1 begins with a discussion on the internal dynamics of spiking neurons and how they introduce memory into SNNs. After that, Section 4.2 explains how this inherent form of memory can be augmented by adding explicit recurrent connections to a SNN. Alternative implementations of explicit connections that introduce memory called temporal skip connections, which under certain circumstances promote lower latency, are presented in Section 4.3. In Section 4.4, implementations found in the literature that use delays for temporal integration are elaborated on.

4.1 Inherent Temporal Dynamics Spiking Neuron

Spiking neural networks, as opposed to feedforward artificial neural networks, are naturally well suited for the processing of spatio-temporal information, due to having an internal state that changes over time. At any point in time, the internal state (i.e., the membrane potential & synaptic current) of the spiking neuron is not just dependent on present synaptic inputs, but also on the previous state of the neuron, which in turn has resulted from prior inputs. We might say that this principle introduces a form of 'memory' into a spiking neuron because it 'remembers' (integrates) previous synaptic input.

In order to gain a better understanding of what exactly determines the behaviour of the spiking neuron, we revisit the dynamics of the LIF neuron in Equation 4.1. We observe that two discretised differential equations and a spiking function govern the behaviour. The discretised differential equations describe the change in time of the synaptic current and the membrane voltage. The parameters α and β are the decay factors of the synaptic current and the membrane voltage, respectively. The parameter θ is the threshold of the neuron.

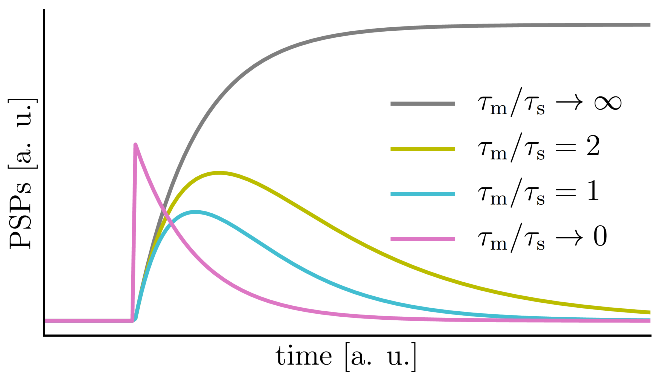


Figure 4.1: The shape of the PSP for varying ratios of the time constants τ_{mem} and τ_{syn}. As long as the ratio remains finite, the neuron can gradually forget pas inputs. Arbitrary units (a.u.) are used for a better comparison. This illustration is retrieved from Göltz et al. (2021)

$$
\begin{aligned}
I_i^{(l)}[n+1] &= \alpha I_i^{(l)}[n] + \sum_j W_{ij}^{(l)} S_j^{(l-1)}[n] \\
U_i^{(l)}[n+1] &= \beta U_i^{(l)}[n] + I_i^{(l)}[n] - S_i^{(l)}[n] \\
S_i^{(l)}[n] &= \Theta\left(U_i^{(l)}[n] - \vartheta\right)
\end{aligned}
\tag{4.1}
$$

The parameter α is an exponent of the simulation time step and the synaptic time constant ($e^{-\frac{\Delta_t}{\tau_{syn}}}$), while β is an exponent of the simulation time step and the membrane timeconstant ($e^{-\frac{\Delta_t}{\tau_{mem}}}$). The choice of τ_{mem} and τ_{syn} ultimately influences the shape of membrane potential (Göltz et al., 2021). Figure 4.1 shows how the response changes for different ratios of the time constants. For $\tau_{mem} \ll \tau_{syn}$, a lot of synaptic current leaks away while barely any voltage of the membrane leaks away. This results in a membrane potential curve which looks like a graded step function. When τ_{mem} and τ_{syn} are similar, the membrane potential is defined by a difference of exponentials (with a sum of α-functions for the special case of $\tau_{mem} = \tau_{syn}$). In the case of $\tau_{mem} \gg \tau_{syn}$, a lot of leakage occurs in the membrane, but almost none in the synapse, which results in simple behaviour of a decaying exponential.

4.1.1 Importance of Multi-Timescale Dynamics

With the introduction of surrogate gradient backpropagation (Neftci et al., 2019), it became possible to optimise neuronal parameters concurrently with synaptic connectivity. Optimising neuronal parameters was a significant departure from past modelling standards and paved the way for comprehending the functional role of the brain's cellular diversity, which is known to be very heterogeneous at many scales (Koch & Laurent, 1999). Perez-Nieves et al. (2021) discovered that increasing heterogeneity (providing each neuron with a unique membrane and synaptic time constant) enhanced overall performance across a variety of tasks and training methods, but especially for tasks with a more complex temporal structure. When the learning rule was permitted to modify both time constants and synaptic weights, a consistent distribution of time constants resembling a log-normal or gamma distribution was seen, which qualitatively matched time constants measured in experimental data. In addition, heterogeneity enables a substantial improvement in performance during inference at the expense of a minimal increase in the number of parameters and without the use of any additional neurons or synapses. Heterogeneity is thus a computationally efficient technique for neuromorphic computing (adding just $\mathcal{O}(n)$ memory vs. $\mathcal{O}(n^2)$ memory for adding neurons, because all neurons are connected).

Gehrig et al. (2020)'s study of the applicability of SNNs to do regression of numeric values in the continuous-time domain from event-based data was one of the earliest attempts to shift the emphasis away from discrete tasks such as classification. These authors trained a feedforward convolutional SNN architecture to estimate the angular velocity (tilt, pan, and roll rates) of a rotating event-camera based on the events captured during rotation. Because no such real-world dataset was available at the time of publishing, they created their own using the ESIM (Rebecq et al., 2018) event-camera simulator. The network architecture they trained to predict angular velocities consists of five convolutional layers and a fully connected layer, which was inspired by state-of-the-art designs for self-supervised ego-motion prediction. The idea was that the convolutional structure of the network interprets the spatial aspect of the input data, while the spiking dynamics account for the temporal dependencies. They found that the SNN was capable of regressing angular velocity across multiple rates with reasonable accuracy. A 6-layer ANN equivalent only marginally outperformed the SNN in terms of error. This result led the authors to the conclusion that it is possible to train SNNs to ANN-competitive accuracy on this continuous-time regression problem.

4.2 Recurrent Connections

In addition to the inherent form of memory in spiking neurons, it is also possible to introduce temporal dependencies explicitly: recurrent connections. Non-spiking recurrent neural networks (RNNs) were introduced in the late 1980s, but truly gained prominence when Hochreiter and Schmidhuber (1997) proposed the long-short-term-memory (LSTM) structure. In RNNs, recurrency in the network induces memory in the form of internal hidden states which are updated while time-stepped input feeds in. RNNs have been shown to be effective for tasks that involve temporal patterns fed into the network such as sequence to sequence classification, prediction tasks, speech-to-text transcription and other applications (Sherstinsky, 2020).

Several RNN variations have been created to ease and enhance RNN learning, which is known to suffer from issues as vanishing/exploding gradients. When the dependencies in the target sequence span many samples, these issues become particularly obvious. The LSTM unit was developed specifically as an RNN for sequential machine learning applications such as speech recognition, language comprehension, and sequence-to-sequence translation (Graves, 2014). Cho et al. (2014) proposed the gated recurrent unit (GRU) in the context of machine translation, which was later shown to produce comparable performance to the LSTM on sequence processing tasks by Chung et al. (2014). Recent developments, like the independently recurrent neural network (indrnn) (Li et al., 2020), build on the success of residual connections in CNNs to promote gradient flow in the network and obtain state-of-the-art RNN performance.

Other than the choice of neuron type and activation function, RNNs and spiking recurrent neural networks (SRNNs) are identical. The biggest difference is that on subsequent time steps, instead of floats being fed back into the artificial neurons of the same layer, spikes are fed back into the spiking neurons of the same layer. When adding explicit recurrent connectivity, the dynamics of the spiking neuron change to

$$I_i^{(l)}[n+1] = \alpha I_i^{(l)}[n] + \sum_j W_{ij}^{(l)} S_j^{(l-1)}[n] + \sum_j V_{ij}^{(l)} S_j^{(l)}[n], \tag{4.2}$$

where V_{ij} are the weights of the recurrent synaptic connections and $S_j^{(l)}$ are the output spikes of layer l. In fact, this formulation of the dynamics is also applicable to feedforward SNNs, where the past state of the spiking neuron is only fed back into itself, resulting in a V_{ij} matrix with only non-zero elements on the diagonal. For the sake of clarity, it was decided not to mention this recurrency matrix explicitly because it may lead to the misunderstanding that it is a recurrently connected SNN.

Yin et al. (2020) developed a simple but effective SRNN that is trained using surrogate gradients in order to directly apply back-propagation-through-time (BPTT) with auto-differentiation in PyTorch. They demonstrate that their implementation reaches or sometimes even outperforms artificial RNNs while exhibiting sparse activity. They report a >100x energy improvement for their SRNNs over classical RNNs on challenging tasks like the spoken Heidelberg digits (SHD) spiking dataset[1]. To achieve

[1]The Spoken Heidelberg Digits spiking dataset was developed specifically for benchmarking spiking neural networks (Cramer et al., 2022)

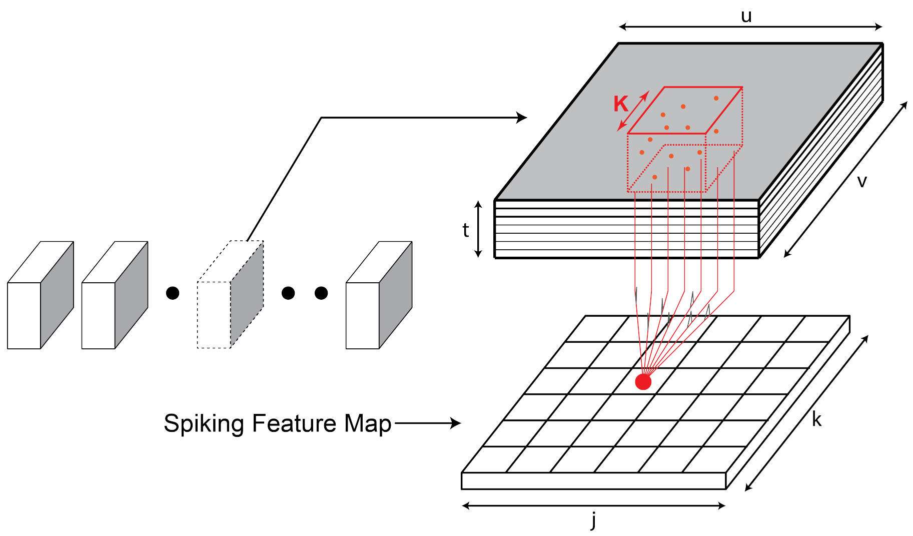


Figure 4.2: Schematic representation of how the voxel-grid representation of the input data is mapped onto 2D feature maps. These 2D feature maps enter the recurrent cells, where they are convolved with feature maps from earlier time steps.

this, the authors employ basic LIF neurons and adaptive multiple-timescale spiking neurons (Bellec et al., 2018), for which the threshold is increased after each emitted spike before decaying exponentially with time constant τ_{adp}.

Another implementation of recurrent connections was devised by Xing et al. (2020). The authors created a spiking convolutional recurrent neural network (SCRNN) architecture that utilises both convolution operations and recurrent connections, for hand gesture recognition. By combining convolutions with recurrent connections, the network can retain both the spatial and temporal relationships of event-based sequence data. Figure 4.2 shows how this is achieved. The input event stream is segmented into separate voxel grids. These segments are convolved with 3D convolution kernels with spatial size K and temporal size equal to the length of the input segment. This way the convolution operation projects spatio-temporal spiking patterns onto a 2D feature map. This 'processing' of the input event stream can be viewed as an alternative to event accumulation for acquiring frames without possibly sacrificing temporal resolution. Each coordinate of the feature map then corresponds to a spatio-temporal dynamic pattern in a different 3D volume. These feature maps are then introduced to recurrent cells in which feature maps of previous time steps are concatenated along the time dimension and convolved again. Although other researchers attempted spiking recurrent approaches, they never combined convolutions and recurrency in a complete spiking setting and for dynamic scene inputs. Furthermore, by modifying the integration duration of the input data sequence and the convolution kernel, SCRNN can accomplish arbitrary spatio-temporal resolution in accordance with the task at hand. For training BPTT was applied using SLAYER. A potential disadvantage of utilizing SCRNN is that as temporal dependencies in the input become lengthy, inference could be accompanied by significant latency, as input segments grow in size.

4.3 Skip Connections

As the name implies, skip connections in deep architectures bypass some of the neural network's layers and use the output of one layer as the input for deeper layers. Skip connections were initially developed to address a variety of architectural challenges such as vanishing and exploding gradients and degradation issues. Skip connections in neural networks can typically be utilised in two ways: addition and concatenation. Both methods are illustrated in figure 4.3.

He et al. (2015) introduced residual networks (ResNets) to solve the challenge of image classification.

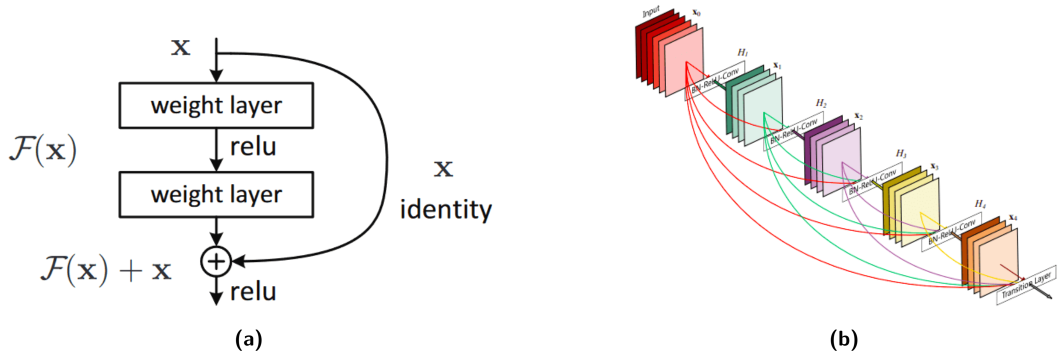


Figure 4.3: (a) Skip connection using addition, taken from original ResNet paper (He et al., 2015). **(b)** Skip connections using concatenation, taken from original DenseNet paper (Huang et al., 2018).

Using matrix addition, ResNets pass information from initial layers to deeper layers. This procedure has no additional parameters, as the output of the preceding layer is added to the next layer. Furthermore, backpropagation does not become computationally more expensive because vector addition simply introduces a gradient of one. Skip connections using concatenation were introduced by Huang et al. (2018), when they published their paper on densely connected convolutional networks (DenseNets). DenseNets concatenate the output feature maps of one layer with a deeper layer, in order to preserve high level features for use in deeper layers. Because these high level feature maps are directly concatenated to deeper layers, convolution layers in between do not need to “let through” or “preserve” shallowly acquired features for the prediction layer to have access to them. DenseNets therefore actually require fewer parameters to achieve state-of-the-art performance than Resnets, which may appear counterintuitive.

The reader might be wondering what skip connections have to do with introducing temporal integration or memory at this point. This may be accomplished by manipulating the manner in which information propagates a network with skip connections. By introducing propagation delays with skip connections, the *rollout* of the network may be altered to promote temporal integration. Before discussing this in more depth, the concept of rollout is first elaborated on.

Conventionally, the layers of a network are computed *sequentially*, resulting in no temporal integration and possibly lengthy response times. Fischer et al. (2018) presented a paradigm for describing rollouts and the degree of model parallelization they create. They demonstrate that particular rollouts enable earlier and more frequent responses, even for networks with only skip connections and no recurrent connections, and demonstrate empirically that these early responses have superior performance. The *streaming* rollout optimises these characteristics and enables a fully parallel execution of the network, hence decreasing runtime on massively parallel machines.

4.3.1 Streaming Rollout

Rolling out (or unrolling) a neural network is the process of disentangling the recursive dependencies and transforming the network into a feed-forward network (Fischer et al., 2018). Since unrolling a network is ill-defined (Liao & Poggio, 2020), various rollouts are possible for the same neural network. In general, there are two ways to unroll each edge: (1) connecting the source and target nodes at the same point in time or (2) bridging time steps. For an illustration of the first method, consider the vertical edges in Figure 4.4b. For an illustration of second method, see Figure 4.4c. Edges are typically applied in a non-time-bridging (method 1) fashion throughout a rollout and only bridge time steps when necessary, as is the case with recurrent connections. Fischer et al. (2018) refer to these rollouts as *sequential rollouts*. Alternative rollouts result in varying degrees of model parallelism and distinct network characteristics. In rollouts that induce complete model-parallelism, nodes of a certain time step in the unrolled network become computationally disentangled and can be computed in parallel (see Figure 4.4c). Fischer et al. (2018) calls these *streaming rollouts*.

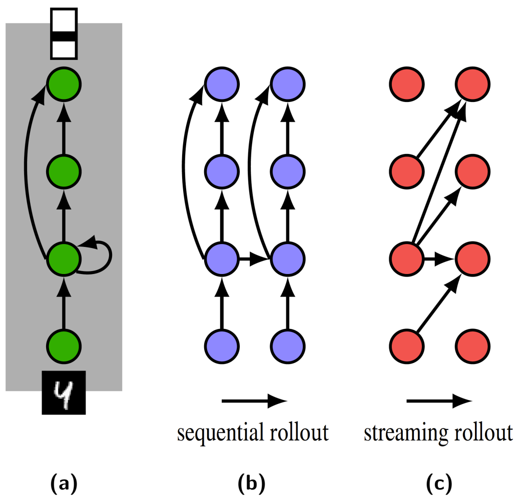


Figure 4.4: **(a)** Neural network with skip and recurrent connections. **(b)** Sequential rollout. **(c)** Streaming rollout. Nodes correspond to layers, edges represent transformations (e.g., convolution). Just one rollout step is shown. Illustration is adapted from (Fischer et al., 2018).

In addition to rollouts of RNNs, Fischer et al. (2018) also investigate rollouts of non-recurrent networks. They found that different rollout types result in distinct network behaviour, particularly in networks with skip connections. The rollout pattern effects a network's response time, which is the period between the onset of an input and the network's output. Skip connections can significantly reduce response times. A frequently useful property for dealing with spatio-temporal sequences as input from event-based sensors, for instance.

4.3.2 Temporal Skip Connections

As seen in Figure 4.4b, the combination of skip connections and a more 'traditional' sequential rollout does not offer any type of temporal integration. At no point in time can information from different input times be integrated at a single point in time at the network's output. When a streaming rollout is applied, however, information can travel deeper into the network from one time step to the next (or any other desirable amount of time steps). Now, skip connections bypass not only layers in the depth direction, but they also bridge time steps, and we can regard them as *temporal skip connections*.

Using temporal skip connections, Kugele et al. (2020) introduced a novel technique for obtaining extremely accurate SNNs for sequence processing. Specifically, they utilised ANN-to-SNN conversion to match the delays caused by ANN rollouts to the propagation delays in the target SNN. Their solution is based on the concept of streaming rollouts, as introduced by Fischer et al. (2018). They remark that the streaming rollout is in fact the same as introducing an axonal delay d_{ANN} of at least one *rollout frame* (i.e., column in the network rollout) to all *edges* (i.e., connections). The subsequent state of each neuron can therefore be derived solely from values computed in the preceding rollout frame, enabling entirely parallel updates inside a single rollout frame.

Figure 4.5 depicts how Kugele et al. (2020) implemented streaming rollouts to achieve temporal integration for a network graph (Figure 4.5a) with a single block $N_b = 1$, consisting of $N_l = 4$ convolutions, and a fully-connected layer. By giving the four (processed) input frames $F_1, ..., F_4$ a distinct colour, the colour gradients reveal which frames affect which layer at each time step. Observing the green arrows, it can be seen that the skip connection from layer 1 to layer 4 already causes activity at the output layer at $k = 3$, giving the network a latency of just 2 time steps. Furthermore, paths of different length connect the output to the input frames, with the longest path being marked in red and the shortest path being marked in blue. The difference between the longest and the shortest path defines the spatio-temporal

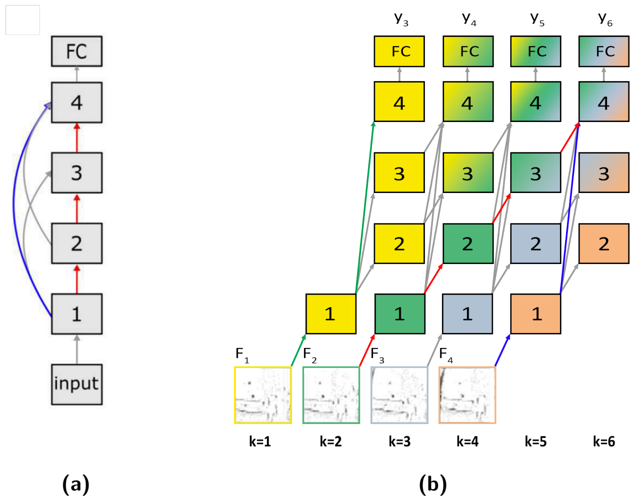


Figure 4.5: (a) Network graph of feedforward SNN with skip connections consisting of 4 convolutional layers and a fully connected layer **(b)** The streaming rollout of the same network. Different path lengths (red and blue path from F_2 and F_4 to y_6) allow for temporal integration. Illustration is adapted from (Kugele et al., 2020).

receptive field. In the case of the example $\tau = 3$ (hence there is a mix of three colours). This also suggests that the size of the receptive field can be controlled by selecting a network depth. This allows for adaptation to the temporal scale of a particular problem at hand.

Kugele et al. (2020) use the DenseNet architecture for all their experiments. A schematic of the DenseNet architecture is show in Figure 4.3b. Each layer, in each DenseNet block, is connected to all previous layers. What makes such dense connectivity possible without running out of memory is that these networks require only very narrow layers of just 9 output feature maps. Reaching state-of-the-art performance on N-MNIST requires only $3 \cdot 10^5$ parameters using the DenseNet implementation of Kugele et al. (2020) compared to approximately $2 \cdot 10^6$ parameters using SNN implementations of Wu et al. (2019) and Lee et al. (2016). Similar parameter numbers compared to other SNN implementations are also reported for other datasets including DVS-Gesture and Cifar10-DVS. The DenseNets in Kugele et al. (2020) use networks with $N_b = 3$ blocks, containing $N_l = 5$ layers each. The spatio-temporal receptive field of implemented DenseNet becomes $D = N_l N_b + 1 = 5 \cdot 3 + 1 = 16$. Furthermore, the latency of DenseNets is equal to N_b rollout frames, compared to D of recurrent networks which compute all D layers for each output. These temporal skip connections in streamingly deployed DenseNets facilitate rapid approximate network outputs that are improved over time. Finally, a remark is that the authors used ANN-to-SNN conversion with spike rate encoding in order to train their networks. Despite using relatively few parameters compared to other convolutional SNN approaches, energy efficient implementation on neuromorphic hardware is not guaranteed this way.

4.4 Delays

Numerous studies have demonstrated that synaptic delays are present in biological neurons (Katz & Miledi, 1965; Minneci et al., 2012; Wang et al., 1985). For instance, they affect the nervous system's capability to process information. In SNNs, the concept of learning delays for processing spatio-temporal input streams is typically implemented using two distinct methods. These methods can be referred to as "delay selection" and "delay shift".

4.4.1 Delay Selection

Delay selection is a mechanism in which a presynaptic and a postsynaptic neuron are connected via a connection consisting of a fixed number of synaptic terminals, each of which functions as a sub-

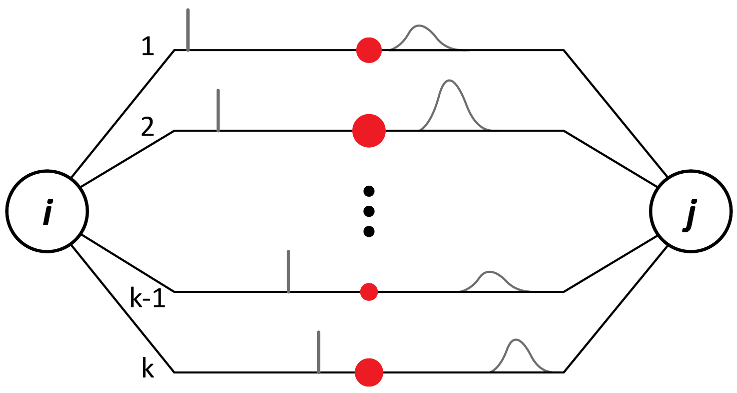


Figure 4.6: Schematic representation of a synapse between presynaptic neuron i and postsynaptic neuron j, consisting of k sub-connections. Each sub-connection has its own delay and own weight (red dot).

connection with a distinct delay and weight. This configuration is depicted schematically in Figure 4.6. Throughout the learning process, the weights are modified and the sub-connections with appropriate delays are strengthened, while those producing undesirable output spikes are reduced. It were Bohte et al. (2002) who proposed the SpikeProp algorithm (sec. 3.5.3), which was the first framework to perform delay selection in conjunction with error backpropagation. Numerous researchers have utilised this algorithm to learn delays in a wide variety of scenarios since (e.g., Ghosh-Dastidar & Adeli, 2007, 2009; McKennoch et al., 2006; Shrestha & Song, 2015; Sporea & Grüning, 2013; Xu et al., 2013). Using sub-connections with multiple synaptic delays increases learning performance by facilitating the production of output spikes at various desired timings. This, however, increases the number of learning parameters (synaptic weights) and, consequently, the computing cost, as updating a large number of parameters at each learning epoch is required.

4.4.2 Delay Shift

The delay shift technique is a delay-learning approach used to train a coincidence detector (CD). A CD is a neuron model that is biologically realistic, in which the neuron fires in response to simultaneous input spikes. A key element of the learning method is the adjustment of the synaptic delay in the model and not the synaptic weights (examples of these implentations include: Adibi et al., 2005; Pham et al., 2008; Taherkhani et al., 2015, 2018). These implementations, however, are suitable for learning a desired spike train from an input spike train. A restriction in which we have less interest. In addition, all approaches employ local learning rules rather than error backpropagation. Due to these factors, these approaches fall outside the scope of this literature review and will not be discussed in greater detail.

4.4.3 Delay using Backpropagation

In order for a network to learn a delay shift via error backpropagation, an entire time sequence must be introduced at once. The reason for this is that at the time of the backward cycle, spatio-temporal error credit has to be assigned to the parameters of the network. This error can only be assigned to connected nodes in the computational graph; otherwise, the chain of derivatives would be broken. Computing a gradient towards a delay suggests a change in delay and, thus, an alteration to a connection in the computational graph. In practice, however, this is not possible because the network has never computed alternative paths during a forward pass. It has no means of knowing how changing a delay would have affected the network retroactively. For a network to learn the delays with backpropagation, a longer sequence must be introduced all at once so that it retains access to all time dependencies during backpropagation. A backpropagation framework which meets the criterion was introduced by Shrestha and Orchard (2018). Their framework, named SLAYER, takes time sequences as inputs as a whole. To illustrate how SLAYER learns delays, an example network with an event-based input data stream are

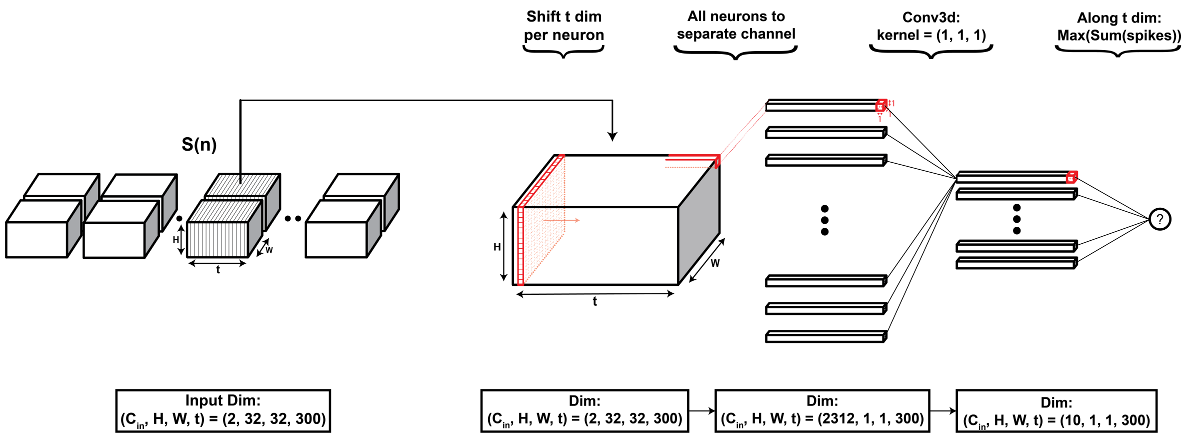

Shift t dim per neuron
All neurons to separate channel
Conv3d: kernel = (1, 1, 1)
Along t dim: Max(Sum(spikes))
S(n)
H
W
t
?
Input Dim: (C_in, H, W, t) = (2, 32, 32, 300)
Dim: (C_in, H, W, t) = (2, 32, 32, 300)
Dim: (C_in, H, W, t) = (2312, 1, 1, 300)
Dim: (C_in, H, W, t) = (10, 1, 1, 300)

Figure 4.7: Schematic respresentation of how a segmented input event stream propagates a fc feedforward network when using the SLAYER framework. $\{1, 1, 1\}$ 3D convolutions allow the spikes to propagate as if it were fc and at the same time allow spikes to be shifted within the segment through error backpropagation.

presented.

Consider an input sequence $S(n)$, $n = 0, 1, 2, ..., N$ as illustrated in Figure 4.7. At each time step, $S(n)$ is a 4D tensor with shape $\{C, H, W, t\}$, where C is the number of channels, H and W denote the height and the width of each frame and t corresponds to a predefined time resolution. A given event-based video stream can be freely segmented into multiple tensors according to the desired temporal resolution. For instance, a 3 second $32x32$ spatial resolution event data stream, with a $300ms$ temporal window and a $1ms$ sampling time, can form an input sequence $S(n)$, $n = 0, 1, 2, ..., 10$, where each individual segment is a tensor of shape $\{2, 32, 32, 300\}$ (one channel for positive events and one channel for negative events).

Segments of $S(n)$ are fed into the network sequentially. After a segment enters the network, a delay is applied along the time dimension. More precisely, we regard the spikes at every spatial position $\{H, W\}$, in every channel C, in the segment as the spike train $s_{C,H,W}(t)$ within the temporal resolution window t. These spike trains are causally shifted in time, according to

$$s_d^l(t) = \delta(t - d^l) * s^l(t) \tag{4.3}$$

$$d^l = \begin{cases} 0 & d < 0 \\ d & d \geq 0 \end{cases} \tag{4.4}$$

where δ is the Dirac delta function, $s_d^l(t)$ is the shifted spike train and d is a certain delay which is clipped at zero (for causality). In order to extract temporal features, the shifted spike trains then are to be forwarded into a fully connected layer, while preserving the time dimension. However, it is not readily apparent how this might be accomplished. Here, the authors of SLAYER have devised a clever trick, since they employ a 3D convolution for this purpose. Each spike train $s_{C,H,W}(t)$ is assigned to its own channel in the tensor, and the height and width dimensions are flattened. This results in $C \cdot H \cdot W$ spike trains of length t. By applying a 3D convolution with a kernel size of $\{1, 1, 1\}$ and $C \cdot H \cdot W$ input channels, a fully connected layer can be imitated.

To understand why this is the case, we will first review the mathematics underlying the 3D convolutional operator in neural networks. Suppose that we have a 3D convolutional layer with N_C input channels and a convolution kernel of size $\{m, n, o\}$. In the simplest case (stride of 1, no padding, etc.) every output channel (feature map) can be described as

$$C_{i,j,k}^l = bias^l + \sum_{\gamma=1}^{N_C} \left(\sum_{a=0}^{m-1} \sum_{b=0}^{n-1} \sum_{c=0}^{o-1} w_{\gamma,a,b,c}^l C_{\gamma,i+a,j+b,k+c}^{l-1} \right), \tag{4.5}$$

where $bias^l$ is the bias of that output channel, C_γ^{l-1} is the γ^{th} input channel and w_γ^l is the 3D kernel for that input channel. What Equation 4.5 does can be summarised as follows: it describes kernels that slide through all three dimensions of the input tensor, in each channel, after which all channels' contributions are summed together. Lastly, a bias is added. Because SLAYER uses a kernel of size $\{1,1,1\}$, and because the height and width of the input tensor are also equal to 1, Equation 4.5 reduces to a single weight sliding along the time dimension, in each channel:

$$C_{1,1,k}^l = bias^l + \sum_{\gamma=1}^{N_C} w_\gamma^l C_{\gamma,1,1,k}^{l-1} \tag{4.6}$$

Equation 4.6 shows that in this context, applying a 3D convolution to the input tensor, is the same as weighting all input spike trains and then summing them, which in fact is equivalent to the operation of a fully connected layer. To convert the resulting spike trains into response signals, the spike trains are convolved with the spike response kernel. The SRM neuron state is updated by the sum of the postsynaptic potential and refractory response, according to

$$\begin{aligned} u_i^l(t) &= \sum_j w_{ji}^l \left(\varepsilon * s_j^{l-1}\right)(t) + \left(\nu * s^l\right)(t) \\ &= \sum_\gamma w_\gamma \left(\varepsilon * C_\gamma^{l-1}\right)(t) + \left(\nu * s^l\right)(t) \\ &= \left(\varepsilon * C^l\right)(t) + \left(\nu * s^l\right)(t) \end{aligned} \tag{4.7}$$

where we have used that the weight of the $\{1,1,1,\}$ convolution kernel, times the output channel of the 3D convolution, is equal to a weighted spike train, as in the SRM.

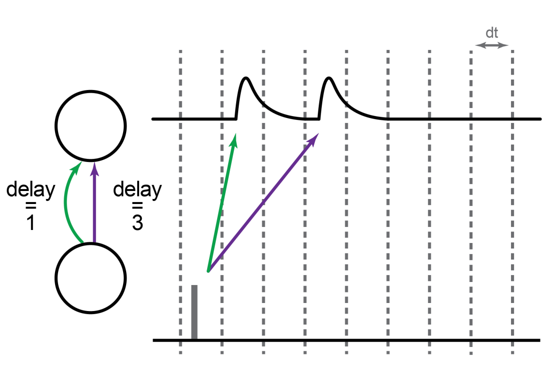
Coming back to the delay layer in the example network, we notice that we learn one delay per spike train $s_{C,H,W}(t)$ and thus one delay per neuron. Such a delay is considered to be an axonal delay and not a synaptic delay, which were discussed in the previous subsection (the difference is that with axon delays, each neuron's outgoing synapse has the same delay, as opposed to different delays for each synapse.). Synaptic delays can also be learned in a similar fashion using SLAYER (Shrestha & Orchard, 2018), but for the sake of simplicity only axonal delays were considered here.

Due to spike buffering, implementing synaptic transmission delays in real-time neuromorphic devices incurs a significant memory and computing overhead. Moreover, only a handful of neuromorphic devices allow synaptic transmission delays. We will discover in the following part that there are ways to avoid this.

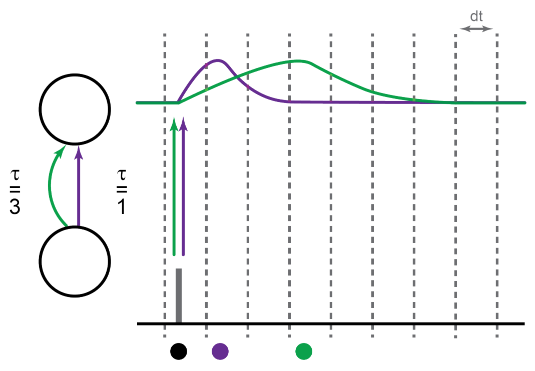
4.4.4 Delays as Temporal Convolutions

An alternative way of implementing delays to achieve temporal integration was suggested by Weidel and Sheik (2021). They proposed 'WaveSense', a spiking neural network that utilises the time constant of the neuron's spiking dynamics as an alternative to (dilated) temporal convolutions. The authors argue that compared to buffering spikes or employing synaptic delays, this reduces the memory needs for temporal spiking algorithms, despite the absence of quantitative evidence. WaveSense is based on the WaveNet architecture (van den Oord et al., 2016), which was designed to spot keywords. Temporal input data, such as audio streams, are continuously processed without buffering, making WaveSense potentially suited for processing other event-based data streams.

The central concept is to model the delay periods in WaveNet's dilated convolutions as synaptic time constants in spiking neurons for efficient implementation on neuromorphic hardware. For this goal, the authors extend the original WaveNet architecture to the spiking domain and train the network with the well-known spiking backpropagation method SLAYER. They accomplish the implementation of synaptic time constants by using exponential kernels for synaptic $\epsilon_{syn}(t) = e^{(-t/\tau_{syn})}$ and membrane $\epsilon_{mem}(t) = e^{(-t/\tau_{mem})}$ dynamics and then convolving these kernels to arrive at the PSP kernel according to



(a)



(b)

Figure 4.8: (a) Projection in time with convolution kernel of size 2, when implemented with delays. **(b)** Projection in time when implemented with chosen synaptic time constants, as in WaveSense. Note that the colored dots are located where the response reaches its maximum, 1 and 3 time steps from the input spike time for $\tau = 1$ and $\tau = 3$, respectively.

$$\epsilon(t) = (\epsilon_{syn} * \epsilon_{mem})\,(t) = \frac{\tau_{syn}\tau_{mem}\left(e^{-\frac{t}{\tau_{syn}}} - e^{-\frac{t}{\tau_{mem}}}\right)}{\tau_{syn} - \tau_{mem}}. \tag{4.8}$$

Figure 4.8 illustrates how the concept of WaveSense differs from 'normal' delays. Where in Figure 4.8a the spiking input is delayed by one simulation time step after which the PSP is computed, Figure 4.8b shows that the PSP starts rising immediately after the input spike, reaching its maximum value only after τ time steps.

The study demonstrates that the time constant of the LIF neuron can in fact be utilised to successfully incorporate time delays, as necessary for temporal convolution. Because WaveNet does not buffer any spikes from the past explicitly, it should be very efficient in terms of memory compared to WaveNet (Weidel & Sheik, 2021). Furthermore, during inference, the WaveSense architecture accepts spike streams directly and not buffered frames as input. A favourable characteristic when trying to achieve brain-like computing. A final remark is that the authors propose that the synaptic time constants be manually changed based on the task, as opposed to learning the ideal time constants for a given activity, a choice that potentially leads to suboptimal results.

III

Preliminary Evaluation of Memory Mechanisms in SNNs

5 Methodology

Before starting the actual research for this thesis, preliminary experiments have been carried out in order to gain a good understanding of the concepts and challenges that are associated with designing and training spiking neural networks, on tasks that require memory, in a simulation environment. To that end several spiking and non-spiking neural network models found in literature that posses different forms of memory mechanisms have been trained on two variants of an artificial dataset. The simulation of the experiments is done in Pytorch.

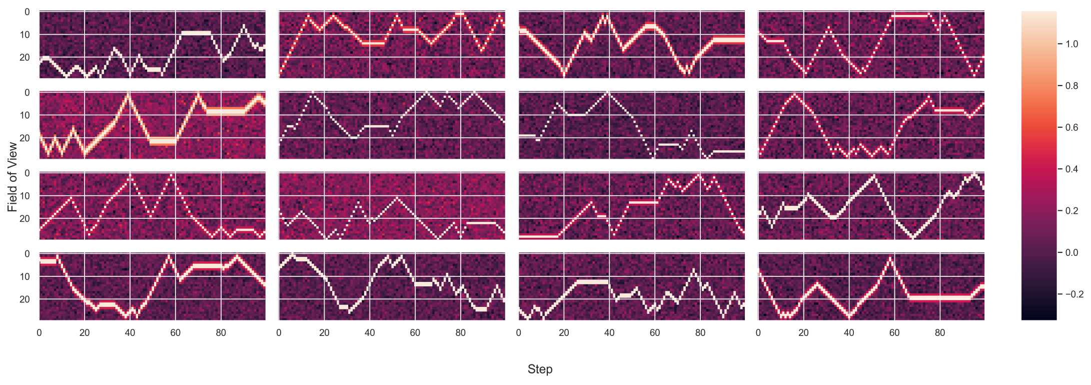
This chapter is structured as follows. In the first section (5.1), two variations of an artificial task in which a pixel moves along a 1D trajectory at varying speeds are discussed. Both of these datasets require memory in order for a neural network to successfully solve them. After describing these tasks, section 5.2 discusses ten neural network models that will be trained to solve these tasks, as well as what their respective memory mechanisms are. An emphasis will be placed on pairing spiking and non-spiking networks.

5.1 The Artificial Tasks

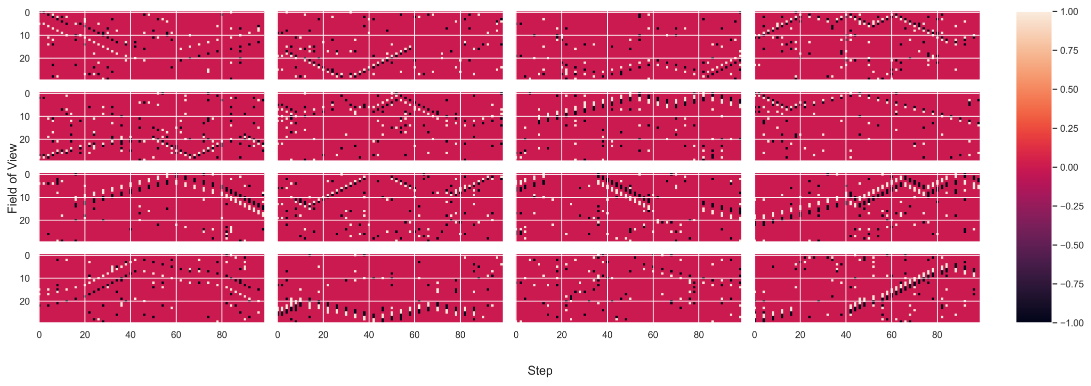
For the preliminary evaluation of memory mechanisms in spiking neural networks, a task was needed that requires memory to solve. A task that can only be solved by integrating multiple timesteps of the input, in order to produce the correct output. To that end, two variants of a 1D moving pixel task have been devised. The first variant can be compared to observations made by a tradional frame-based video camera. The second variant can be compared to observations made by an event-camera of which the events are accumulated in small equally sized time intervals. By feeding the classic and accumulated event frames of the moving pixel, timestep-by-timestep, into a network, the network must estimate the pixel's velocity.

5.1.1 The classic 1D Moving Pixel

The classic 1D moving pixel task is an artificially generated dataset of a pixel moving at a variable speed along a straight line. Several realisations of the trajectory of the pixel are depicted in Figure 5.1a. Every slice of a generated sequence can be considered as the (simplified) equivalent of a light intensity measurement made by a traditional fixed framerate video camera of a small object, moving up and down. The field of view of the imaginary camera sensor is 30 pixels. As can be seen in Figure 5.1a, a different width of the pixel is picked randomly for each sequence. The possible widths range between 1 and 5. The velocity of the pixel varies continuously within a sequence and changes sign when it reaches the edge of the field of view of the 'sensor'. Available velocities range between -2 and 2 (only integer values). In addition to varying in width and velocity, the edge contrast of the moving pixel also varies, simulating the low contrast of edges captured by a real camera (see red pixel edges in Fig. 5.1a). Similarly, normally distributed noise is added to the background.



(a)



(b)

Figure 5.1: (a) Randomly generated dataset of a 1D moving pixel in a field of view of 30 pixels with a variable speed over 100 time steps. **(b)** Randomly generated event dataset of a 1D moving pixel in a field of view of 30 pixels with a variable speed over 100 time steps.

5.1.2 The event-based 1D Moving Pixel

With traditional fixed framerate observations of a small moving object, a network only requires two consecutive frames to estimate the pixel's velocity since, the velocity is already well approximated by the difference in position between the two frames. This prevents evaluating the longer term memory capabilities of networks. In order to address this need, a second artifical task has been compiled. This task represents observations from an event camera, observing the same small object moving along a 1D trajectory (see Figure 5.2). Events captured by this fictitious event sensor are accumulated over fixed time intervals as opposed to a fixed number of events. This is done because, under the condition that the object's velocity is tiny in comparison to the size of these intervals, this introduces empty frames in between frames that do contain events (see Fig. 5.2). This method of constructing our artificial task enables us to evaluate the memory mechanisms in our networks over longer timescales because velocity estimation requires the integration of more than one timestep. Note that the lower the velocity, the less frequent the events and the longer their temporal dependencies. To simulate a real-world environment, noise events are also added to the sequences. Note that as velocity decreases, events become less frequent and their temporal dependencies become longer. Noise events are also added to the sequences to simulate real-world observations.

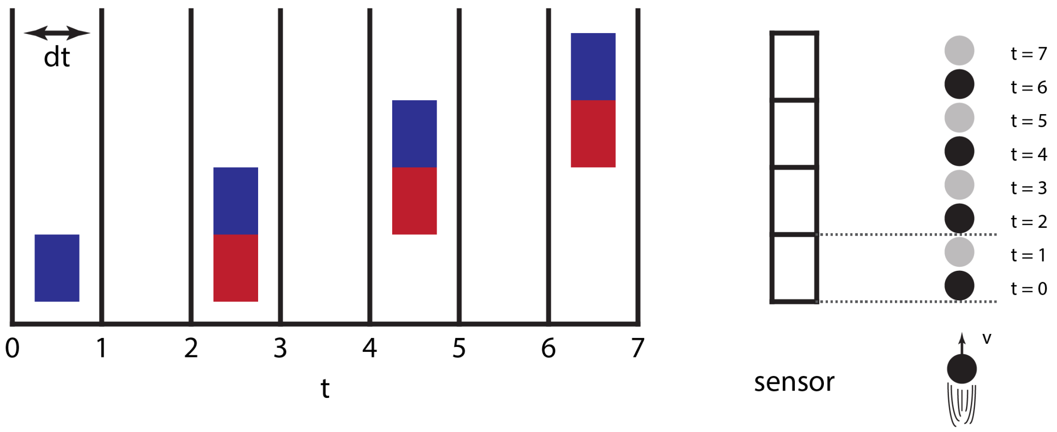


Figure 5.2: Illustration of how the event dataset is realised. On the right: a small object moving through the field of view of an event sensor of size four pixels. On the left: equally sized time bins that accumulate the events, captured by the fictitious event sensor. The assumption is that the object moves slow, relative to the bin size dt, so that even at the largest speed at least one empty bin is in between events. Faster motion of the object thus results in faster successive events and slower motion in a greater number of empty bins in between events. Blue and red denote positive and negative events, respectively.

5.2 Network Models

This section describes the neural network models that are trained to solve the moving pixel tasks. All these networks, both spiking and non-spiking, were simulated in PyTorch. PyTorch provides a very customizable and intuative framework for performing gradient-based optimisation. It benefits from fast tensor computing with strong acceleration via the GPU and has an automatic differentiation system built in. Standard neural network layers such as the fully-connected feedforward layer, the convolutional layer and LSTM block layers are easily customised to their spiking equivalents. Additionally, surrogate gradients for the spiking non-linearity can simply be defined mathematically and the auto-differentiation function of PyTorch does the rest. Code listing A.1 provides a very basic example of an SNN implementation in PyTorch. SNN implementations in PyTorch are to a large extend analogous to non-spiking ANN implementations in PyTorch. The only difference is an added spiking mechanism and a variable that keeps track on the spiking neuron's state.

Eleven neural networks have been implemented for the preliminary experiments in total. One of those does not have the ability for temporal integration, while all others do. Section 5.2.1 features three very common NN architectures that will acts as baselines. Afterward, the focus is placed on four pairs of spiking neural networks and their non-spiking counterparts. By implementing the non-spiking counterparts as well, we will have access to a good baseline and may discover interesting things when comparing between the two. Further details of the networks that are discussed in the following can be found in Appendix A.1 on page 83.

5.2.1 ANN & RNN & GRU

The non-recurrent, non-spiking ANN is the first neural network trained to solve the moving pixel tasks. The ANN has a single hidden layer comprising 60 neurons. Because this type of network lacks internal memory mechanisms, it should be incapable of learning both the classic and the event task. It is used to verify that temporal integration is indeed required to complete the tasks.

The next architecture is the basic non-spiking RNN. It comprises a single hidden layer of 60 recurrently connected neurons. The explicit recurrent connections should give the RNN plenty of temporal integration capability to learn the classic moving pixel task. The event based moving pixel task will most likely prove more difficult, but it is still expected that the RNN is capable of learning it.

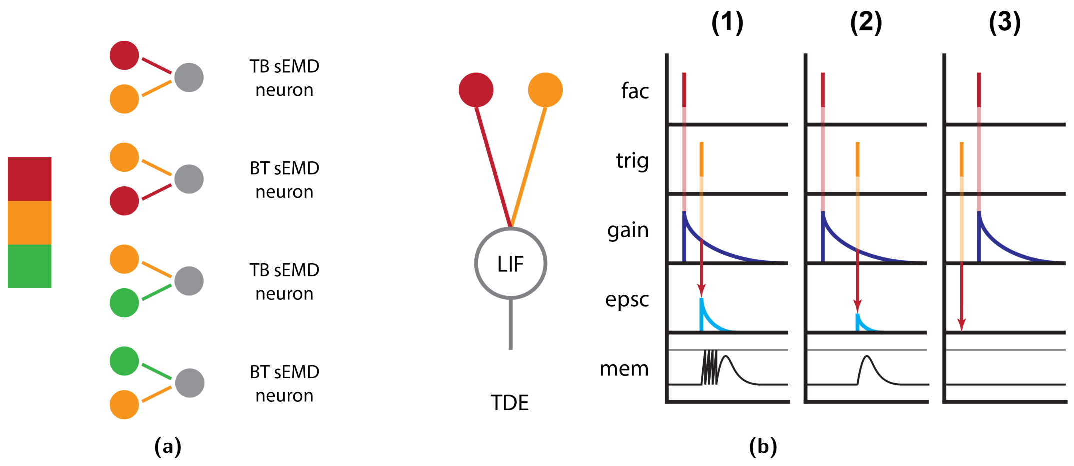


Figure 5.3: The sEMD as proposed in D'Angelo et al. (2020). **(a)** A vertical field of view consisting of three pixels. Neighboring pixels are connected by a sEMD. For all pairs of neighboring pixels there is a top-bottom (TB) detector and a bottom-top (BT) detector. **(b)** Internal functioning of the TDE. A spiking input to the facilitator (fac) results in an exponentially decaying gain. The EPSC is the product of the gain and the trigger (trig). The EPSC feeds into the LIF neuron. Three situation can occur: (1) the trigger receives an input spike right after the facilitator, resulting in a high EPSC; (2) the trigger recieves an input spike long after the facilitator, resulting in a low EPSC; (30) the trigger receives input *before* the facilitator, resulting in zero EPSC.

As the name suggests, the GRU network is a network consisting of 60 GRUs in a single hidden layer. The GRU, which is particularly suited for learning long-range dependencies in input data due to its use of update and reset gates to avoid the vanishing gradient problem, is anticipated to perform well on both moving pixel tasks. The GRU is regarded as a state-of-the-art architecture for processing sequences and serves as a benchmark for the highest possible performance on moving pixel tasks.

5.2.2 SNN & Leaky ANN

In addition to the non-spiking neural networks previously mentioned, four pairs of equivalent spiking/non-spiking networks are trained on the moving pixel tasks. The basic feedforward SNN and its non-spiking counterpart, the leaky feedforward ANN, encompass the first of these four pairs. Due to the fact that spiking neurons and leaky artificial neurons have an internal state, it is anticipated that they will be able to make a meaningful, albeit imprecise, velocity estimate in the case of the classic moving pixel task. The SNN and LeakyANN are only capable of temporal integration by decaying the states of their neurons at different rates. Due to the fact that two timesteps of temporal integration are sufficient for estimating velocity in the classic task, it should still be able to make useful estimations. However, the event task with longer dependencies will likely prove to be extremely challenging.

The introduction of implicit temporal dynamics for the leaky, non-spiking neurons, follows the definition of Hagenaars et al. (2021). Their definition of the leaky artificial neuron is analogues to the dynamics of the spiking neuron, except that no reset mechanism is used and a non-spiking activation function is applied. By replacing the spiking activation with ReLu activation, the activation of any neuron becomes

$$y_i^k = \text{ReLU}\left(\alpha y_i^{k-1} + (1-\alpha)\sum_j W_{ij}^{\text{ff}} I_i^k\right), \tag{5.1}$$

where j and i denote presynaptic and postsynaptic neurons respectively, α is the neuron leak, k the timestep, and W^{ff} are the feedforward weights that are multiplied with the input signal I.

5.2.3 Leaky RNN & recurrent SNN

The recurrent SNN and leaky RNN are the the next pair of spiking/non-spiking networks. These networks also feature a single hidden layer containing sixty neurons. The literature has repeatedly demonstrated that the SNN with explicit recurrent connections is highly capable of temporal integration (Hagenaars et al., 2021; Huh & Sejnowski, 2017; Li et al., 2020; Xing et al., 2020; Yin et al., 2020). Despite the theoretical promise of excellent temporal integration capabilities, vanishing gradients could pose a problem when training these networks.

5.2.4 Delay ANN & Delay SNN

A spiking and non-spiking network pair utilising synaptic delays for temporal integration is also trained to solve the moving pixel task. The type of delay learning is delay selection (see Sec. 4.4.1). The maximum synaptic delay multiplied by the number of hidden layers determines the temporal receptive field of delay networks. Temporal dependencies in input data longer than this receptive field cannot be learned by the non-spiking delay network. In contrast, the spiking delay network can, in addition to the synaptic delays, use its spiking dynamics to further expand its temporal receptive field and still make accurate predictions when dependencies exceed the synaptic delays' receptive field. In order to study these effects, the temporal receptive field of the synaptic delays is set to be sufficient for the classic task but insufficient for the event task. Under the latter conditions, the delay ANN can only observe the greater speeds, and its performance should suffer. The delay SNN should be less affected due to its spiking dynamics.

5.2.5 sEMD & LIFsEMD

The fourth pair of networks consists of two variants of the spiking elementary motion detector (sEMD). The sEMD was originally designed to encode optic flow using event-based visual sensors (D'Angelo et al., 2020). [1] The principle of the sEMD is depicted in Figure 5.3. The edge of an object moving in the field of view from one pixel to the next one generates a spike in each pixel with a given time difference. This time difference depends on the velocity of the edge (and the distance from the pixels). An sEMD is composed of two pixels and a time difference encoder (TDE). What the TDE essentially does is converting a time interval between two spikes into a number of output spikes generated in response to the second input spike. There are two types of synapses connecting the inputs to the TDE: facilitator and trigger. The facilitator synapse regulates the TDE neuron's activity. Only if the trigger synapse's input event occurs after the event from the facilitator synapse does the trigger synapse induce a response from the TDE neuron (compare (1) and (2) in Figure 5.3b). As depicted in Figure 5.3b, the output current of the trigger synapse raises the membrane potential of the TDE neuron. The strength of the current is dependent on the gain variable of the facilitator synapse, which decays exponentially.

A limitation of the sEMD is that it requires a minimum temporal resolution of the input data in order for edges not to move more than one pixel per timestep. If this does occur, the sEMD will either detect motion events incorrectly or not at all. This presents a problem for learning the classic moving pixel task, as the pixel can sometimes move multiple pixels from input to input. This is not a problem for the event task because the pixel never jumps more than one pixel from input to input.

[1] Even though the 's' in the abbreviation sEMD denotes 'spiking', in this work the 'sEMD' is considered as the non-spiking variant and the LIFsEMD as the spiking variant.

6 Results & Discussion

Here the results of the preliminary evaluation of memory mechanisms in (spiking) neural networks are presented. For this evaluation, ten networks capable of integrating temporal information, and one that is not, have been trained to solve two variants of a moving pixel task. This chapter is built up as follows. First the network parameters and simulation settings are discussed in Section 6.1. After that, the training process and the final accuracy of the networks on the classic and event task are assessed in Section 6.2 and Section 6.3, respectively. Following this general analysis, Section 6.4, Section 6.5 and Section 6.6 will dive deeper under the hood of these networks and explore the temporal integration mechanisms that arise as a result of training by back propagating errors through time.

6.1 Network Parameters & Simulation Settings

All simulations have for this preliminary evaluation have been done in PyTorch. The networks are trained using the Adam optimizer, for 100 epochs. To mitigate problems associated with exploding gradients, which is of particular concern for recurrent networks, the gradient norm of all parameters is clipped at 10.

Initialisation of the neuron leaks is done by applying the sigmoid $\alpha = \frac{1}{1+\exp(-a)}$ to values a, drawn from a normal distribution centred around -4. This results in small initial leak and thus in little initial 'memory', encouraging the spiking neurons to learn such behaviour instead of giving it from the start. In addition, during every forward pass the leaks are squished between 0 and 1 using the sigmoid function in order to prevent instability (Fang et al., 2021). The thresholds θ are clamped to $[0.05, \rightarrow)$ as implemented in Hagenaars et al. (2021). The thresholds are initialised by drawing from a normal distribution centred around 0.8. To initialise the weights in the spiking layers, values are drawn from the uniform distribution which is delimited by the square root of the number of input neurons to ensure enough activity. No biases are included (Hagenaars et al., 2021).

Table 6.1: SNN parameter initialisation

	Init
a	$\mathcal{N}(-4, 0.1)$
θ	$\mathcal{N}(0.8, 0.01)$
weights	$\mathcal{U}(-\frac{1}{\sqrt{N_{in}}}, \frac{1}{\sqrt{N_{in}}})$
biases	None
arctan′ width	10
Grad Norm	10

Table 6.2: Simulation parameters

Epochs	100
Optimizer	Adam
Learning Rate	0.002
Batch size	1

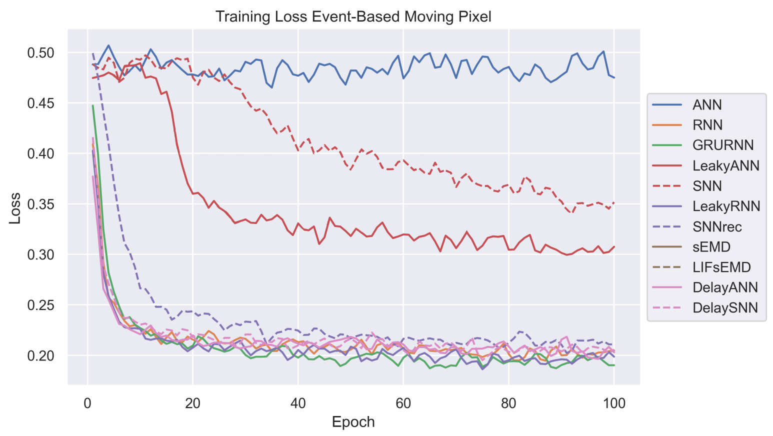


Figure 6.1: Training loss for the classic moving pixel task. For better readability the running average of the loss of two subsequent epochs are plotted. Artificial (non-spiking) networks and their spiking counterparts have the same colour. The non-spiking networks are plotted as solid lines and the spiking networks as dashed lines.

Table 6.3: Test loss for the classic moving pixel dataset. Evaluated with seed over 600 sequences.

	$\mathcal{L}_{\textbf{test}}$
ANN	0.48
RNN	0.21
GRU	**0.20**
LeakyANN	0.30
SNN	0.36
LeakyRNN	0.21
SNNrec	0.22
sEMD	0.35
LIFsEMD	0.39
DelayANN	0.21
DelaySNN	0.21

6.2 The Classic Moving Pixel

From Figure 6.1 and Table 6.3, we can conclude that, except the ANN, all networks were able to learn the moving pixel task. This is to be expected given that the ANN is the only network incapable of temporal integration. In arranging the networks, we find that explicit recurrency and delays perform the best. These are followed by the sEMD networks, which perform marginally poorer. The networks that rely solely on leaks to achieve temporal integration have the poorest performance (but have still learned something). Note that the temporal receptive field of the delay networks is large enough to cover all time dependencies in the input data.

It is notable that the sEMD and LIFsEMD networks start converging significantly faster than their purely feedforward competitors, the LeakyANN and the SNN. Ultimately however, they don't achieve as low of a test loss. Remember that by definition, the sEMD networks are not fully connected, but the input neurons are neighbour connected. The smaller number of parameters, in conjunction with the automatic detection of a motion direction, likely makes it easier for the network to determine the initial direction of the negative gradient. Eventually, this initial advantage becomes a limitation: due to the smaller solution space (only movements of at most 1 pixel), the accuracy cannot be improved after approximately 20 epochs.

Figure 6.1 also reveals that, for each non-spiking and spiking network pair, the final test loss of the spiking network is either greater than or equal to that of the non-spiking network. This observation is consistent with previous reports in the literature by, among others, Hagenaars et al. (2021), Kugele et al. (2020) and Xing et al. (2020). In addition to having comparable or slightly inferior performance, spiking networks also converge more slowly than their non-spiking counterparts. This is the case at least for three out of four pairs. The delay networks however convergence approximately at the same rate. Both reach 80% of their final loss after just two epochs. For comparison: the LeakyRNN reaches 80% of its final loss after three epochs versus six for its spiking counterpart the recurrent SNN. It appears as if explicit recurrency increases the difference in convergence speed between non-spiking to spiking neurons. An explanation for this might be that the recurrency emphasizes the already more difficult flow of gradients in spiking networks. The lengthy BPTT derivative chain causes gradients of less active neurons, which are already small, to decrease even further until they eventually vanish.

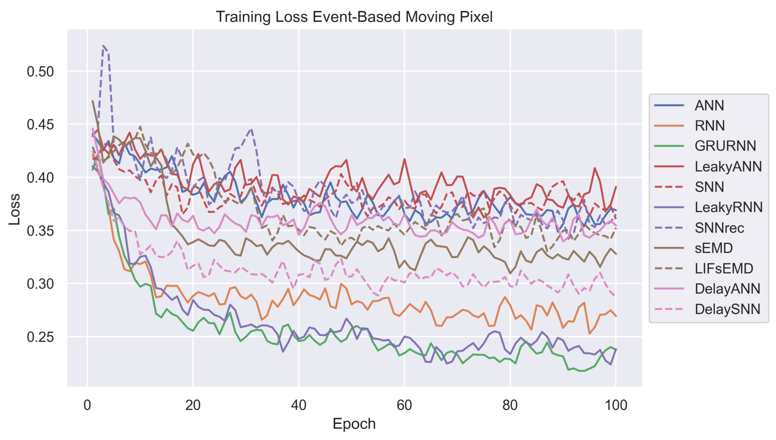


Figure 6.2: Training loss for the event-based 1D moving pixel task. For better readability the running average of the loss of two subsequent epochs are plotted. Artificial (non-spiking) networks and their spiking counterparts have the same colour. The non-spiking networks are plotted as solid lines and the spiking networks as dashed lines.

Table 6.4: Test loss for the event-based moving pixel dataset. Evaluated with seed over 600 sequences.

	$\mathcal{L}_{\text{test}}$
ANN	0.35
RNN	0.28
GRU	**0.22**
LeakyANN	0.35
SNN	0.38
LeakyRNN	0.23
SNNrec	0.37
sEMD	0.32
LIFsEMD	0.35
DelayANN	0.34
DelaySNN	0.29

6.3 The Event-Based Moving Pixel

Figure 6.2 displays the training loss curves for the event-based 1D moving pixel task. Remember that the temporal dependencies are longest in this dataset when the pixel speed is lowest. In this case, at lowest pixel speed ± 1, the minimum memory required is four simulation steps. In order to encourage the use of leaks in the spiking delay network, the maximum synaptic delay for the DelayANN and DelaySNN was set to 2.

In addition to the networks clearly finding it more difficult to estimate the correct velocity with these longer temporal dependencies, it is quite remarkable that the memoryless ANN and the stateless Delay-ANN (without sufficient temporal receptive field) are still able to learn the task to some degree. On the surface, this does not make sense. These networks have no (or insufficient) knowledge of the past to make an accurate estimate. To determine what is occurring, we revisit the images of some realisations of the event-based moving pixel task, depicted in Figure 5.1b. A more detailed analysis clarifies the situation at hand. Each time the moving pixel triggers an event, the events come twofold: one positive and one negative. In this extremely basic 1D setting, this pair of events gives away the direction of motion, which is always from the negative to the positive event. The reason for this is that the moving pixel is darker than the background. If the pixel travels upwards, this results in a negative event with a positive event above it. Despite never actually integrating input information or learning to estimate the exact pixel velocity, the ANN and DelayANN can minimize the loss function to a certain degree by reacting to a pair of positive and negative events within the same simulation time step.

Moreover, Figure 6.2 demonstrates how spiking dynamics can compensate for a lack of temporal receptive field produced by synaptic delays. This follows from the fact that the pink dashed line of the DelaySNN is notably below the solid pink line of the DelayANN. In a setting where the temporal receptive field of the synaptic delays matches up to the largest time dependencies in the input data, the DelaySNN performs on par with the DelayANN. However, when the temporal receptive field of the synaptic delays does not match the longest dependencies in the data, the DelaySNN is substantially more accurate than the DelayANN.

Turning our attention to the SNN and its non-spiking analogue the LeakyANN, it becomes evident that spiking and leak dynamics have considerable difficulty learning the task's longer dependencies. They even perform worse than the ANN. This is somewhat surprising, as the additional memory capacity associated with spiking neurons and artificial leaks should theoretically result in slightly improved performance. What has happened here is that the aforementioned phenomenon, in which events in a single step already reveal the direction of motion, has distracted the SNN and the LeakyANN from their true learning objective. Further examination reveals that all membrane leaks of the trained models are approximately equal to $\alpha = 0$, indicating that no memorization occurred. By setting all leaks to zero, the networks found a clever way to respond quickly to single frames revealing the direction of motion with two events. After reaching this local minimum of the loss function, the networks are unable to approach the global minimum of the loss function.

Also intriguing is the fact that both sEMD and LIFsEMD outperform SNN and LeakyANN. All of these networks are purely feedforward, although the EMD networks are only connected to their neighbours, unlike the SNN and LeakyANN which are fully connected. In this instance, reducing the dimensionality of the solution space actually improves the final performance. By design, the TDEE in the sEMD converts an interval between two spikes in adjacent pixels into a specific number of output spikes. Due to the fact that for this event task, events never skip a pixel, this neighbour connectivity constraint actually benefits the network rather than constraining it. The noise-resilience of the sEMD is another factor contributing to its superior performance. Only if a trigger input is received *after* a facilitator input in an adjacent pixel is an EPSC generated. As noise is random, this dramatically reduces the likelihood of noise occurrences infiltrating the network.

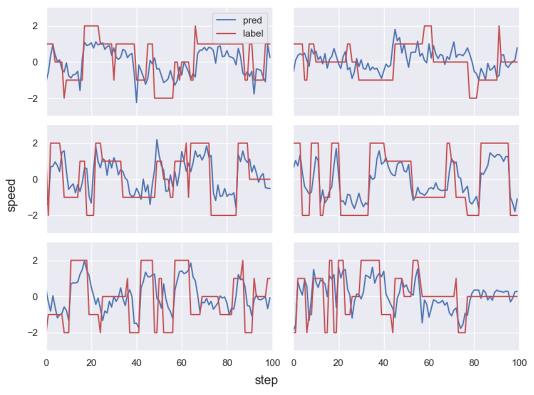
Finally, we observe that the recurrent SNN is an outlier. Its non-spiking counterpart, the LeakyRNN, has remarkable accuracy; therefore, based on what we have observed thus far, the recurrent SNN should be capable of approaching this level of performance. The training loss curve contains some unusual peaks indicating that learning was very unstable. After retraining the recurrent SNN across a grid of various learning rates and gradient norm clipping points, we were able to achieve a minimum test loss of 0.30 (see Appendix A.3 for further details). The fact that this network initially failed to converge illustrates clearly the challenges involved with back propagating errors in recurrently connected SNNs.

6.4 Analysis of Leaks in LeakyANN and SNN

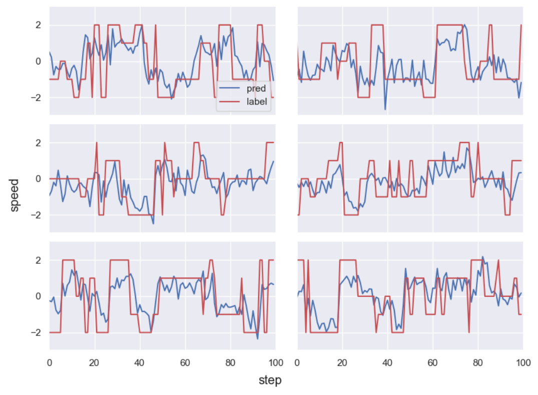
Figure 6.3 displays qualitative results and an analysis of the leakage for the LeakyANN (top row) and the SNN (bottom row). The qualitative results (Fig. 6.3a & Fig. 6.3b) illustrate the estimated pixel velocities for four test samples. The blue line indicates the estimation, while the red line represents the true velocity. It is clear that both networks are capable of producing meaningful velocity estimates. However, it should also be mentioned that the estimate is extremely volatile. This is unsurprising, given that the only form of temporal integration is leakage, and we have seen that the accuracy of such networks is significantly poorer than that of networks with, for instance, recurrency or delays.

To gain insight into how a network with leaks integrates information, we begin by creating a histogram of the hidden layer leaks (red histograms in Fig. 6.3e and Fig. 6.3f for the non-spiking and spiking, respectively). At first look, it appears that the distribution of leaks in the LeakyANN is significantly different from the distribution of leaks in the SNN. For the non-spiking leak network, leaks are bimodally distributed around $\alpha = 0$ and $\alpha = 0.4$, but for the SNN, leaks are bimodally distributed around $\alpha = 0$ and $\alpha = 0.8$ (remember: $\alpha \to 0$ equals high leakage and $\alpha \to 1$ equals no leakage). The bimodality is not unexpected because the pixel velocity can be approximated by comparing two snapshots of the input taken at separate times but at the same network time point. Consequently, it makes sense that the network's leaks generate two unique timescales: one timescale generates memory neurons (smaller leak) while the other generates fast neurons (large leak). The memory neurons store input data from prior times for comparison with the current activations of fast neurons. Why do the leaks of the spiking and non-spiking leak networks differ so drastically? The answer to this question can be found in the neural activation function definition. Non-spiking neurons do not possess a reset mechanism to "clear their memory" once a spike is released, whereas spiking neurons do. To avoid being stuck with the first input it receives for many time steps afterward, non-spiking neuron need to find a balance between remembering and forgetting, and this balance turns out to lie around a leak of $\alpha = 0.4$.

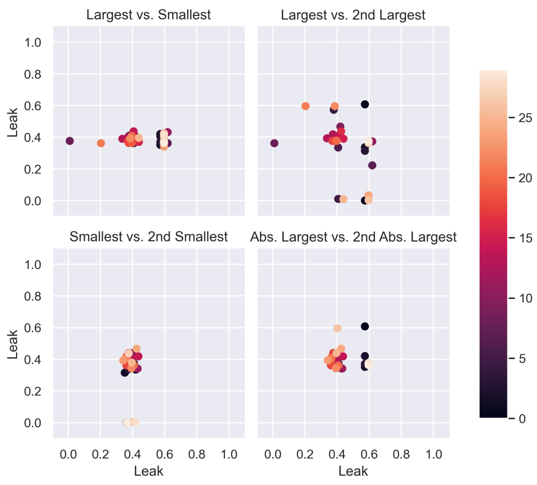
To test the claim that leak networks route the activation of each input pixel to a memory neuron and



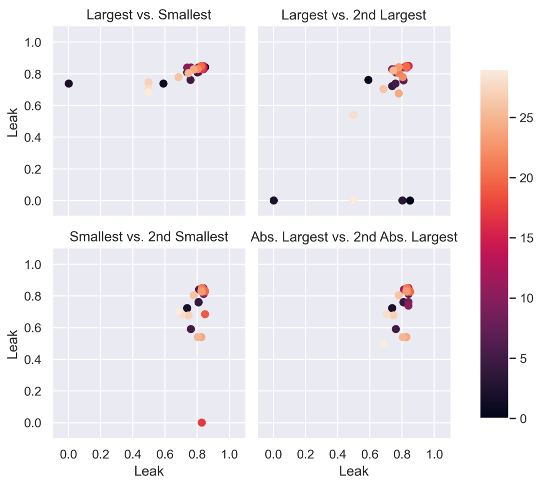
(a) ANN



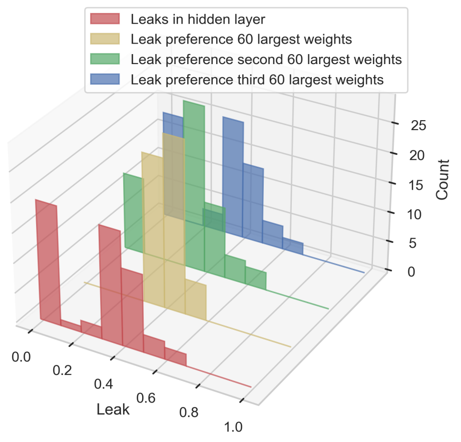
(b) SNN



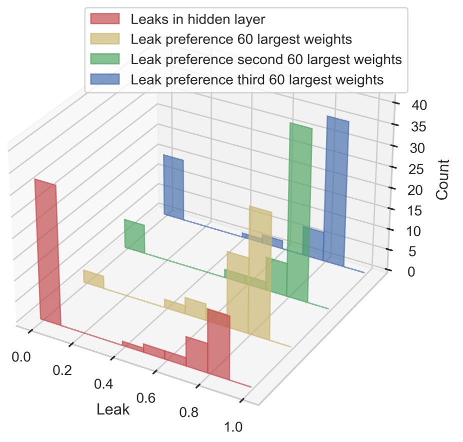
(c) ANN



(d) SNN

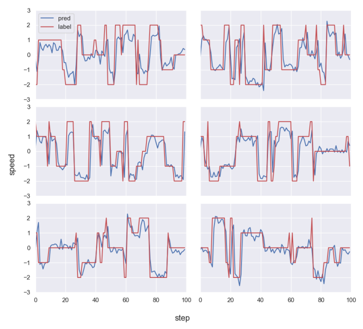


(e) ANN

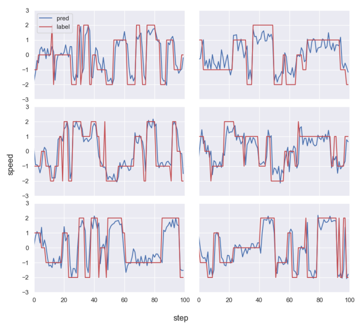


(f) SNN

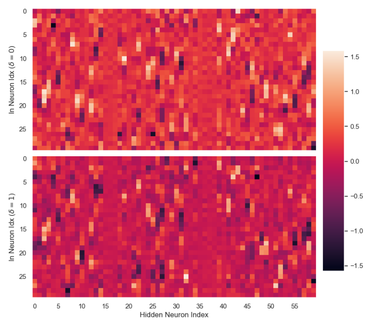
Figure 6.3: **(a)** Pixel velocity estimate of the LeakyANN. The red line represents the true velocity and the blue line is the predicted velocity. **(b)** Pixel velocity estimate of the SNN. **(c)** Scatter plot of the two preferred leaks in the hidden layer, per input pixel of the LeakyANN; 'preferred' in this context meaning: connections with largest weight. Colour represents index of input neuron. **(d)** Scatter plot of preferred hidden leaks, per input pixel of the SNN. **(e)** Histogram of the leaks in the hidden layer (red) of the LeakyANN and histograms of the preferred leaks by strength of the connection (yellow, green, blue). **(f)** Histogram of hidden leaks and preferred hidden leaks of the SNN.



(a)



(b)



(c)

Figure 6.4: (a) Pixel velocity estimate of the DelayANN. **(b)** Pixel velocity estimate of the DelaySNN. **(c)**

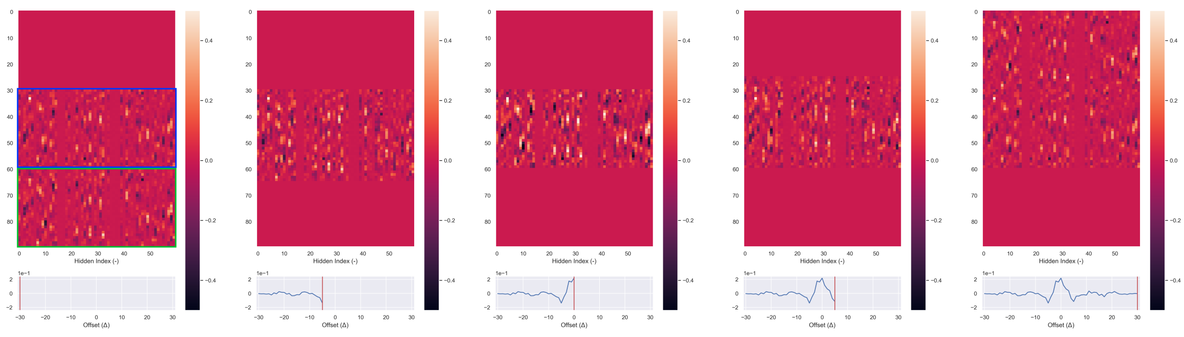
a fast neuron, scatter plots are created to examine the leaks. Specifically, the leaks associated with the largest weight of each input pixel. On the basis of the hypothesis, one would anticipate that the majority of points would be located near $(\alpha_1, \alpha_2) = (0, 0.4)$ for the LeakyANN and $(\alpha_1, \alpha_2) = (0, 0.8)$ for the SNN, as these correspond to differing timescales. However, looking at the scatter plots in Figure 6.3c (non-spiking) and in Figure 6.3b (spiking) this turns out not to be the case. Almost all combinations are situated at $0.4, 0.4$ and $0.8, 0.8$ for LeakyANN and SNN, respectively. Apparently, the claim was too easy. To determine what actually occurs in the leak networks, we examine the distribution of the preferred leaks for various weight sizes. These distributions are plotted in Figure 6.3e and 6.3f in yellow, green, and blue. The yellow histogram represents the leaks associated with the sixty largest weights of the hidden layer, the green histogram represents the leaks associated with the second sixty largest weights of the hidden layer, and the blue histogram represents the leaks associated with the third sixty largest weights of the hidden layer. The largest weights (yellow histograms) have a strong bias for memory neurons, which follows from larger bins close to $\alpha = 1$ and absence of bins near $\alpha = 0$. In the non-spiking leak network, none of the sixty largest weights are connected to a fast neuron. Upon further consideration, this makes perfect sense. At the time of pixel velocity estimation, the prediction layer requires neural activations from memory neurons and fast neurons. At that moment, however, contributions to a memory neuron from an earlier input will have leaked, whereas contributions to a fast neuron from the current time step have not yet leaked. To have equal sized contributions in the output layer, contributions to the memory neurons therefore require a larger weight than contributions to fast neurons.

6.5 Delays Analysis

The qualitative results of the non-spiking and spiking delay networks are shown in Figure 6.4a and 6.4b, respectively. The velocity traces appear reasonably accurate, as reflected by the relatively low test loss in Table 6.3. However, these qualitative results do not reveal a great deal of useful information and are consequently less intriguing as an analytical subject. Instead, we concentrate on the internals of the networks and how they accomplish temporal integration.

To understand how these delay networks integrate information, the weights for the different synaptic delays are compared and the membrane leaks of the spiking neurons are analysed. Starting with the latter, what became instantly clear was that all spiking neurons had α very close to 0, meaning no memory of the spiking neurons is being exploited. This is due to the fact that the network's synaptic delays already give it sufficient temporal integration capacity for the task at hand. Therefore, it becomes redundant to leverage the spiking neuron's dynamics. Realizing this, the DelaySNN must in this context be viewed as a stateless delay network with a spiking activation function and a reset mechanism. The emphasis is therefore shifted to how the weights "choose" particular delays to arrive at an estimate of velocity. To facilitate a fair comparison between spiking and non-spiking, sigmoid activation was used for this analysis in the DelayANN This corresponds most closely to a spiking activation.

We start by examining the weights associated with a delay of $\delta = 0$ and $\delta = 1$ separately, in the



(a) $\tau = -30$ **(b)** $\tau = -5$ **(c)** $\tau = 0$ **(d)** $\tau = +5$ **(e)** $\tau = +30$

Figure 6.5: Weight matrices of the delay layer plotted as pixels in an image. The colour of the pixel corresponds to the size of the weight. The weights of synapses with delay 0 (blue stroke in (a)) are kept centred. The weights of synapses with delay 1 (green stroke in (a)) slide over the weights of synapses with delay 0. The plotted line beneath each image corresponds to the squared difference (Eq. 6.1) between the two weight matrices for the given offset. The x-axis of the image corresponds to the index of the hidden neuron.

DelayANN. The weight matrices are plotted as an image in Figure 6.4c, where the colour of the pixel located at (i, j) corresponds to the size of the weight $w_{i,j}$ that connects input neuron i to hidden neuron j. Because the number of input neurons (field of view) is 30 and the number of hidden neurons is 60, both images have size $(30, 60)$. Notable is that the size of the weights of the hidden neurons varies significantly. For some neurons, nearly all synaptic weights are close to zero (low efficacy), whereas others have numerous synaptic connections with high efficacy. These synapses with higher efficacy also seem to be spatially correlated in the sense that if a synapse connecting neuron i to neuron j has high efficacy, then it appears likely that synapses in the vicinity $i - k < i < i + k$ that are also connected to hidden neuron j, have high efficacy too (for small k). This is best seen in 6.4c as the emergence of vertical pixel stripes. Moreover, this suggests that distinct hidden neurons detect motion in distinct regions of the input field of view. Having discovered this, it is still unclear how exactly these hidden neurons "detect" this motion. However, we have a hunch as to how this principle operates, as a closer examination of the weights for delays 0 and 1 in Figure 6.4c reveals that the weights of both delays appear large and small in exactly the same places. To determine whether this observation is accurate, the weight matrices for synaptic delay 0 and 1 are quantitatively compared.

The quantitative similarity analysis is done by sliding the weight matrices for the different synaptic delays over each other while computing the average correlation coefficient of the synaptic weights associated with a hidden neuron. Figure 6.5 depicts the results of this analysis for a moving pixel of width 2. The blue-framed pixels in **(a)** have synaptic delay of 0 and the green-framed pixels have synaptic delay 1. The offset ranges from no overlap in **(a)**, to full overlap in **(c)**, to no overlap again in **(e)**. The average correlation coefficient over the hidden neurons is calculated according to

$$c_{\text{ave}}[\tau] = \frac{1}{N_h} \sum_{j}^{N_h} \left(\sum_{i} \overline{w_j^k[i]} \cdot w_j^{k-1}[i + \tau] \right), \tag{6.1}$$

where w_j^k are the weights of the synapses with delay 0 connected to hidden neuron j, w_j^{k-1} are the weights of the synapses with delay 1 connected to hidden neuron j and c_{ave} is the average correlation coefficient. The values for the correlation coefficient are always in the interval $[-1, 1]$, with $+1$ meaning strong positive correlation, 0 meaning no correlation and -1 meaning strong negative correlation.

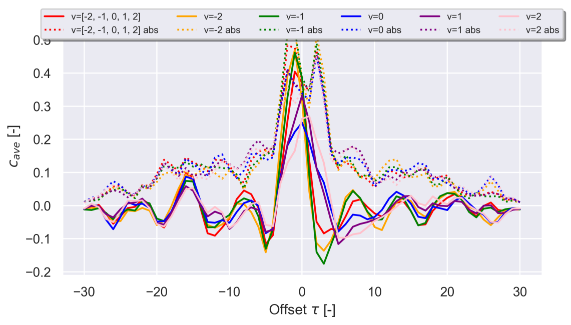
The curves underneath the images in Figure 6.5a t/m 6.5e represent the average correlation coefficient c_{ave} as a function of the offset τ. We notice that for certain τ there is indeed a degree of similarity between the weights of synaptic delay 0 and synaptic delay 1. It appears that the network is using its hidden neurons to select small portions of the input field of view for both delay 0 and 1, establishing a type of motion-detecting "gates". Distinct hidden neurons are covering distinct regions of the input vision field.

Next, we investigate whether these gates are sensitive to the width and velocity of the pixel and if so, which hidden neurons responds to which metric. Figure 6.6 displays four plots of the average correlation between the weights of synapses having delay $d = 0$ and delay $d = 1$ of the ten most active hidden neurons per pixel width and per pixel velocity. The top two plots are correlations for different speeds (Fig. 6.6a) and widths (Fig. 6.6c) in the DelayANN. The bottom two plots are correlations for different speeds (Fig. 6.6b) and widths (Fig. 6.6d) in the DelaySNN. The solid lines take into account the sign of the correlation when averaging over the ten most active neurons. Thus, five strongly positively correlated together with five strongly negatively correlated can result in net no correlation. In order to still be able to observe correlation if such a case occurs, the averages of the absolute values of the correlations have also been plotted with the dotted lines.

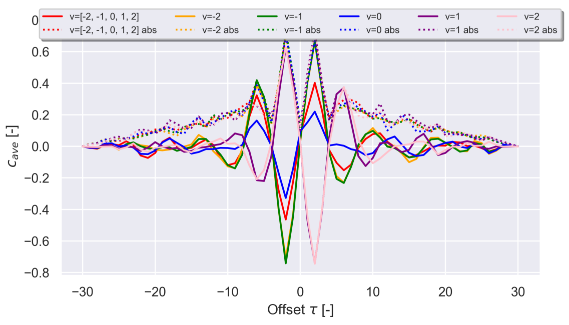
The first thing that attracts the eye is that Figure 6.6c looks to be missing several plot lines. However, this appearance is deceptive, as all lines are in fact plotted; some are just superimposed. That these curves overlap means that the DelayANN prefers the exact same hidden neurons for these widths. In other words, the width of the pixel has no impact on how the network estimates velocity. If we take the 10 most active neurons for each of the six different pixel widths, we find that just twelve distinct hidden neurons are utilised (where sixty would have been possible if each pixel width had used different hidden neurons). When we count the number of unique hidden neurons used if we take the ten most active for six different pixel widths, we see that only twelve different ones are used (where sixty would have been possible if each pixel width had used different hidden neurons). This observation is consistent with the correlation of the weights of the DelaySNN in 6.6d. The correlation curves are quite similar, indicating that the network chooses the same hidden neurons for estimating velocity across all pixel widths (only 15 used in total). Given that the width of the pixel has little to no effect on how the network determines the pixel velocity, we conclude that the network uses the pixel's edges to do so. This would be consistent with optical flow estimation theory, which implies that only velocity components perpendicular to a contrast gradient are observable.

Secondly, we see that the weights of the synaptic delays of the non-spiking (Fig. 6.6a) and spiking (Fig. 6.6b) delay network correlate very differently, for varying pixel velocities. The correlation between the synaptic delay weights of the DelayANN appears comparable for various pixel velocities and is also nearly symmetrical in the vertical axis. In contrast, the weights of the DelaySNN correlate considerably differently with different pixel velocities and are completely asymmetric in zero. The most notable aspect of these asymmetric correlations is that the spiking network for positive and negative velocities has completely opposing correlation curves (yellow and green lines vs. purple and pink lines in 6.6b), indicating that positive and negative pixel velocities use completely separate hidden neurons. Comparing this to the correlations for different pixel velocities of the DelayANN (Fig. 6.6a), we initially fail to see this tendency. Yet, on closer inspection, the same maxima at shifts $\tau = \pm 2$ are observable for the dotted lines, which arise when taking the absolute value of the correlation. Because the correlation in $\tau = 0$ does not change noticeably when absolute values are considered, this indicates that the signal contains asymmetrical, negatively correlated components.

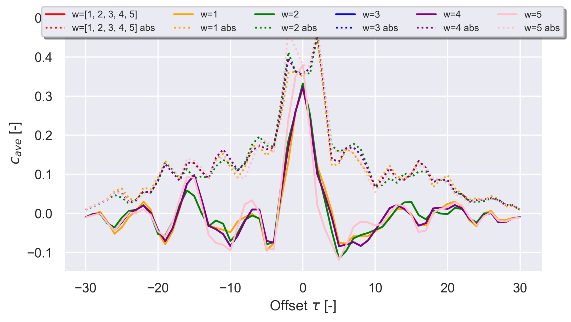
To get more insight into the difference between the delay ANN and SNN, we inspect the prediction weights associated with the most active hidden neurons, per pixel velocity in Figure 6.6e and 6.6f. The partly overlapping points denote the same hidden neuron, but for illustration purposes they are slightly shifted in the horizontal direction. We notice primarily two differences between the plots for the non-spiking (6.6e) and spiking (6.6f) delay networks. The first is consistent with what we have already deduced from the correlation plots, namely that the non-spiking network uses the same hidden neurons for both positive and negative pixel velocities, whilst the spiking network keeps them completely distinct. This follows from the fact that the warm and cold colours in Figure 6.6e of the ANN are mixed, while the warm and cold colours in Figure 6.6f of the SNN are separated completely. The second important difference is that the non-spiking network utilises both positive and negative prediction weights for positive and negative pixel velocities (warm and cold colours for positive and negative weight values), while the spiking network only employs positive prediction weights for positive pixel velocities and negative ones for negative velocities (warm colours all negative weight and cold colours all positive). The answer for this must lay in the difference in neuron activation. The spiking network



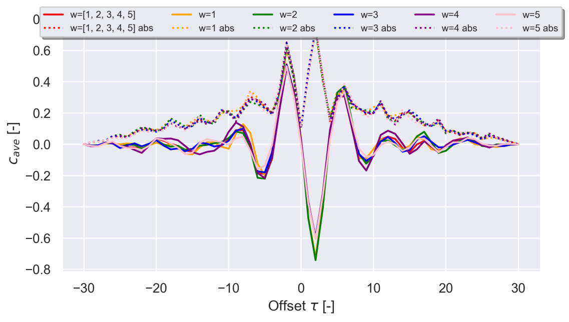
(a) ANN, differing velocities



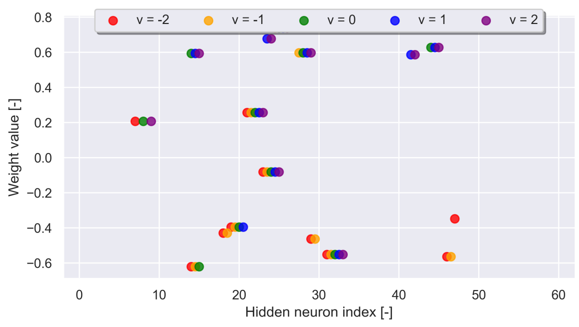
(b) SNN, differing velocities



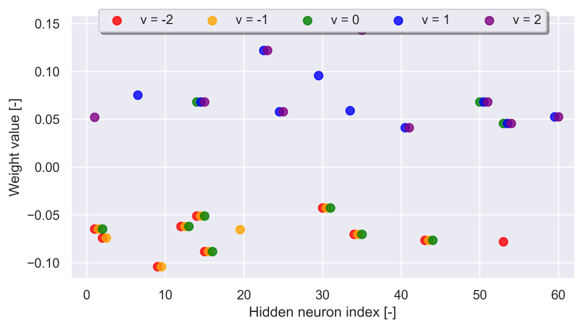
(c) ANN, differing widths



(d) SNN, differing widths



(e) ANN



(f) SNN

Figure 6.6: (a,b,c,d) Average correlation between the synaptic weights having delay $\delta = 0$ and $\delta = 1$, over the ten most active hidden neuron as defined by Equation 6.1. The solid lines take into account the sign of the correlation (positive and negative correlations can cancel each other out). The dotted lines take the absolute value of the correlation for each hidden neuron. **(e,f)** Prediction weights of the ten most active hidden neurons, per pixel velocity. Partly overlapping points correspond to the same hidden neuron, meaning that for different pixel velocities, the same hidden neuron is active.

can switch its prediction weights 'on' or 'off' by either exceeding or not exceeding a neuron's threshold. The non-spiking network lacks this ability. There will always be activation greater than zero (sigmoid after all) because subthreshold inputs cannot be suppressed. It therefore becomes much more difficult to take, for example, only a small selection of hidden neurons with positive prediction weights to estimate positive pixel velocity. Instead, the network opts for a linear combination of more and larger weights. By exchanging a small proportion of the most active neurons for varying pixel speeds, the correct speed can be estimated for all speeds. The spiking network only sums over a small number of weights of the same sign. The fact that the spiking network chooses a much simpler and more insightful approach is, on second thought, not that surprising, because the solution space of the spiking network is very limited. It can only turn on or off 60 weights, which equates to just $60^2 = 3600$ possible outcomes.

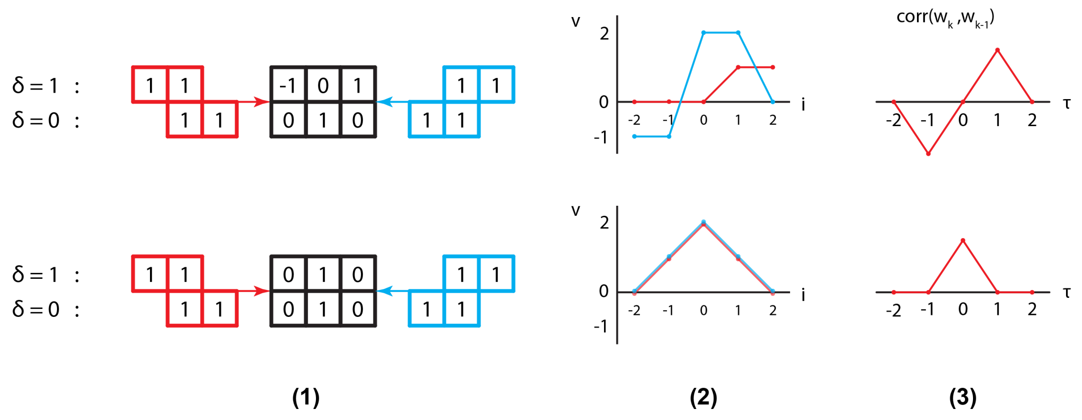


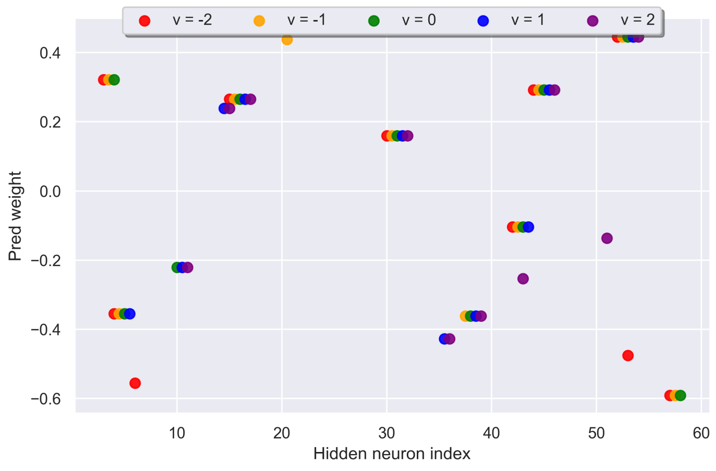
Figure 6.7: Two examples of how a pixel of width 2, moving through a part of a field of view, gives rise to a velocity estimate for two different sets of synaptic weights. **(1)** In red: pixel moving to the right with 1 px/timestep. In black: synaptic weights. In blue: same pixel moving to the left with 1 px/timestep. Synaptic delay weights with asymmetrical correlation **(3)** between them give rise to different activations **(2)** for different directions of motion. For synaptic delay weights with symmetrical correlation between them, the direction of motion does not change the activation.

Finally, to illustrate the importance of asymmetrically correlating weights between synaptic delay $\delta = 0$ and $\delta = 1$ in order for a hidden neuron to recognise the direction of motion, Figure 6.7 displays two very basic example of a pixel moving through a small part of a field of view. The synaptic weights of the top example in **(1)** correlate asymmetrically **(3)** and therefore the resulting activation is direction dependent. In contrast, the synaptic weights of the bottom example in **(1)** correlate symmetrically **(3)** and therefore the resulting activation is direction invariant. The latter situation is obviously problematic for the network and additional mechanisms are needed to determine the direction of motion.

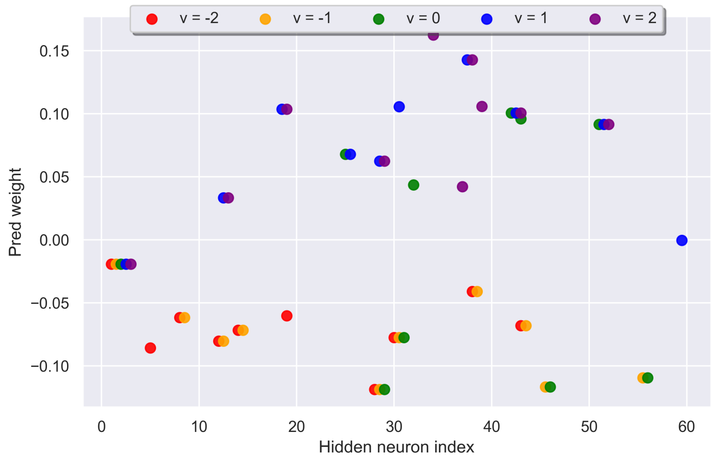
6.6 Explicit Recurrency

In line with our finding concerning the leaks in the delay networks, we notice that after training, all the leaks in both the LeakyRNN and the SNNrec have converged to zero, meaning no temporal integration is achieved using the spiking dynamics. Consequently, we regard the spiking network as a recurrent BNN and have opted for this analysis to use sigmoid activation in the non-spiking RNN.

Figure 6.8 displays the prediction layer weights of the ten most active hidden neurons, per pixel velocity. Overlapping points correspond to the same hidden neuron. Comparing Figures 6.8a and 6.8b of the recurrent networks with Figures 6.6e and 6.6f of the delay networks, we recognise exactly the same characteristics between non-spiking and spiking. For non-spiking the warm colours (negative velocity) and the cold colours (positive velocity) are shared and distributed over both positive and negative weights, while for spiking the warm colours are separated from the cold colours and have the same sign each time. This means that the spiking network prefers certain hidden neurons for positive velocities and others for negative velocities (and a mix of the two for velocity 0), while the non-spiking prefers the same neurons for multiple velocities, whether positive or negative. In addition, the size of the weights tends to be larger for the non-spiking network. The size of the weights for the spiking network can be smaller as most of the contributions to each estimation have the same sign.



(a)



(b)

Figure 6.8: The prediction weights of the ten most active hidden neurons, per pixel velocity for the DelayANN. Markers which partly overlap represent the same hidden neuron. For improved readability the markers are slightly jittered in the horizontal direction. **(a)** LeakyRNN **(b)** SNNrec

7 Conclusions

The goal of this preliminary evaluation of memory mechanisms in spiking neural networks was to get a grasp of the concepts and challenges associated with designing and training SNNs in a simulation environment. To that end, eleven neural network architectures were trained on two variations of an artificial 1D moving pixel dataset. The analysis can be divided into three distinct sections. The first section evaluated the performance of the networks on the classic moving pixel task. The second evaluated the performance of the networks on the event-based moving pixel task. The third section was an in-depth assessment of three pairs of spiking/non-spiking networks trained on the traditional moving pixel dataset. Following is a summary of the lessons learned from each of these sections of the analysis.

Based on the results obtained by training the networks on the classic moving pixel task we conclude that:

- Explicit recurrent connections and synaptic delays (provided that their temporal receptive field is sufficiently large to cover all time dependencies in the input data) reach comparable performance to the well known GRU architecture on the classic moving pixel dataset. This applies to both spiking and non-spiking.
- Spiking networks perform similarly or somewhat worse than their non-spiking counterparts. In addition, learning is slowed by the combination of spiking activation and recurrence.
- The neighbour connectivity of the sEMD prohibits it from reliably predicting velocity when motion exceeds 1 pixel every simulation time step, as is the case for this test.
- Training SNNs with SG decent does not promote the utilisation of membrane leaks. As long as recurrence and delays provide adequate temporal receptive field, spiking networks tend to maintain their leaks at 0.

Similarly, based on the outcomes of training the networks on the event-based moving pixel problem, we reach the following conclusions:

- In a simple 1D scenario, event-based observations of motion reveal the direction of motion within a single simulation time step. This creates the false impression that networks are integrating input data, but in reality they merely respond to current inputs.
- Increasing time dependencies from two time steps in the standard moving pixel task to four time steps in the event-based moving pixel job has a significant effect on both convergence speed during training and final accuracy.
- Due to its inherent characteristics, the sEMD is the only network that achieves greater accuracy on the event-based task than on the conventional task.

- When hyperparameters are not appropriately tuned, error backpropagation in SNNs with recurrent connections can be highly unstable.

Finally, based on a comprehensive analysis of spiking/non-spiking neural network pairs trained on the classic moving pixel task, we reach the following conclusions:

- A reset mechanism has a significant impact on the behaviour of leaks. Without a reset mechanism, a neuron is not capable of forgetting previous input. Consequently, the leaks only reach a maximum value of approximately $\alpha = 0.5$.
- Different hidden neuron detect different speeds in SNNs. The same is not true for ANNs. An SNN can simply 'switch' on and off the hidden neurons to come to a velocity estimation. Continuous activation requires an ANN to account for a great deal more hidden activation in order to make an accurate estimate.
- The activity of hidden neurons is pixel width invariant for both spiking and non-spiking. Just the pixel edges carry the information.

IV

Appendices

A Appendices

A.1 Network Parameters

Table A.1: Default hyperparameter values for the discrete simulation cases. Hyphens indicate no units, blanks indicate an irrelevant parameter for that case.

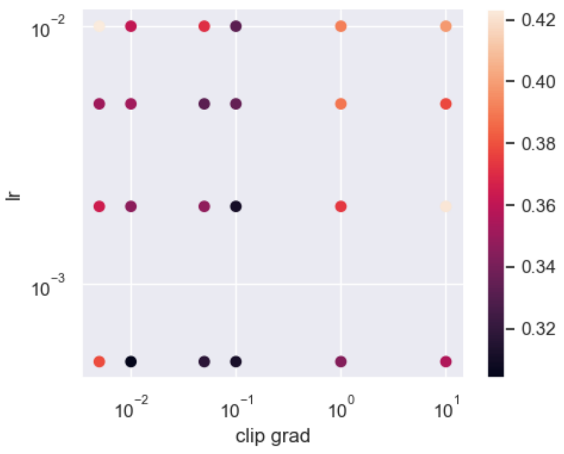
Net Params	**ANN**	**RNN**	**GRU**	**LeakyANN**	**LeakyRNN**	**SNN**	**SNNrec**	**sEMD**	**LIFsEMD**	**DelayANN**	**DelaySNN**
N_{in}	30	30	30	30	30	30	30	30	30	30	
N_{hidden}	60	60	60	60	60	60	60	58	58	60	60
activation	ReLU	ReLU	ReLU	ReLU	ReLU	Spiking	Spiking	ReLU	Spiking	ReLU	Spiking
N_{leaks}	-	-	-	60	60	60	60	-	-	-	60
reset	-	-	-	-	-	hard	hard	-	hard	-	hard

A.2 Spiking NN in Pytorch

Listing A.1: A very basic, single hidden layer, SNN implemented using Pytorch in Python.

```python
import torch
import torch.nn as nn
import torch.nn.functional as f


class SNN(nn.Module):
    def __init__(self, Ninput=2, Nhidden=4, Noutput=1):
        super().__init__()
        self.hidden = LinearLIF(Ninput, Nhidden)
        self.output = nn.Linear(Nhidden, Noutput)
        self.state  = None

    def forward(self, x):
        s_out, self.state = self.hidden(x, self.state)
        pred = self.output(s_out)
        return pred


class LinearLIF(nn.Module):
    def __init__(self, Nin, Nout):
        super().__init__()
        self.leak, self.thres = 0.5, 1
        self.ff = nn.Linear(Nin, Nout)
        self.spike_fn = self.ArctanSpike.apply

    def forward(self, x, prev_state):
        ff = self.ff(x)
        if prev_state is None:                            # generate if None
        ↪   provided
            prev_state = torch.zeros(2, *ff.shape)
        v, s = prev_state                                 # unstack output
        v_out = (1 - s) * self.leak * v + (1 - self.leak) * ff  # compute membrane
        ↪   potential
        s_out = self.spike_fn(v_out, self.thresh)         # compute output spikes
        return s_out, torch.stack([v_out, s_out])         # stack states

    @staticmethod
    class ArctanSpike(torch.autograd.Function):
        @staticmethod
        def forward(ctx, x, width=10):
            ctx.save_for_backward(x, width)
            return x.gt(0).float()

        @staticmethod
        def backward(ctx, grad_output):
            x, width = ctx.saved_tensors
            grad_input = grad_output.clone()
            sg = 1 / (1 + width * x * x)
            return grad_input * sg, None
```

A.3 Optimzing Hypers for SNNrec



Bibliography

Adibi, P., Meybodi, M. R., & Safabakhsh, R. (2005). Unsupervised learning of synaptic delays based on learning automata in an RBF-like network of spiking neurons for data clustering. *Neurocomputing*, *64*, 335–357. https://doi.org/10.1016/j.neucom.2004.10.111
22 citations (Semantic Scholar/DOI) [2022-09-29]

Amir, A., Taba, B., Berg, D., Melano, T., McKinstry, J., Di Nolfo, C., Nayak, T., Andreopoulos, A., Garreau, G., Mendoza, M., Kusnitz, J., Debole, M., Esser, S., Delbruck, T., Flickner, M., & Modha, D. (2017). A Low Power, Fully Event-Based Gesture Recognition System. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 7388–7397. https://doi.org/10.1109/CVPR.2017.781

Balasubramanian, V. (2021). Brain power. *Proceedings of the National Academy of Sciences*, *118*(32), e2107022118. https://doi.org/10.1073/pnas.2107022118
4 citations (Semantic Scholar/DOI) [2023-02-12]

Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., & Maass, W. (2018). Long short-term memory and learning-to-learn in networks of spiking neurons.

Benosman, R., Clercq, C., Lagorce, X., Ieng, S.-H., & Bartolozzi, C. (2014). Event-Based Visual Flow. *IEEE Transactions on Neural Networks and Learning Systems*, *25*(2), 407–417. https://doi.org/10.1109/TNNLS.2013.2273537

Blouw, P., Choo, X., Hunsberger, E., & Eliasmith, C. (2019). Benchmarking Keyword Spotting Efficiency on Neuromorphic Hardware. https://doi.org/10.48550/arXiv.1812.01739

Bohr, M. (2007). A 30 Year Retrospective on Dennard's MOSFET Scaling Paper. *IEEE Solid-State Circuits Society Newsletter*, *12*(1), 11–13. https://doi.org/10.1109/N-SSC.2007.4785534

Bohte, S. (2011). Error-Backpropagation in Networks of Fractionally Predictive Spiking Neurons. *6791*, 60–68. https://doi.org/10.1007/978-3-642-21735-7_8
28 citations (Semantic Scholar/DOI) [2022-10-10]

Bohte, S. M., Kok, J. N., & La Poutré, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, *48*(1), 17–37. https://doi.org/10.1016/S0925-2312(01)00658-0
753 citations (Semantic Scholar/DOI) [2022-09-13]

Booij, O., & tat Nguyen, H. (2005). A gradient descent rule for spiking neurons emitting multiple spikes. *Information Processing Letters*, *95*(6), 552–558. https://doi.org/10.1016/j.ipl.2005.05.023
154 citations (Semantic Scholar/DOI) [2022-09-13]

Bouvier, M., Valentian, A., Mesquida, T., Rummens, F., Reyboz, M., Vianello, E., & Beigne, E. (2019). Spiking neural networks hardware implementations and challenges: a survey. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, *15*(2), 1–35.

Caporale, N., & Dan, Y. (2008). Spike timing-dependent plasticity: a Hebbian learning rule. *Annual Review of Neuroscience*, *31*, 25–46. https://doi.org/10.1146/annurev.neuro.31.060407.125639

Cho, K., van Merrienboer, B., Bahdanau, D., & Bengio, Y. (2014). On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. https://doi.org/10.48550/arXiv.1409.1259
4446 citations (Semantic Scholar/arXiv) [2022-10-06]

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. `https://doi.org/10.48550/arXiv.1412.3555`

Cramer, B., Stradmann, Y., Schemmel, J., & Zenke, F. (2022). The Heidelberg spiking datasets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *33*(7), 2744–2757. `https://doi.org/10.1109/TNNLS.2020.3044364`

D'Angelo, G., Janotte, E., Schoepe, T., O'Keeffe, J., Milde, M. B., Chicca, E., & Bartolozzi, C. (2020). Event-Based Eccentric Motion Detection Exploiting Time Difference Encoding. *Frontiers in Neuroscience*, *14*, 451. `https://doi.org/10.3389/fnins.2020.00451`

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., Dimou, G., Joshi, P., Imam, N., Jain, S., Liao, Y., Lin, C.-K., Lines, A., Liu, R., Mathaikutty, D., McCoy, S., Paul, A., Tse, J., Venkataramanan, G., … Wang, H. (2018). Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro*, *38*(1), 82–99. `https://doi.org/10.1109/MM.2018.112130359`

Davies, M., Wild, A., Orchard, G., Sandamirskaya, Y., Guerra, G. A. F., Joshi, P., Plank, P., & Risbud, S. R. (2021). Advancing Neuromorphic Computing With Loihi: A Survey of Results and Outlook. *Proceedings of the IEEE*, *109*(5), 911–934. `https://doi.org/10.1109/JPROC.2021.3067593`

DeWolf, T., Jaworski, P., & Eliasmith, C. (2020). Nengo and Low-Power AI Hardware for Robust, Embedded Neurorobotics. *Frontiers in Neurorobotics*, *14*.

Dupeyroux, J., Hagenaars, J., Paredes-Vallés, F., & de Croon, G. (2020). Neuromorphic control for optic-flow-based landings of MAVs using the Loihi processor. `https://doi.org/10.1109/ICRA48506.2021.9560937`

Esser, S. K., Merolla, P. A., Arthur, J. V., Cassidy, A. S., Appuswamy, R., Andreopoulos, A., Berg, D. J., McKinstry, J. L., Melano, T., Barch, D. R., di Nolfo, C., Datta, P., Amir, A., Taba, B., Flickner, M. D., & Modha, D. S. (2016). Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences*, *113*(41), 11441–11446. `https://doi.org/10.1073/pnas.1604850113`

Fang, W., Yu, Z., Chen, Y., Masquelier, T., Huang, T., & Tian, Y. (2021). Incorporating Learnable Membrane Time Constant to Enhance Learning of Spiking Neural Networks
74 citations (Semantic Scholar/arXiv) [2022-10-06].

Fischer, V., Köhler, J., & Pfeil, T. (2018). *The streaming rollout of deep networks - towards fully model-parallel execution* (arXiv:1806.04965). `https://doi.org/10.48550/arXiv.1806.04965`
14 citations (Semantic Scholar/arXiv) [2022-09-22]

Frémaux, N., & Gerstner, W. (2016). Neuromodulated Spike-Timing-Dependent Plasticity, and Theory of Three-Factor Learning Rules. *Frontiers in Neural Circuits*, *9*
248 citations (Semantic Scholar/DOI) [2022-09-9].

Furber, S. (2016). Large-scale neuromorphic computing systems. *Journal of Neural Engineering*, *13*(5), 051001. `https://doi.org/10.1088/1741-2560/13/5/051001`
162 citations (Semantic Scholar/DOI) [2022-10-18]

Furber, S. B., Galluppi, F., Temple, S., & Plana, L. A. (2014). The SpiNNaker Project. *Proceedings of the IEEE*, *102*(5), 652–665. `https://doi.org/10.1109/JPROC.2014.2304638`

Gallego, G., Delbruck, T., Orchard, G., Bartolozzi, C., Taba, B., Censi, A., Leutenegger, S., Davison, A., Conradt, J., Daniilidis, K., & Scaramuzza, D. (2019). Event-based Vision: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*(1), 154–180. `https://doi.org/10.1109/TPAMI.2020.3008413`
490 citations (Semantic Scholar/arXiv) [2022-10-11]

Gardner, B., Sporea, I., & Grüning, A. (2015). Learning Spatiotemporally Encoded Pattern Transformations in Structured Spiking Neural Networks. *Neural Computation*, *27*(12), 2548–2586.

https://doi.org/10.1162/NECO_a_00790
21 citations (Semantic Scholar/DOI) [2022-09-15]

Gehrig, M., Shrestha, S. B., Mouritzen, D., & Scaramuzza, D. (2020). Event-Based Angular Velocity Regression with Spiking Networks. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 4195–4202. https://doi.org/10.1109/ICRA40945.2020.9197133
33 citations (Semantic Scholar/DOI) [2022-12-15]

Gerstner, W., & Kistler, W. M. (2002). *Spiking neuron models: single neurons, populations, plasticity.* Cambridge university press.

Gerstner, W., Kistler, W. M., Naud, R., & Paninski, L. (2014). *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition* (1st ed.). Cambridge University Press
702 citations (Semantic Scholar/DOI) [2022-07-29].

Ghosh-Dastidar, S., & Adeli, H. (2007). Improved spiking neural networks for EEG classification and epilepsy and seizure detection. *Integrated Computer-Aided Engineering*, *14*(3), 187–212. https://doi.org/10.3233/ICA-2007-14301
325 citations (Semantic Scholar/DOI) [2022-09-29]

Ghosh-Dastidar, S., & Adeli, H. (2009). A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection. *Neural Networks: The Official Journal of the International Neural Network Society*, *22*(10), 1419–1431. https://doi.org/10.1016/j.neunet.2009.04.003
388 citations (Semantic Scholar/DOI) [2022-09-29]

Gollisch, T., & Meister, M. (2008). Rapid Neural Coding in the Retina with Relative Spike Latencies. *Science*, *319*(5866), 1108–1111. https://doi.org/10.1126/science.1149639
548 citations (Semantic Scholar/DOI) [2022-07-27]

Göltz, J., Kriener, L., Baumbach, A., Billaudelle, S., Breitwieser, O., Cramer, B., Dold, D., Kungl, A. F., Senn, W., Schemmel, J., Meier, K., & Petrovici, M. A. (2021). Fast and energy-efficient neuromorphic deep learning with first-spike times
29 citations (Semantic Scholar/arXiv) [2022-10-07].

Graves, A. (2014). Generating Sequences With Recurrent Neural Networks. https://doi.org/10.48550/arXiv.1308.0850

Guo, W., Fouda, M. E., Eltawil, A. M., & Salama, K. N. (2021). Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems. *Frontiers in Neuroscience*, *15*
12 citations (Semantic Scholar/DOI) [2022-07-26].

Hagenaars, J., Paredes-Vallés, F., & de Croon, G. (2021). *Self-Supervised Learning of Event-Based Optical Flow with Spiking Neural Networks* (arXiv:2106.01862). https://doi.org/10.48550/arXiv.2106.01862
15 citations (Semantic Scholar/arXiv) [2022-09-21]

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep Residual Learning for Image Recognition* (arXiv:1512.03385). https://doi.org/10.48550/arXiv.1512.03385
9999 citations (Semantic Scholar/arXiv) [2022-09-26]

Hebb, D. O. (1949). *The organization of behavior: A neuropsychological theory.* Wiley.

Heil, P. (1997). Auditory cortical onset responses revisited. I. First-spike timing. *Journal of Neurophysiology*, *77*(5), 2616–2641. https://doi.org/10.1152/jn.1997.77.5.2616
223 citations (Semantic Scholar/DOI) [2023-02-15]

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
9993 citations (Semantic Scholar/DOI) [2022-10-06]

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366. `https://doi.org/10.1016/0893-6080(89)90020-8`

Hu, Y., Tang, H., & Pan, G. (2018). Spiking Deep Residual Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 1–6. `https://doi.org/10.1109/TNNLS.2021.3119238`
55 citations (Semantic Scholar/DOI) [2022-09-12]

Huang, G., Liu, Z., van der Maaten, L., & Weinberger, K. Q. (2018). *Densely Connected Convolutional Networks* (arXiv:1608.06993). `https://doi.org/10.48550/arXiv.1608.06993`
9993 citations (Semantic Scholar/arXiv) [2022-09-23]

Huh, D., & Sejnowski, T. J. (2017). Gradient Descent for Spiking Neural Networks
184 citations (Semantic Scholar/arXiv) [2022-10-06].

Imam, N., & Cleland, T. A. (2020). Rapid online learning and robust recall in a neuromorphic olfactory circuit. *Nature Machine Intelligence*, *2*(3), 181–191. `https://doi.org/10.1038/s42256-020-0159-4`

Johansson, R. S., & Birznieks, I. (2004). First spikes in ensembles of human tactile afferents code complex spatial fingertip events. *Nature Neuroscience*, *7*(2), 170–177. `https://doi.org/10.1038/nn1177`
540 citations (Semantic Scholar/DOI) [2022-07-27]

Katz, B., & Miledi, R. (1965). The measurement of synaptic delay, and the time course of acetylcholine release at the neuromuscular junction. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, *161*(985), 483–495. `https://doi.org/10.1098/rspb.1965.0016`
545 citations (Semantic Scholar/DOI) [2022-09-29]

Kendall, J. D., & Kumar, S. (2020). The building blocks of a brain-inspired computer. *Applied Physics Reviews*, *7*(1), 011305. `https://doi.org/10.1063/1.5129306`
74 citations (Semantic Scholar/DOI) [2022-10-12]

Koch, C., & Laurent, G. (1999). Complexity and the nervous system. *Science (New York, N.Y.)*, *284*(5411), 96–98. `https://doi.org/10.1126/science.284.5411.96`
340 citations (Semantic Scholar/DOI) [2022-10-08]

Kogler, J., Sulzbachner, C., & Kubinger, W. (2009). Bio-inspired Stereo Vision System with Silicon Retina Imagers. In *Computer Vision Systems* (pp. 174–183). Springer. `https://doi.org/10.1007/978-3-642-04667-4_18`

Kreiser, R., Renner, A., Leite, V. R. C., Serhan, B., Bartolozzi, C., Glover, A., & Sandamirskaya, Y. (2020). An On-chip Spiking Neural Network for Estimation of the Head Pose of the iCub Robot. *Frontiers in Neuroscience*, *14*.

Kugele, A., Pfeil, T., Pfeiffer, M., & Chicca, E. (2020). Efficient Processing of Spatio-Temporal Data Streams With Spiking Neural Networks. *Frontiers in Neuroscience*, *14*
27 citations (Semantic Scholar/DOI) [2022-09-20].

Lagorce, X., Orchard, G., Galluppi, F., Shi, B. E., & Benosman, R. B. (2017). HOTS: A Hierarchy of Event-Based Time-Surfaces for Pattern Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(7), 1346–1359. `https://doi.org/10.1109/TPAMI.2016.2574707`

Lee, C., Kosta, A. K., & Roy, K. (2021). Fusion-FlowNet: Energy-Efficient Optical Flow Estimation using Sensor Fusion and Deep Fused Spiking-Analog Network Architectures.

Lee, C., Kosta, A. K., Zhu, A. Z., Chaney, K., Daniilidis, K., & Roy, K. (2020). Spike-FlowNet: Event-based Optical Flow Estimation with Energy-Efficient Hybrid Neural Networks. `https://doi.org/10.48550/arXiv.2003.06696`

Lee, J. H., Delbruck, T., & Pfeiffer, M. (2016). Training Deep Spiking Neural Networks Using Backpropagation. *Frontiers in Neuroscience*, *10*
542 citations (Semantic Scholar/DOI) [2022-09-13].

Li, S., Li, W., Cook, C., & Gao, Y. (2020). Deep Independently Recurrent Neural Network (IndRNN). `https://doi.org/10.48550/arXiv.1910.06251`

Liao, Q., & Poggio, T. (2020). *Bridging the Gaps Between Residual Learning, Recurrent Neural Networks and Visual Cortex* (arXiv:1604.03640). `https://doi.org/10.48550/arXiv.1604.03640`
221 citations (Semantic Scholar/arXiv) [2022-09-26]

Liu, M., & Delbruck, T. (2018). Adaptive Time-Slice Block-Matching Optical Flow Algorithm for Dynamic Vision Sensors. *Liu, Min; Delbruck, T (2018). Adaptive Time-Slice Block-Matching Optical Flow Algorithm for Dynamic Vision Sensors. In: British Machine Vision Conference (BMVC) 2018, Newcastle upon Tyne, UK, 3 September 2018 - 6 September 2018, BMVC.* `https://doi.org/10.5167/uzh-168589`

Liu, T.-Y., Mahjoubfar, A., Prusinski, D., & Stevens, L. (2022). Neuromorphic Computing for Content-based Image Retrieval. *PLOS ONE*, *17*(4), e0264364. `https://doi.org/10.1371/journal.pone.0264364`

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural networks*, *10*(9), 1659–1671.

Massa, R., Marchisio, A., Martina, M., & Shafique, M. (2021). An Efficient Spiking Neural Network for Recognizing Gestures with a DVS Camera on the Loihi Neuromorphic Processor.

Mayr, C., Hoeppner, S., & Furber, S. (2019). SpiNNaker 2: A 10 Million Core Processor System for Brain Simulation and Machine Learning. `https://doi.org/10.48550/arXiv.1911.02385`

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*(4), 115–133. `https://doi.org/10.1007/BF02478259`

McKennoch, S., Liu, D., & Bushnell, L. (2006). Fast Modifications of the SpikeProp Algorithm. *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, 3970–3977. `https://doi.org/10.1109/IJCNN.2006.246918`
119 citations (Semantic Scholar/DOI) [2022-09-29]

Mead, C. (1990). Neuromorphic electronic systems. *Proceedings of the IEEE*, *78*(10), 1629–1636. `https://doi.org/10.1109/5.58356`

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., Jackson, B. L., Imam, N., Guo, C., Nakamura, Y., Brezzo, B., Vo, I., Esser, S. K., Appuswamy, R., Taba, B., Amir, A., Flickner, M. D., Risk, W. P., Manohar, R., & Modha, D. S. (2014). Artificial brains. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science (New York, N.Y.)*, *345*(6197), 668–673. `https://doi.org/10.1126/science.1254642`

Minneci, F., Kanichay, R. T., & Silver, R. A. (2012). Estimation of the time course of neurotransmitter release at central synapses from the first latency of postsynaptic currents. *Journal of Neuroscience Methods*, *205*(1), 49–64. `https://doi.org/10.1016/j.jneumeth.2011.12.015`
16 citations (Semantic Scholar/DOI) [2022-09-29]

Mostafa, H. (2018). Supervised Learning Based on Temporal Coding in Spiking Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(7), 3227–3235. `https://doi.org/10.1109/TNNLS.2017.2726060`
214 citations (Semantic Scholar/DOI) [2022-09-13]

Neftci, E. O., Mostafa, H., & Zenke, F. (2019). Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks. *IEEE Signal Processing Magazine*, *36*(6), 51–63. `https://doi.org/10.1109/MSP.2019.2931595`
234 citations (Semantic Scholar/DOI) [2022-09-15]

Orchard, G., Jayawant, A., Cohen, G. K., & Thakor, N. (2015). Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades. *Frontiers in Neuroscience*, *9*, 437. `https://doi.org/10.3389/fnins.2015.00437`
316 citations (Semantic Scholar/DOI) [2022-10-10]

Paredes-Vallés, F., & de Croon, G. C. H. E. (2021). Back to Event Basics: Self-Supervised Learning of Image Reconstruction for Event Cameras via Photometric Constancy. `https://doi.org/10.48550/arXiv.2009.08283`

Patel, K., Hunsberger, E., Batir, S., & Eliasmith, C. (2021). A Spiking Neural Network for Image Segmentation. `https://doi.org/10.48550/arXiv.2106.08921`

Perez-Nieves, N., Leung, V. C. H., Dragotti, P. L., & Goodman, D. F. M. (2021). Neural heterogeneity promotes robust learning. *Nature Communications*, *12*(1), 1–9. `https://doi.org/10.1038/s41467-021-26022-3`
37 citations (Semantic Scholar/DOI) [2022-10-07]

Perrett, D., Rolls, E., & Caan, W. (1982). Visual neurons responsive to faces in the monkey temporal cortex. *Experimental brain research. Experimentelle Hirnforschung. Expérimentation cérébrale*, *47*, 329–42. `https://doi.org/10.1007/BF00239352`
848 citations (Semantic Scholar/DOI) [2022-07-15]

Pfeiffer, M., & Pfeil, T. (2018). Deep Learning With Spiking Neurons: Opportunities and Challenges. *Frontiers in Neuroscience*, *12*
278 citations (Semantic Scholar/DOI) [2022-07-12].

Pham, D. T., Packianather, M. S., & Charles, E. Y. A. (2008). Control chart pattern clustering using a new self-organizing spiking neural network. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, *222*(10), 1201–1211. `https://doi.org/10.1243/09544054JEM1054`
7 citations (Semantic Scholar/DOI) [2022-09-29]

Ponulak, F., & Kasiński, A. (2010). Supervised Learning in Spiking Neural Networks with ReSuMe: Sequence Learning, Classification, and Spike Shifting. *Neural Computation*, *22*(2), 467–510. `https://doi.org/10.1162/neco.2009.11-08-901`
402 citations (Semantic Scholar/DOI) [2022-09-13]

Ponulak, F., & Kasiński, A. (2011). Introduction to spiking neural networks: Information processing, learning and applications. *Acta neurobiologiae experimentalis*, *71*, 409–33
201 citations (Semantic Scholar/DOI) [2022-07-12].

Posch, C., Serrano-Gotarredona, T., Linares-Barranco, B., & Delbruck, T. (2014). Retinomorphic Event-Based Vision Sensors: Bioinspired Cameras With Spiking Output. *Proceedings of the IEEE*, *102*(10), 1470–1484. `https://doi.org/10.1109/JPROC.2014.2346153`
221 citations (Semantic Scholar/DOI) [2022-10-13]

Rebecq, H., Gehrig, D., & Scaramuzza, D. (2018). ESIM: an Open Event Camera Simulator. *Proceedings of The 2nd Conference on Robot Learning*, 969–982.

Rolls, E. T., & Tovee, M. J. (1994). Processing speed in the cerebral cortex and the neurophysiology of visual masking. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, *257*(1348), 9–15. `https://doi.org/10.1098/rspb.1994.0087`
367 citations (Semantic Scholar/DOI) [2022-07-15]

Schuman, C. D., Kulkarni, S. R., Parsa, M., Mitchell, J. P., Date, P., & Kay, B. (2022). Opportunities for neuromorphic computing algorithms and applications. *Nature Computational Science*, *2*(1), 10–19. `https://doi.org/10.1038/s43588-021-00184-y`
18 citations (Semantic Scholar/DOI) [2022-10-12]

Schuman, C. D., Potok, T. E., Patton, R. M., Birdwell, J. D., Dean, M. E., Rose, G. S., & Plank, J. S. (2017). A Survey of Neuromorphic Computing and Neural Networks in Hardware. `https://doi.org/10.48550/arXiv.1705.06963`
458 citations (Semantic Scholar/arXiv) [2022-10-12]

Sherstinsky, A. (2020). Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Physica D: Nonlinear Phenomena*, *404*, 132306. `https://doi.org/10.1016/j.physd.2019.132306`
874 citations (Semantic Scholar/DOI) [2022-09-20]

Shrestha, S. B., & Orchard, G. (2018). *SLAYER: Spike Layer Error Reassignment in Time* (arXiv:1810.08646). `https://doi.org/10.48550/arXiv.1810.08646`
313 citations (Semantic Scholar/arXiv) [2022-09-28]

Shrestha, S. B., & Song, Q. (2015). Adaptive learning rate of SpikeProp based on weight convergence analysis. *Neural Networks: The Official Journal of the International Neural Network Society*, *63*, 185–198. `https://doi.org/10.1016/j.neunet.2014.12.001`
41 citations (Semantic Scholar/DOI) [2022-09-29]

Sironi, A., Brambilla, M., Bourdis, N., Lagorce, X., & Benosman, R. (2018). HATS: Histograms of Averaged Time Surfaces for Robust Event-Based Object Classification. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 1731–1740. `https://doi.org/10.1109/CVPR.2018.00186`

Sporea, I., & Grüning, A. (2013). Supervised Learning in Multilayer Spiking Neural Networks. *Neural Computation*, *25*(2), 473–509. `https://doi.org/10.1162/NECO_a_00396`
149 citations (Semantic Scholar/DOI) [2022-09-15]

Stoffregen, T., Scheerlinck, C., Scaramuzza, D., Drummond, T., Barnes, N., Kleeman, L., & Mahony, R. (2020). Reducing the Sim-to-Real Gap for Event Cameras. `https://doi.org/10.48550/arXiv.2003.09078`

Taherkhani, A., Belatreche, A., Li, Y., Cosma, G., Maguire, L. P., & McGinnity, T. M. (2020). A review of learning in biologically plausible spiking neural networks. *Neural Networks*, *122*, 253–272.

Taherkhani, A., Belatreche, A., Li, Y., & Maguire, L. P. (2015). DL-ReSuMe: A Delay Learning-Based Remote Supervised Method for Spiking Neurons. *IEEE Transactions on Neural Networks and Learning Systems*, *26*(12), 3137–3149. `https://doi.org/10.1109/TNNLS.2015.2404938`
54 citations (Semantic Scholar/DOI) [2022-09-29]

Taherkhani, A., Belatreche, A., Li, Y., & Maguire, L. P. (2018). A Supervised Learning Algorithm for Learning Precise Timing of Multiple Spikes in Multilayer Spiking Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(11), 5394–5407. `https://doi.org/10.1109/TNNLS.2018.2797801`
46 citations (Semantic Scholar/DOI) [2022-09-29]

Tang, G., Shah, A., & Michmizos, K. P. (2019). Spiking Neural Network on Neuromorphic Hardware for Energy-Efficient Unidimensional SLAM. `https://doi.org/10.48550/arXiv.1903.02504`

Tayarani-Najaran, M.-H., & Schmuker, M. (2021). Event-Based Sensing and Signal Processing in the Visual, Auditory, and Olfactory Domain: A Review. *Frontiers in Neural Circuits*, *15*, 610446. `https://doi.org/10.3389/fncir.2021.610446`
4 citations (Semantic Scholar/DOI) [2022-10-11]

Thorpe, S., & Gautrais, J. (1998). Rank Order Coding, 113–118. `https://doi.org/10.1007/978-1-4615-4831-7_19`
224 citations (Semantic Scholar/DOI) [2022-07-27]

Thorpe, S., & Imbert, M. (1989). Biological constraints on connectionist models
46 citations (Semantic Scholar/DOI) [2022-07-12].

Urbanczik, R., & Senn, W. (2009). A Gradient Learning Rule for the Tempotron. *Neural Computation*, *21*(2), 340–352. https://doi.org/10.1162/neco.2008.09-07-605
23 citations (Semantic Scholar/DOI) [2022-09-15]

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., & Kavukcuoglu, K. (2016). WaveNet: A Generative Model for Raw Audio. https://doi.org/10.48550/arXiv.1609.03499
5029 citations (Semantic Scholar/arXiv) [2022-10-05]

Vasyutynskyy, V., & Kabitzsch, K. (2010). Event-based control: Overview and generic model. *2010 IEEE International Workshop on Factory Communication Systems Proceedings*, 271–279. https://doi.org/10.1109/WFCS.2010.5548623

Wang, C., Cleland, B. G., & Burke, W. (1985). Synaptic delay in the lateral geniculate nucleus of the cat. *Brain Research*, *343*(2), 236. https://doi.org/10.1016/0006-8993(85)90740-1
15 citations (Semantic Scholar/DOI) [2022-09-29]

Wang, X., Lin, X., & Dang, X. (2020). Supervised learning in spiking neural networks: a review of algorithms and evaluations. *Neural Networks*, *125*, 258–280.

Weidel, P., & Sheik, S. (2021). *WaveSense: Efficient Temporal Convolutions with Spiking Neural Networks for Keyword Spotting* (arXiv:2111.01456). https://doi.org/10.48550/arXiv.2111.01456
2 citations (Semantic Scholar/arXiv) [2022-09-23]

Werbos, P. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560. https://doi.org/10.1109/5.58337
4033 citations (Semantic Scholar/DOI) [2022-07-13]

Wu, Y., Deng, L., Li, G., Zhu, J., & Shi, L. (2018). Spatio-Temporal Backpropagation for Training High-Performance Spiking Neural Networks. *Frontiers in Neuroscience*, *12*.

Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., & Shi, L. (2019). Direct Training for Spiking Neural Networks: Faster, Larger, Better. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*(01), 1311–1318. https://doi.org/10.1609/aaai.v33i01.33011311

Xing, Y., Di Caterina, G., & Soraghan, J. (2020). A New Spiking Convolutional Recurrent Neural Network (SCRNN) With Applications to Event-Based Hand Gesture Recognition. *Frontiers in Neuroscience*, *14*
23 citations (Semantic Scholar/DOI) [2022-09-20].

Xu, Y., Zeng, X., Han, L., & Yang, J. (2013). A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks. *Neural Networks: The Official Journal of the International Neural Network Society*, *43*, 99–113. https://doi.org/10.1016/j.neunet.2013.02.003
127 citations (Semantic Scholar/DOI) [2022-09-29]

Yin, B., Corradi, F., & Bohté, S. M. (2020). Effective and Efficient Computation with Multiple-timescale Spiking Recurrent Neural Networks. https://doi.org/10.48550/arXiv.2005.11633
27 citations (Semantic Scholar/arXiv) [2022-10-04]

Zeldenrust, F., Wadman, W. J., & Englitz, B. (2018). Neural Coding With Bursts-Current State and Future Perspectives. *Frontiers in Computational Neuroscience*, *12*, 48. https://doi.org/10.3389/fncom.2018.00048
69 citations (Semantic Scholar/DOI) [2022-07-27]

Zenke, F., & Ganguli, S. (2018). SuperSpike: Supervised learning in multi-layer spiking neural networks. *Neural Computation*, *30*(6), 1514–1541. https://doi.org/10.1162/neco_a_01086
288 citations (Semantic Scholar/arXiv) [2022-10-10] 288 citations (Semantic Scholar/DOI) [2022-10-10]

Zenke, F., & Vogels, T. P. (2021). The Remarkable Robustness of Surrogate Gradient Learning for Instilling Complex Function in Spiking Neural Networks. *Neural Computation*, *33*(4), 899–925. `https://doi.org/10.1162/neco_a_01367`
68 citations (Semantic Scholar/DOI) [2022-10-10]

Zhu, A. Z., Yuan, L., Chaney, K., & Daniilidis, K. (2018). EV-FlowNet: Self-Supervised Optical Flow Estimation for Event-based Cameras. *Robotics: Science and Systems XIV*. `https://doi.org/10.15607/RSS.2018.XIV.062`