REVIEW **Communicated by Fernando Perez-Peña**

Advancements in Algorithms and Neuromorphic Hardware for Spiking Neural Networks

Amirhossein Javanshir
a.javanshir@deakin.edu.au
School of Engineering, Deakin University, Geelong, VIC 3216, Australia

Thanh Thi Nguyen
thanh.nguyen@deakin.edu.au
School of Information Technology, Deakin University (Burwood Campus) Burwood, VIC 3125, Australia

M. A. Parvez Mahmud
m.a.mahmud@deakin.edu.au
Abbas Z. Kouzani
abbas.kouzani@deakin.edu.au
School of Engineering, Deakin University, Geelong, VIC 3216, Australia

Artificial neural networks (ANNs) have experienced a rapid advancement for their success in various application domains, including autonomous driving and drone vision. Researchers have been improving the performance efficiency and computational requirement of ANNs inspired by the mechanisms of the biological brain. Spiking neural networks (SNNs) provide a power-efficient and brain-inspired computing paradigm for machine learning applications. However, evaluating large-scale SNNs on classical von Neumann architectures (central processing units/graphics processing units) demands a high amount of power and time. Therefore, hardware designers have developed neuromorphic platforms to execute SNNs in and approach that combines fast processing and low power consumption. Recently, field-programmable gate arrays (FPGAs) have been considered promising candidates for implementing neuromorphic solutions due to their varied advantages, such as higher flexibility, shorter design, and excellent stability. This review aims to describe recent advances in SNNs and the neuromorphic hardware platforms (digital, analog, hybrid, and FPGA based) suitable for their implementation. We present that biological background of SNN learning, such as neuron models and information encoding techniques, followed by a categorization of SNN training. In addition, we describe state-of-the-art SNN simulators. Furthermore, we review and present FPGA-based hardware implementation of SNNs. Finally, we discuss some future directions for research in this field.

Neural Computation **34, 1289–1328** (2022)
https://doi.org/10.1162/neco_a_01499

1 Introduction

In recent years, artificial neural networks (ANNs) have become the best-known approach in artificial intelligence (AI) and have achieved superb performance in various domains, such as computer vision (Abiodun et al., 2018), automotive control (Kuutti, Fallah, & Bowden, 2020), flight control (Gu, Valavanis, Rutherford, & Rizzo, 2019), and medical systems (Shahid, Rappon, & Berta, 2019). Taking inspiration from the brain, the third generation of neural networks, known as spiking neural networks (SNNs), has been developed to bridge the gap between machine learning and neuroscience (Maass, 1997). Unlike ANNs that process data values, SNNs use discrete events (or spikes) to encode and process data, which makes them more energy efficient and more computationally powerful than ANNs (Jang, Simeone, Gardner, & Gruning, 2019).

SNNs and ANNs are different in terms of their neuron models. ANNs typically use computation units, such as sigmoid, rectified linear unit (ReLU), or tanh and have no memory, whereas SNNs use a nondifferentiable neuron model and have memory, such as leaky integrate-and-fire (LIF). However, simulation of large-scale SNN models on classical von Neumann architectures (central processing units (CPUs)/graphics processing units (GPUs)) demands a large amount of time and power. Therefore, high-speed and low-power hardware implementation of SNNs is essential. Neuromorphic platforms, which are based on event-driven computation, provide an attractive solution to these problems. Thanks to neuromorphic hardware benefits, SNNs have become applicable to emerging domains, such as the Internet of Things and edge computing (Mead, 1990; Calimera, Macii, & Poncino, 2013).

Neuromorphic hardware can be divided into analog, digital, and mixed-mode (analog/digital) design. Although analog implementation offers small area and low power consumption, digital implementation is more flexible and less costly for processing large-scale SNN models (Indiveri et al. 2011; Seo & Seok, 2015). Field-programmable gate arrays (FPGAs) have been considered a suitable candidate for implementing digital neuromorphic platforms. Compared to ASICs, FPGAs offer shorter design and implementation time and excellent stability (Perez-Peña, Cifredo-Chacon, & Quiros-Olozabal, 2020). There have been several attempts to implement SNNs on single FPGA devices, which demonstrate promising speed-up compared to CPU implementation and lower power consumption compared to GPU implementation (Ju, Fang, Yan, R., Xu, & Tang, 2020; Zhang et al. 2020).

In this review, we introduce recent progress in spiking neural networks and neuromorphic hardware platforms suitable for their implementation. Section 2 introduces the SNNs' operation and typical spiking neuron and encoding schemes. Section 3 discusses the learning algorithms for SNNs, including unsupervised, supervised, and conversion approaches.

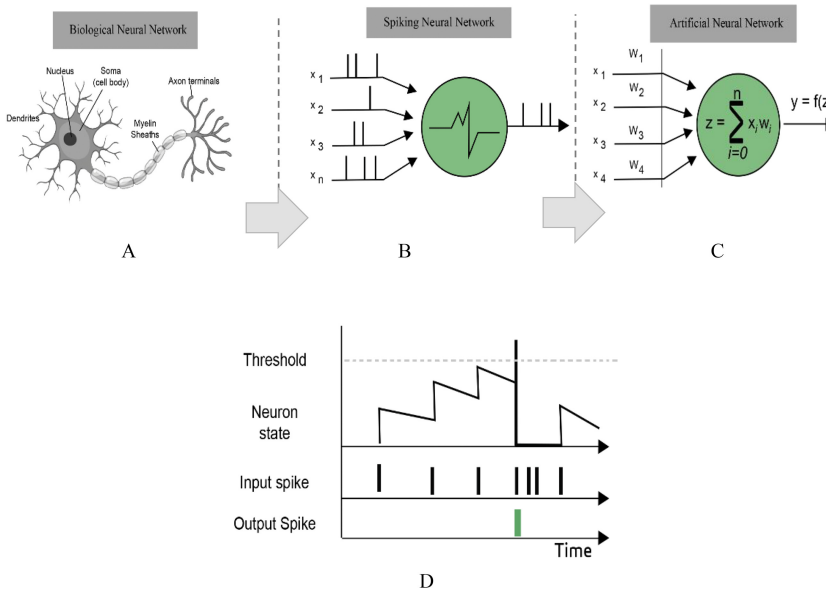

Figure 1: Schematic of a biological neural network, spiking neural network, artificial neural network, and behavior of a leaky-integrate-and-fire spiking neuron.

Performance comparison of the hardware and software implementations of SNNs is given in section 5. In section 6, major challenges and future perspectives of spiking neural networks and their neuromorphic implementations are given. Section 7 concludes.

2 Spiking Neural Networks

Spiking neural networks, considered the third generation of neural networks (Maass, 1997), communicate by sequences of spikes, discrete events that take place at points in time, as depicted in Figure 1. SNNs have been widely used in numerous applications, including the brain-machine interface (Mashford, Yepes, Kiral-Kornek, Tang, & Harrer, 2017), machine control and navigation systems (Tang & Michmizos, 2018), speech recognition (Dominguez-Morales et al. 2018), event detection (Osswald, Ieng, Benosman, & Indiveri, 2017), forecasting (Lisitsa & Zhilenkov, 2017), fast signal processing (Simeone, 2018), decision making (Wei, Bu, & Dai, 2017), and classification problems (Dora, Subramanian, Suresh, & Sundararajan, 2016). They have increasingly received attention as powerful computational platforms that can be implemented in software or hardware. Table 1 shows the differences between SNNs and ANNs in terms of neuron, topology, and

Table 1: Comparison between SNNs and ANNs.

	Spiking Neural Network	Artificial Neural Network
Neuron	Spiking neuron (e.g., integrate and fire, Hodgkin-Huxley, Izhikevich)	Artificial neuron (sigmoid, ReLU, tanh)
Information representation	Spike trains	Scalars
Computation mode	Differential equations	Activation function
Topology	LSM, Hopfield Network, RSNN, SCNN	RNN, CNN, LSTM, DBN, DNC
Features	Real-time, low power, online learning, hardware friendly, biological close, fast and massively parallel data processing	Online learning, computation intensive, moderate parallelization of computations

their features. A spiking neuron has a similar structure as an ANN neuron but different behavior. There are various spiking neuron models.

2.1 Spiking Neuron Model. A spiking neuron has a similar structure to that of an ANN neuron but shows different behavior. Over time, many different neuron models have developed in the literature, such as Hodgkin-Huxley (HH), Izhikevich, leaky integrate-and-fire (LIF), and spike response models. These models differ not only on which biological characteristics of real neurons they can reproduce but also based on their computational complexity. In this section, we review four popular and representative neuron models that are widely used in the literature in terms of their biological plausibility, the neuronal properties or behaviors that can be exhibited by each model and computational efficiency, and the number of floating-point operations needed to accomplish 1 millisecond (ms) of model simulation.

2.1.1 Hodgkin-Huxley Model. The HH model is the first biological model of a spiking neuron that describes how action potentials in the neuron are initiated and propagated (Hodgkin & Huxley, 1952). It shows the mathematical description of electric current through the membrane potential, which can be calculated as

$$I = C\frac{dv}{dt} + G_{Na}m^3h(V - V_{Na}) + G_k n^4(V - V_k) + G_L(V - V_L), \tag{2.1}$$

where I is the external current, C is the capacitance of the circuit; V_{Na}, V_k and V_L are called reverse potentials; and G_{Na}, G_k, and G_L are parameters modeling conductance of sodium, potassium, and leakage channels, respectively. Gating parameters n controls the potassium channel, while m and h control the sodium channel. These parameters are determined by equations 2.2, 2.3,

and 2.4, respectively:

$$\frac{dm}{dt} = \alpha_m(V)(1-m) - \beta_m(\upsilon)m \tag{2.2}$$

$$\frac{dn}{dt} = \alpha_n(V)(1-n) - \beta_n(\upsilon)n \tag{2.3}$$

$$\frac{dh}{dt} = \alpha_h(V)(1-h) - \beta_h(\upsilon)h. \tag{2.4}$$

The HH model, the most biologically plausible spiking neuron model, accurately capture the dynamics of many real neurons (Gerstner & Kistler 2002). However, it is too computationally expensive due to the feedback loop initiated and the differential equations for n, m, and h to be calculated continuously. Moreover, the Hodgkin-Huxley model requires about 1200 floating-point computations (FLOPS) per 1 ms of simulation (Paugam-Moisy & Bohte, 2012). Therefore, this model is less suitable for computational intelligence applications, such as large-scale neural network simulations.

2.1.2 Izhikevich Model. This biologically plausible spiking neuron model was proposed by Izhikevich (2003). This two-dimensional model can reproduce a large variety of spiking dynamics (Izhikevich, 2004). The model can be described mathematically as

$$\frac{d\upsilon(t)}{dt} = 0.04\upsilon^2 + 5\upsilon + 140 - u + I(t), \tag{2.5}$$

$$\frac{du(t)}{dt} = a(b\upsilon - u), \tag{2.6}$$

$$\upsilon(\upsilon > \upsilon_{th}) = c \text{ and } u(\upsilon > \upsilon_{th}) = u + d. \tag{2.7}$$

Izhikevich is a 2D spiking neural model that offers a good trade-off between biological plausibility and computational efficiency. It can produce various spiking dynamics and requires 13 FLOPS per 1 ms of simulation (Paugam-Moisy & Bohte, 2012). Izhikevich is a suitable model for simulation or implementation of spiking neural networks, such as hippocampus simulation and engineering problems.

2.1.3 Integrate-and-Fire Model. Integrate-and-fire (IF), one of the simple models, integrates input spikes to membrane potential; if it reaches the defined threshold, an output spike is generated, and membrane potential changes to a resting state. (Gerstner, Kistler, Naud, & Paninski, 2014; Gerstner & Kistler, 2002). This model can be determined by

$$C_m \frac{dv}{dt} = I(t), v \leftarrow v_{rest} \text{ when } v \geq v_{th}, \tag{2.8}$$

where C_m is the membrane capacitance, v_{th} is the threshold, v is the membrane potential, and v_{rest} is the resting potential. This model is the lowest one in terms of computational power consumption. In a machine learning context, spiking neurons are most often based on this simple model, which is prevalent for digital hardware implementations (Nitzsche, Pachideh, Luhn, & Becker, 2021). The leaky integrate-and-fire model, an important type of IF neuron model, adds a leak to the membrane potential. This model is defined by the following equation,

$$\tau_{leak} \frac{dv}{dt} = [v(t) - v_{rest}] + r_m I(t), v \leftarrow v_{rest} \text{ when } v \geq v_{th}, \tag{2.9}$$

where $\tau_{leak} = r_m c_m$ is the membrane time constant and r_m is the membrane resistance. The LIF model is one of the widely used spiking neuron models because of its very low computational cost (it requires only five FLOPS; Izhikevich, 2004), its accuracy in terms of replicating the spiking behavior of biological neurons, and its speed in simulating (Brette et al., 2007; Maass, 1997). Therefore, it is particularly attractive for large-scale network simulation (Aamir et al., 2018; Benjamin et al., 2014; Merolla et al., 2014). The LIF model is very popular for analog hardware implementations since the neuron's integration and decay dynamics can easily be modeled by the behavior of subthreshold transistors and capacitors (Aamir et al., 2018).

There are also more complex types of IF model such as exponential integrate-and-fire, quadratic integrate-and-fire, and adaptive exponential integrate-and-fire (Borst & Theunissen, 1999).

2.1.4 Spike Response Model. The spike response model (SRM) is a bio-inspired spiking neuron that describes more precisely the effect of input spikes on the membrane potential. Similar to the LIF model, an SRM neuron generates spikes whenever its internal membrane potential reaches the threshold (Gerstner & Kistler, 2002). However, in contrast to LIF, it includes a function dependent on reset and refractory periods. Moreover, unlike the LIF model that uses differential equations for the voltage potential, the SRM is formulated using response kernels (filters). The SRM model mathematical formulation is expressed as

$$v(t) = \eta(t - \hat{t}) + \int_{-\infty}^{+\infty} \kappa(t - \hat{t}, s)\, I(t - s) ds, \tag{2.10}$$

where $v(t)$ is the neuron's internal potential, $\hat{t}$ is the emission time of the last neuron output spike, η describes the state of the action potential, κ is

a linear response to an input spike, and $I(t)$ represents the stimulating or external current.

The 1D spike response model is simpler than other models on the level of the spike generation mechanism. It offers low computational cost, as it requires 50 computations (FLOPS) per 1 ms simulation. However, it provides poor biological plausibility compared with the Hodgkin and Huxley model (Paugam-Moisy, 2006). This model is computationally complex when used in digital systems. However, the equations that define it can be modeled by analog circuits since the postsynaptic potential function can be seen as charging and discharging RC circuits (Iakymchuk, Rosado-Muñoz, Guerrero-Martínez, Bataller-Mompeán, & Francés-Víllora, 2015).

3 Information Coding

Neural coding is still a high-impact research domain for both neuroscientists and computational artificial intelligence researchers (Borst & Theunissen, 1999) Neurons use spikes to communicate with each other in SNN architectures. Therefore, frame-based images and feature vectors need to be encoded to spike trains, a process called an encoding scheme. This scheme has a significant influence on the performance of the network. Choosing the optimal coding approaches is related to the choice of the neuron model, application target, and hardware constraints (Thiele, 2019). Rate encoding and temporal encoding are the two main encoding schemes (Kiselev, 2016).

Rate coding or frequency coding is one of the most used approaches to encode information in SNNs where information is conveyed in the firing rate. Temporal coding is another efficient coding approach for SNNs, where information is conveyed in the exact timing of spikes (Brette, 2015). Temporal coding is normally used for time series processing. Various approaches are used to generate spikes based on temporal coding, such as latency code, rank-order coding (ROC), phase coding, and population coding. In latency coding, information is encoded in the timing of response related to the encoding window (Fontaine & Peremans, 2009). Rank-order coding strategies depend on the order of spike arrivals rather than on the exact timing (Thorpe, Delorme, & Van Rullen, 2001). Compared to rate coding, ROC is able to bring more information with fewer spikes. However, it is sensitive to noise. The phase coding strategy encodes information in the phase of a pulse according to the background oscillations. This method has been used in robotic navigation and olfactory systems (Kayser, Montemurro, Logothetis, & Panzeri, 2009). In the population coding method, several neurons are used to encode one value. Sparse code is one of the examples of the population coding scheme (Wu, Amari, & Nakahara, 2002; Tkačik, Prentice, Balasubramanian, & Schneidman, 2010). An example of spike-based information coding strategies is presented in Figure 2.

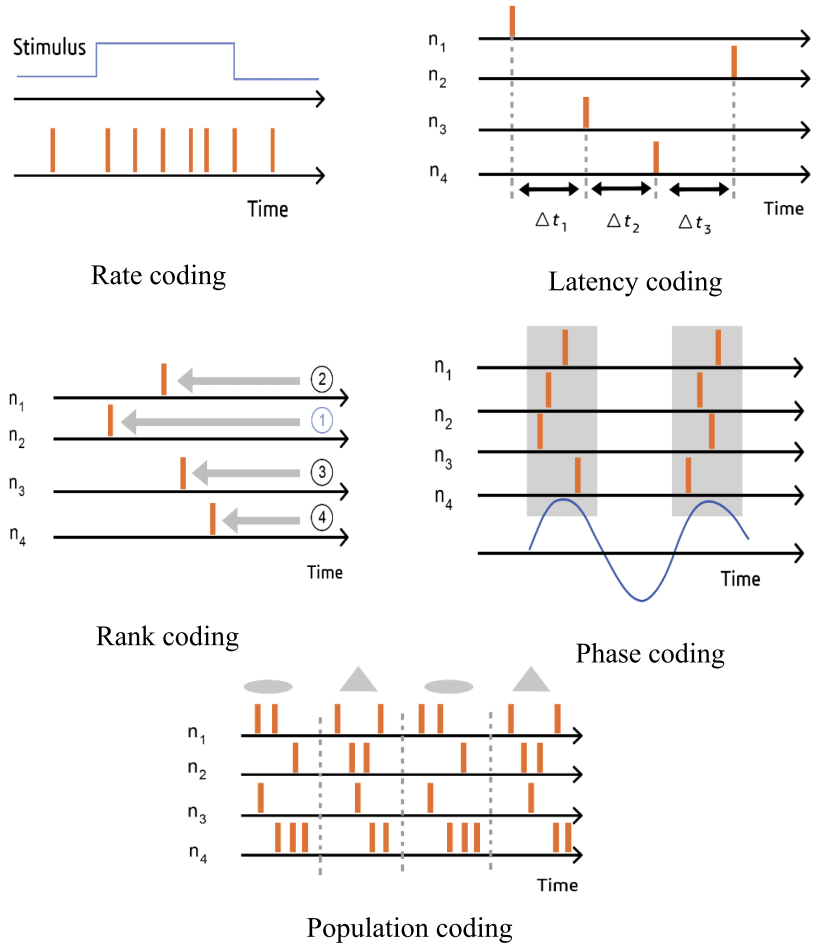

Figure 2: Spike-based information coding strategies, rate coding, latency coding, rank coding, phase coding, and population coding. $n_1, \ldots, n_4$ are labels of neurons; Δt is the relative timing of spikes; and the numbers in the circles shows the order of spike arrival.

4 Algorithms for SNNs

Learning in a spiking neural network is an arduous task. Backpropagation-based gradient descent learning is a very successful method in traditional artificial neural networks; however, training SNNs is difficult due to the nondifferentiable nature of spike events. As a result, considerable research effort has been mobilized to develop suitable learning algorithms that can

be applied to multilayer SNNs, which are thus interesting for deep learning. There are four main strategies for training SNNs: unsupervised learning, supervised learning, conversion from trained ANNs, and evolutionary algorithm. These strategies are briefly reviewed in the following subsections.

4.1 Unsupervised Learning. Unsupervised learning is the process of learning without preexisting labels. Unsupervised learning of SNNs is based on the Hebbian rule that consists of adapting the network's synaptic connections to the data received by the neurons (Caporale & Dan, 2008). The spike-timing-dependent plasticity (STDP) algorithm is an implementation of Hebb's rule. STDP is a phenomenon observed in the brain and describes how the efficacy of a synapse changes as a function of the relative timing of presynaptic and postsynaptic spikes. A presynaptic spike in this context is the spike arriving at the synapse of the neuron. The postsynaptic spike is the spike emitted by the neuron itself (Markram, Gerstner, & Sjöström, 2011). The mechanism of STDP is based on the concept that the synapses that are likely to have contributed to the firing of the neuron should be reinforced. Similarly, the synapses that did not contribute or contributed in a negative way should be weakened (Dan & Poo, 2006).

STDP is frequently used as part of the learning technique in unsupervised learning in SNNs. According to STDP, a synaptic weight is strengthened if a presynaptic neuron fires shortly before the postsynaptic neuron. Similar to that, the synaptic weight is weakened if the presynaptic spike comes briefly after the postsynaptic neuron (Xu et al. 2020). The most observed STDP rule is described by equation 4.1:

$$\Delta w = \begin{cases} +A_+ \exp\left(\frac{-\Delta t}{\tau}\right) & \text{if } \Delta t > 0 \\ -A_- \exp\left(\frac{+\Delta t}{\tau}\right) & \text{if } \Delta t \leq 0 \end{cases} \tag{4.1}$$

$$\Delta t = t_{post} - t_{pre}, \tag{4.2}$$

where w is the synaptic weights, τ is the time constant, and A_+ and A_- are constant parameters indicating the strength of potentiation and depression.

In recent years, significant research efforts have been focused on training SNNs using STDP. Qu, Zhao, Wang, and Wang (2020) developed two novel hardware-friendly methods, lateral inhibition and homeostasis, which reduce the number of inhibitory connections that lead to lowering the hardware overhead. An STDP rule was used to adapt the synapse weight between input and the learning layer and achieved 92% recognition accuracy on the MNIST data set. Xu et al. (2020) proposed a hybrid learning framework, named deep CovDenseSNN, that combines the biological plausibility of SNNs and feature extraction of CNNs. An unsupervised STDP learning rule was used to update the parameters of their proposed deep CovDenseSNN model, which is suitable for neuromorphic hardware implementation. Supervised learning and reinforcement learning are

other types of STDP methods for learning (Mozafari, Ganjtabesh, Nowzari-Dalini, Thorpe, & Masquelier, 2018; Mozafari, Kheradpisheh, Masquelier, Nowzari-Dalini, & Ganjtabesh, 2018).

Lee, Panda, Srinivasan, and Roy (2018) proposed a semisupervised strategy to train a convolutional SNN with multiple hidden layers. The training scheme had two steps: initializing the weights of the network by unsupervised learning (namely, SSTDP), and then employing the supervised gradient descent backpropagation (BP) algorithm to fine-tune the synaptic weight. Pretraining approaches led to better generalization, faster training time, and 99.28% accuracy on the MNIST database. Tavanaei, Kirby, and Maida (2018) developed a novel method to train multilayer spiking convolutional neural networks (SCNNs). The training process includes unsupervised (a novel STDP learning scheme for feature extraction) and supervised (a supervised learning scheme to train spiking CNNs (ConvNets)) components.

4.2 Supervised Learning. One of the first algorithms to train SNNs using backpropagation errors is SpikeProp, proposed by Bohte, Kok, and La Poutre (2002). This model is applied successfully to classification problems using a three-layer architecture. A later advanced version of SpikeProp called spike train SpikeProp (ST-SpikeProp) used the weight updating rule of the output layer to train the single-layer SNNs (Xu, Zeng, Han, & Yang, 2013). In order to solve the nondifferentiable problem of SNNs, Wu et al. (2018). Proposed the spatiotemporal backpropagation (STBP) algorithm, which combines the timing-dependent temporal domain and the layer-by-layer spatial domain. Supervised learning using temporal coding has shown a significant decrease in the energy consumption of SNNs. Mostafa (2017) developed a direct training approach via backpropagation error with the temporal coding scheme. His network has no convolutional layers, and the preprocessing method is not general. Zhou, Chen, Ye, and Li (2019) improved on Mostafa's work by incorporating convolutional layers into the SNN, developing a new kernel operation, and proposing a new way to preprocess the input data. Their SCNN achieved high recognition accuracy with less trainable parameters. Stromatias, Soto, Serrano-Gotarredona, and Linares-Barranco (2017) presented a supervised method for training a classifier by using the stochastic gradient descent (SGD) algorithm and then converting it to an SNN. In other work, Zheng and Mazumder (2018a) proposed backpropagation-based learning for training SNNs. Their proposed learning algorithm is suitable for implementation in neuromorphic hardware.

4.3 Conversion from Trained ANN. In the third technique, an offline-trained ANN is converted to an SNN so that the transformed network can take advantage of a well-established, fully trained ANN model. This approach is often called "spike transcoding" or "spike conversion."

Converting an ANN to SNN offers several benefits. First, a simulation of the exact spike dynamics in a large network can be computationally expensive, particularly if high firing rates and precise spike times are required. Therefore, this approach allows applying SNNs to complex benchmark tasks that require large networks, such as ImageNet or CIFAR-10, and the accuracy loss compared to their formal ANNs is small (Sengupta, Ye, Wang, Liu, & Roy, 2018; Hu, Tang, & Pan, 2018). Second, we can leverage highly efficient training techniques developed for ANNs and many state-of-the-art deep networks for classification tasks for conversion to SNNs. Moreover, the optimization process can be performed on ANNs. This permits the use of state-of-the-art optimization procedures and GPUs for training (Diehl et al., 2015). The main disadvantage is that the conversion technique fails to provide on-chip learning capability. Furthermore, some particularities of SNNs, which do not exist in the corresponding ANNs, cannot be considered during training. For this reason, the inference performance of the SNNs is typically lower than that of the original ANNs (Pfeiffer & Pfeil, 2018).

Significant research has been carried out to convert an ANN to an SNN with successful performance on the MNIST data set. Diehl et al. (2015) proposed a technique for converting an ANN into an SNN that has the minimum performance loss in the conversion process, and a recognition rate of 98.64% was achieved on the MNIST database. In another work, Rueckauer, Lungu, Hu, Pfeiffer, and Liu (2017) converted continuous-valued deep CNN to accurate spiking equivalent. This network, which includes common operations such as softmax, max-pooling, batch normalization, biases, and inception modules, demonstrates a recognition rate of 99.44% on the MNIST data set. Xu, Tang, Xing, and Li (2017) proposed a conversion method that is suitable for mapping on neuromorphic hardware. They presented a threshold rescaling method to reduce the loss and achieved a maximum accuracy of 99.17% on the MNIST data set. Xu et al. (2020) established an efficient and hardware-friendly conversion rule to convert CNNs into spiking CNNs. They proposed an "n-scaling" weight mapping method that achieves high accuracy and low-latency classification on the MNIST data set. Wang, Xu, Yan, and Tang (2020) proposed a weights-thresholds balance conversion technique that needs fewer memory resources and achieves high recognition accuracy on the MNIST data set. In contrast to the existing conversion techniques, which focus on the approximation between the artificial neurons' activation values and the spiking neurons' firing rates, they focused on the relationship between weights and thresholds of spiking neurons during the conversion process.

4.4 Evolutionary Spiking Neural Networks. Evolutionary algorithms (EAs) are population-based metaheuristics. Historically, their design was motivated by observations about natural evolution in biological populations. Such algorithms can be used to directly optimize the network topology and model hyperparameters or optimize synaptic weights and delays

(Saleh, Hameed, Najib, & Salleh, 2014; Schaffer, 2015). Currently, evolutionary algorithms such as differential evolution (DE), grammatical evolution (GE), harmony search algorithm (HSA), and particle swarm optimization (PSO) are used to learn the synaptic weights of SNNs. Vazquez (2010), López-Vázquez et al. (2019), and Yusuf et al. (2017) have shown how the synaptic weights of a spiking neuron, including integrate-and-fire, Izhikevich, and spike response model (SRM) models can be trained by using algorithms such as DE, GE, and HSA to perform classification tasks. Vazquez and Garro (2011) applied the PSO algorithm to train the synaptic weights of a spiking neuron in linear and nonlinear classification problems. They discovered that input patterns of the same class produce equal firing rates. The parallel differential evolution approach was introduced by Pavlidis, Tasoulis, Plagianakos, Nikiforidis, and Vrahatis (2005) for training supervised feedforward SNNs. Their approach is tested on exclusive-OR, which does not represent its benefits. Evolutionary algorithms can be an alternative to exhaustive search. However, they are very time-consuming, notably because the fitness function is computationally expensive (Gavrilov & Panchenko, 2016).

Table 2 shows the models for developing SNNs—their architectures and learning type along with their accuracy rates on the MNIST data set. This comparison provides an insight into different SNN architectures and learning mechanisms to choose the right tool for the right purpose in future investigations.

The new concepts and architectures are still frequently tested on MNIST. However, we argue that the MNIST data set does not include temporal information and does not provide spike events generated from sensors. Compared to a static data set, a dynamic data set contains richer temporal features and therefore is more suitable to exploit an SNN's potential ability. The event-based benchmark data sets include N-MNIST (Orchard, Jayawant, Cohen, & Thakor, 2015), CIFAR10-DVS (Hongmin Li, Liu, Ji, Li, & Shi, 2017), N-CARS (Sironi, Brambilla, Bourdis, Lagorce, & Benosman, 2018), DVS-Gesture (Amir et al., 2017), and SHD (Cramer, Stradmann, Schemmel, & Zenke, 2020). Table 3 shows the models for developing SNNs—their architectures and learning type along with their accuracy rates on the neuromorphic data sets.

5 Available Hardware and Software/Frameworks

Different methods can be used for neural network implementation. Computational cost, speed, and configurability are the main concerns for the implementations. Although CPU-based simulations offer a relatively high-speed execution, they are designed to be used for general-purpose and everyday applications. They also offer a serial implementation that limits the number of neurons that can be implemented at the same time. Hardware implementations instead can provide a platform for parallel

Table 2: Summary of Recent SNN Learning Models and Their Accuracy on Handwritten Digits Data Set (NIST).

Reference	Network-Type	Encoding Method	Structure Configuration	Neuron Type	Learning Type	Learning Rule	Training/Test Sample	CA (%)
Mirsadeghi et al. (2021)	SNN	Temporal coding	784-500-10	Linear SRM	Supervised	STiDi-BP	60,000/10,000	97.4
Fu & Dong (2021)	Spiking CNN	Rank order coding	NA	LIF	Unsupervised	Variable threshold + STD	60,000/10,000	99.27
Qu et al. (2020)	SNN	Temporal coding	784-400-400-10	Nonleaky IF	Unsupervised	STDP	60,000/10,000	92
Xu et al. (2020)	Deep CovDenseSNN	Rate coding	6C6@28×28-12C5-24C5-P	LIF	Unsupervised	Hybrid spike-based learning. STDP	60,000/10,000	91.4
Xu et al. (2020)	Spiking CNN	Rate coding	5C5-2P-64C5-2P-10FC	IF	Supervised	Conversion rule	70,000	99.09
Wang et al. (2020)	Deep SNN	Rate coding	64C3-MP2-64C3-2MP-128FC-10	LIF	Supervised	Weights-threshold balance conversion	60,000/10,000	99.43
Zhou et al. (2019)	Spiking CNN	Temporal coding	NA	Nonleaky IF	Supervised	Temporal backpropagation	60,000/10,000	98.50
Zheng and Mazumder (2018a)	SNN	Rate coding	784-300-100-10	LIF	Supervised	Online learning stochastic GD	60,000/10,000	97.8
Kulkarni and Rajendran (2018)	SNN	NA	784-8112-10	LIF	Supervised	Normalized approximate descent	50,000/10,000	98.17
Lee et al. (2018)	Deep CovDenseSNN	Rate coding	28×28-20C5-2P-50C5-2P-200FC-10FC	LIF	Semisupervised	STDP-based pretraining + backpropagation	60,000/10,000	99.28
Shrestha and Orchard (2018)	Deep SNN	Temporal coding	28×28-12c5-2a-64c5-2a-10o	LIF	Supervised	Backpropagation	60,000/10,000	99.36
Tavanaei et al. (2018)	Spiking CNN	Temporal coding	64C5-2P-1500FC-10FC	LIF	Both	STDP rep. learning and BP-STDP	60,000/10,000	98.60
Mostafa (2017)	SNN	Temporal coding	784-400-400-10	Nonleaky IF	Supervised	Temporal backpropagation	60,000/10,000	97.14
Xu et al. (2017)	Spiking ConvNet	Rate coding	28×28-32c5-2s-64c5-2s-1024f-10o	IF	Supervised	Conversion rule	20,000	99.17
Stromatias et al. (2017)	Spiking CNN	Temporal coding	NA	LIF	Supervised	Stochastic GD	60,000/10,000	98.42

Table 3: Summary of Recent SNN Learning Models and Their Accuracy on Event-Based Data Set.

Reference	Network Type	Learning Rule and Structure Configuration	Data Set	CA %
Kugele et al. (2020)	SNN	ANN-to-SNN conversion	N-MNIST CIFAR-DVS DvsGesture N-Cars	95.54 66.61 96.97 94.07
Wu et al. (2018)	Spiking MLP	Spatiotemporal backpropagation (STBP) $34 \times 34 \times 2$-800-10	N_MNIST	98.78
Wu et al. (2019)	SNN	Spatiotemporal backpropagation (STBP) 128C3(Encoding)-128C3-AP2-384C3-384C3-AP2-1024FC-512FC-Voting	N-MNIST CIFAR-DVS	99.53 60.5
Zheng et al. (2020)	ResNet17 SNN	Threshold-dependent batch normalization method based on spatiotemporal backpropagation (STBP-tdBN)	CIFAR-DVS DvsGesture	67.80 96.87
Lee et al. (2016)	SNN	Supervised backpropagation $(34 \times 34 \times 2)$-800-10	N-MNIST	98.66
Yao et al. (2021)	Spiking CNN	Temporal-wise attention SNN (TA-SNN) (1) Input-MP4-64C3-128C3-AP2-128C3-AP2-256FC-11 (2) Input-32C3-AP2-64C3-AP2-128C3-AP2-256C3-AP2-512C3-AP4-256FC-10 (3) Input-128FC-128FC-20	DvsGesture CIFAR-DVS SHD	98.61 72 91.08
Neil and Liu (2016)	Spiking CNN	ANN-to-SNN conversion	N-MNIST	95.72

implementations. Although analog implementation is relatively efficient, it suffers from an expensive and long design and implementation process. FPGA instead offers a configurable platform that offers parallel processing, which makes it a suitable candidate for SNN implementations.

5.1 Available Software. There are many different SNN simulators—for example, BindsNET, Nengo, NeMo, Brian2GeNN, Nest, and CARLsim. Existing simulators have different levels of biological models, computational speed, and support for hardware platforms. They are classified into three main groups depending on how the neural model dynamic evaluation is computed: event driven (asynchronous), where the membrane potential is modified only when a spike arrives; clock-driven (synchronous), where the neural state is updated at every tick of a clock; and hybrid strategies (asynchronous and synchronous) (Rudolph-Lilith, Dubois, & Destexhe, 2012).

Event-driven simulators are not as widely used as clock-driven simulators due to their implementation complexity. Moreover, they are difficult to parallelize due to their sequential nature. Their main advantage is their higher operation speed because they do not calculate small update steps for a neuron. Another benefit of event-driven simulators is that the timing of spikes can be represented with high precision. These simulators are more suitable for neural network layers with low and sparse activity (Naveros, Garrido, Carrillo, Ros, & Luque, 2017).

The majority of SNN simulators are clock-driven. Because of high parallelism, clock-driven simulators take full advantage of parallel computing resources in CPU and GPU platforms. Their platforms perform better for small and medium-size groups of neurons with a low to medium mathematical complexity, whereas GPU clock-driven platforms perform better for large-size groups of neurons with high mathematical complexity. The main advantage of clock-driven simulators is that they are suitable for simulating large networks when a large number of events is triggered. Many of these simulators are built on top of the existing deep learning frameworks because they are structurally similar to simulating an ANN. Their main disadvantages are that spike timings are aligned to ticks of the clock and threshold conditions are checked only at the ticks of the clock (Brette et al., 2007). Selecting the most appropriate technique requires a trade-off among three elements: (1) neural network architecture (e.g., number of neurons, neural model complexity, number of input and output synapses, mean firing rates), (2) hardware resources (number of CPU and GPU cores, RAM size), and (3) simulation requirements and targets.

Among the SNN simulators that have been reported in the literature are BindsNET (Hazan et al., 2018), Nengo (Bekolay et al., 2014), NeMo (Fidjeland, Roesch, Shanahan, & Luk, 2009), GeNN (Yavuz et al., 2016), Brain 2 (Stimberg, Brette, & Goodman, 2019), Brian2GeNN (Stimberg, Goodman, & Nowotny, 2020), NEST (Gewaltig & Diesmann, 2007), CARLsim (Beyeler, Carlson, Chou, Dutt, & Krichmar, 2015; Chou et al., 2018), NeuCube (Kasabov, 2014), PyNN (Davison, 2009), ANNarchy (Vitay, Dinkelbach, & Hamker, 2015), and NEURON (Hines & Carnevale 1997). There are some major criteria for choosing an SNN simulator. It should be open access; easy to debug and run; and support various hardware such as ASIC and FPGA to execute the simulation and support the level of biological complexity. We describe the main features of prominent existing SNNs simulators in Table 3.

BindsNET is an open-source Python package for rapid building and simulation of SNNs, which developed on top of the PyTorch deep learning library for its matrix computation. BindsNET allows researchers to test the software prototypes on CPUs or GPUs and then deploy the model to dedicated hardware (Hazan et al., 2018).

Nengo is a neural simulator based on the neural engineering framework for simulating both large-scale spiking and nonspiking neural models. It

is written in Python and supports the TensorFlow back end. This Python library allows users to define neuron types, learning rules, and optimization methods (Bekolay et al., 2014).

NeMo, a C++ class library for simulating SNNs, can simulate tens of thousands of neurons on a single workstation. It has bindings for Matlab and Python and one of the supported back ends for the PyNN simulator interface (Fidjeland et al. 2009).

GeNN is an open-source library for accelerating SNN simulations on CPUs or GPUs via code generation technology (Yavuz, Turner, & Nowotny, 2016).

Brain is a popular open-source simulator for SNNs written in Python. It is highly flexible, easily extensible, and commonly used in computational neuroscience. Version 2 of Brain allows scientists to efficiently simulate SNN models (Stimberg et al., 2019). In a newly developed software package, Brian2GeNN, the GPU-enhanced neural network simulator (GeNN) can be used to accelerate simulation in the Brain simulator (Stimberg et al., 2020).

Another popular and open-source simulator for SNNs is NEST, focusing on the dynamics, size, and structure of neural network. It is suitable for large networks of spiking neurons (Gewaltig et al. 2007). CARLsim is a user-friendly and GPU-accelerated SNN library written in C++ that supports CPU-GPU co-execution (Beyeler et al., 2015). Version 4 of CARLsim has been improved to simulate large-scale SNN models in with real-time constraints (Chou et al. 2018). Table 4 shows the features of the most SNNs simulation software.

5.2 Available Hardware. Spiking neuromorphic hardware can be subdivided into analog and digital or mixed-mode (analog/digital) design. Analog hardware uses physical processes to model certain computational functions of artificial neurons. The advantage of this approach is that operations that might be costly to implement as an explicit mathematical operation can be realized very efficiently by the natural dynamics of the system (Neil & Liu, 2016). Additionally, real-valued physical variables could have almost infinite precision. Analog hardware implementations differ on the degree to which analog elements are used. Many implementations perform only the computation in the neuron with analog elements, keeping the communication of spike signals digital (Camuñas, Linares-Barranco, & Serrano-Gotarredona, 2019).

Digital hardware represents all variables of the neurons by bits, just like a classical computer. This means that the precision of variables depends on the number of bits used to represent the variables. This precision also strongly influences the energy consumption of the basic operations and the memory requirements for variable storage. The great advantage of digital designs compared to analog hardware is that the precision of variables is controllable and guaranteed. Additionally, digital hardware can be designed with established state-of-the-art techniques for chip

Table 4: Features of the Best-Known SNN Software.

Simulator	Open Source	GPU	Simulation	Programming Language	Features
BindsNET	Yes	Yes	Clock-driven	C++ (Python package)	• BindsNET can be connected to different hardware (e.g., FPGA, ASIC), to execute the simulations. • Provide an interface with the OpenAI gym library for training SNNs on reinforcement learning environment. • Suitable for application in the domain of machine learning. • A Torchvision data set has integrated into the library for computer vision tasks.
Nengo	Yes	Yes	Clock-driven	C++ (Python package)	• Focuses on high-level behaviors of spiking neural networks. • Supports multithreaded execution on CPUs. • Has simulation back ends, such as NengoFPGA, Nengo Loihi, Nengo SpiNNaker, and Nengo OpenCL. • Nengo's libraries are designed to help with deep learning, adaptive control, and cognitive modeling.
NeMo	Yes	Yes	Clock-driven	C++	• Supports multithreaded execution on CPU • Good candidate for applications in machine learning. • Can to simulate different neuron models and hardware configurations. • Can run on CUDA-enabled GPUs.
Brian2GeNN	Yes	Yes	Clock-driven	C++ (Python package)	• Supports all major operating system. • Takes advantage of both GeNN and Brain2, which allow users to run their Brian 2 scripts on NVIDIA GPU accelerators without any further necessary programming. • Using the Brian2GeNN library enhances performance by tens to hundreds of times faster

Table 4: Continued.

Simulator	Open Source	GPU	Simulation	Programming Language	Features
NEST	Yes	No	Hybrid	C++ (Python package)	• Fast and memory efficient with minimal dependencies. • Best suited for models that focus on the dynamics, size, and structure of neural systems. • Can take advantage of multicore computer in a local network to increase the available memory or to speedup the simulation.
CARLsim	Yes	Yes	Clock-driven	C++ (Python package)	• Balances between flexibility and performance by providing optimized CUDA/C++ implementations of a large number of biologically plausible model features. • Provides an easy-to-use programming interface • Provides an integrated automatic parameter tuning interface for spiking neural networks. • Allows users to add a new feature with minimal effort. • Finds open parameters of an SNN that best fit a given response behavior.

design and manufacturing. The digital solutions could be implemented on either FPGAs or application-specific integrated circuits (ASICs) (Schuman et al., 2017). Alternatively, due to the high production costs of ASICs, other research groups have focused on implementing SNNs on FPGAs.

5.2.1 Learning with Neuromorphic Hardware. Learning mechanisms are crucial for the ability of neuromorphic systems to adapt to specific applications. Various types of learning can be performed depending on the number of hyperparameters included in the learning, and the learning time can be greatly varied. When such learning is performed in a neuromorphic chip, the learning is referred to as *on-chip training* (Lee, Lee, Kim, Lee, & Seo, 2020). In order to perform on-chip training, the neuromorphic chip should have almost all of the functions required for learning (Walter, Röhrbein, & Knoll, 2015). Off-chip training is a method of implementing learning outside a neuromorphic chip using, for example, software. After external learning is completed, the weights are postprocessed according to the neuromorphic system, or the neuromorphic system is fabricated using the postprocessed weights.

Whether to implement on-chip or off-chip training depends on the application under consideration. If the objective is to design a general accelerator for machine learning, obviously the chip should allow on-chip training (Burr et al., 2015). If the purpose is to perform a unique machine learning task on embedded low-power hardware, off-chip learning, which potentially is power consuming, can be realized only once, after which the resulting network is programmed on-chip. At that point, one could argue that in some cases, the system should need to adapt to its sensing environment while operating, which is referred to as online learning. One solution is to enable off-chip training between operation times and update or fine-tune the SNNs during inactive or loading time. However, this approach still brings some drawbacks; for example, it requires adding a memory that would store the input data acquired during operation. In addition, online learning is still being researched because machine learning currently has the major drawback of forgetting, which means that a trained network cannot learn a new task without losing accuracy on its previously learned task (Zheng & Mazumder, 2018b).

For many years, STDP has been the algorithm of choice for implementing machine learning tasks in spiking neuromorphic systems (Diehl & Cook, 2014). It is popular in the neuromorphic community for several reasons. First, the field of neuromorphic computing has traditionally been inspired by biology. This is the reason that early approaches for learning in neuromorphic hardware have been inspired by mechanisms observed in the brain. In addition, STDP is straightforward to implement in analog neuromorphic hardware. Its time dependence is often modeled by an exponential decay, which can simply be calculated by analog electronic elements. Finally, if we want to apply supervised learning, these algorithms

require either complex neurons and synaptic models or floating-point values communication of gradients between layers, and thus between neurocores, which makes their hardware implementation impractical. Moreover, if weight update is performed online (i.e., during inference), feedforward operation must be paused for learning, which adds an operational delay to the system.

5.2.2 Large-Scale Neuromorphic Hardware. Appraisal of large-scale neural networks requires dedicated hardware to be highly configurable. The well-known neuromorphic architectures TrueNorth (Merolla et al., 2014), Neurogrid (Benjamin et al., 2014), BrainScaleS (Schemmel et al., 2010), Loihi (Davies et al., 2018), and SpiNNaker (Furber, Galluppi, Temple, & Plana, 2014) pursue various characteristics to emulate networks of spiking neurons. (Note that this review addresses the well-known fully digital and mixed digital-analog neuromorphic hardware.)

The IBM TrueNorth chip is a neuromorphic platform implemented in digital electronics. This chip is designed for large-scale networks evaluation and closer to human brain structure rather than von Neumann architecture used in conventional computers. A single TrueNorth chip contains 5.4 billion transistors and 4096 neurosynaptic cores. Each core includes 12.75 KB of local static random-access memory (SRAM), 256 neurons, 265 axons, and a 265×265 synapse crossbar. This chip can simulate up to 1 million neurons and 265 million synapses. A TrueNorth chip is programmable via Corelet programming language (Merolla et al., 2014).

Neurogrid is a mixed digital-analog neuromorphic device that targets real-time simulation of biological brains. The Neurogrid board is composed of 16 complementary metal-oxide-semiconductor (CMOS) NeuroCore chips, each of which has 256×256 analog neurons fabricated in a 180 nm CMOS technology. This board is able to perform real-time biological simulations of the brain with billions of synaptic connections and 1 million neurons (Benjamin et al., 2014).

BrainScaleS is a mixed-mode analog/digital neuromorphic hardware system, based on physical emulations of neuron, synapse, and plasticity models, that targets the emulation of brain-size neural networks. The system is composed of 8-inch silicon wafers capable of simulation up to 50×106 plastic synapses and 200,000 neurons. Adaptive exponential IF neuron models and synapses in an analog network core structure have been implemented in the BrainScaleS system. The communication units in the system are digital, while the pro-cessing units are analog circuits (Schemmel et al., 2010).

A fully digital neuromorphic research chip known as Loihi has been designed by Intel labs to implement SNNs. The chip is fabricated in Intel's 14 nm process technology and contains 128 cores, along with three managing Lakemont cores. The Loihi chip can implement up to 130,000 neurons and 130 million synapses. Moreover, a learning engine embedded in each

core enables on-chip learning, with various learning rules, which makes Loihi more flexible for supervisor/nonsupervisor and reinforcing models. It can process information faster and more efficiently than conventional processors up to 1000 and 10,000, respectively, which makes it an ideal candidate for solving specific types of optimization problems (Davies et al., 2018).

SpiNNaker is a large, digital neuromorphic system designed to simulate large-scale neural computational models in real time. The SpiNNaker board consists of 48 chips, each one containing 18 ARM microprocessors and a network on chip (NoC). Each core contains an ARM968 and a direct memory Access (DMA) controller to implement almost 1000 spiking neurons in real time. One of the advantages of the SpiNNaker is using an asynchronous scheme of communication. The PyNN interface makes the SpiNNaker board programmable. PyNN is a Python library that provides various spiking neuron model and synaptic plasticity rules. This neuromorphic platform has been used in neuroscience applications, such as the simulation of the visual cortex or the cerebellum (Furber, 2016).

The main features of these neuromorphic systems are shown in Table 5. Note that for the execution of the network, only the learning approaches that are implemented on-chip to run online are reviewed in this table.

5.2.3 FPGA-Based Implementation of SNN. SNN algorithms have parallel and distributed nature. Today's computer architecture and software are not suitable for SNN execution. An alternative approach is to accelerate SNN applications through dedicated hardware. Neuromorphic hardware is designed to minimize energy and cost while keeping maximum accuracy. It presents promising speed-ups compared with software programs running on CPUs, with lower power consumption than GPU.

Several neuromorphic accelerators have been used for implementing SNNs. However, they encounter some limitations, such as maximum fan-in/fan-out of a neuron and synaptic precision, and are not suited for embedded systems due to their high cost (Ji et al., 2016). FPGAs as a programmable and low-cost device can address this issue: they exhibit high performance and reconfiguration capability and are more energy efficient than current CPUs and GPUs. Furthermore, they support parallel processing and contain enough local memory to restore weights, which make them a suitable candidate for implementing SNNs (Guo, Yantir, Fouda, Eltawil, & Salama, 2021). Rahman (2017) demonstrated that with a single CPU, the processing time is slow (around 1 minute per image). However, with the successful FPGA hardware acceleration and the use of a more complex network with a higher number of filters and convolutional layers, it will be possible to use SNNs in real-time scenarios (1 second per image). Compared to application-specific integrated circuits (ASICs), FPGAs are suitable candidates for implementing digital neuromorphic platforms. They provide rapid design and fabrication time, low cost, high flexibility, more straightforward computer interface, and excellent stability. While the improvement

Table 5: Summary of Neuromorphic Systems Implementations.

Platform	Electronics	Technology (nm)	Chip Area (mm^2)	On-Chip Learning	Neuron Model	Neuron Number (chip)	Synapse Model	Synapse Number (chip)	Online Learning	Power
BrainScaleS	Analog/digital	ASIC-CMOS 180	50	Yes (STDP)	Adaptive exponential IF	512	Spiking 4-bit digital	100 K	Yes	2 kW per module (peak)
TrueNorth	Digital	ASIC-CMOS 28	430	No	LIF	1 million	Binary, 4 modulators	256 M	No	65 mW (per chip)
SpiNNaker	Digital	ASIC-CMOS 130 nm	102	Yes (synaptic plasticity rules)	LIF, IZH, HH	16,000	Programmable	16 M	Yes	1 W (per chip)
Neurogrid	Analog/digital	ASIC-CMOS 180	168	No	Adaptive quadratic IF	65,000	Shared dendrite	100 M	Yes	2.7 W
Loihi	Digital	ASIC-CMOS 14 nm	60	Yes (with plasticity rule)	Adaptive LIF	131,000	N/A	126 M	Yes	$\approx$.45 W

potential using FPGAs is high, there are still many open research questions that limit the current mainstream appeal of FPGAs.

Implementation of neural networks on FPGAs is time-consuming compared to CPUs and GPUs. An important reason that FPGAs are still not as widely used as general-purpose hardware platforms like CPUs and GPUs in neural network computing is their relatively low programmability (Hofmann, 2019). Software frameworks such as Caffe and TensorFlow support only hardware units like CPUs and GPUs and can be executed on such operating systems. Although high-level synthesis (HLS) improves development cycle on FPGAs, efficient HLS system designs still require a deep understanding of hardware details, which can be a problem for general neural network developers (Zhang & Kouzani, 2020). There is still a need for FPGA-based frameworks that support the mainstream software neural network libraries like TensorFlow and Caffe.

Several studies have reported different approaches for implementing SNNs on FPGAs for various applications. FPGA-based implementation of SNNs has been presented for classifying musical notes (Cerezuela-Escudero et al., 2015), electrocardiogram (ECG), edge detection (Qi et al., 2014), real-time image dewarping (Molin et al., 2015), locomotion systems (Guerra-Hernandez et al., 2017), biomimetic pattern generation (Ambroise, Levi, Joucla, Yvert, & Saïghi, 2013), and event-driven vision processing (Yousefzadeh, Serrano-Gotarredona, & Linares-Barranco, 2015).

Note that we focus here on recent FPGA-based implementation of SNNs for image classification domain, currently a significant field of machine learning. Many research groups are now concentrating their efforts on developing reservoir computing for solving various classification and recognition problems. Tanaka et al. (2019) summarized recent advance in physical reservoir computing, such as analog circuits, and FPGA. Yi et al. (2016) developed a real-time, hardware-based FPGA architecture of the reservoir computing method of recurrent neural network (RNN) training. Numerous studies have been focusing on designing a suitable neuromorphic architecture for liquid state machines (LSMs) on FPGAs (Liu, Jin, & Li, 2018; Wang, Jin, & Li, 2015; Jin, Liu, & Li, 2016).

There have been several attempts to implement SNNs on FPGAs for pattern recognition. Ju et al. (2020) proposed an FPGA-based deep SNN implementation. They applied a hardware-friendly, spiking, max-pooling operation and two parallel methods, shift register and coarse-grained parallel, to improve the data reuse rate. The FPGA implementation obtained 22 times lower power consumption than a GPU implementation and 41 times speed-up compared to a CPU implementation. Abderrahmane and Miramond (2019) explored a spike-based neural network for embedded artificial intelligence applications. They implemented two architectures, time-multiplexed and fully parallel, on an FPGA platform. However, the FPGA on-chip memory is not sufficient for deeper networks with these two architectures. Efficient memory access is essential to store the parameters

and evaluation of a SNN. On-chip memory has a limitation, and off-chip memory consumes more energy than on-chip memory. Thus, designing a suitable architecture can reduce memory access. Nallathambi and Chandrachoodan (2020) proposed a novel probabilistic spike propagation method that reduces the number of off-chip memory accesses required to evaluate a SNN, thus saving time and energy.

To take advantage of both event-based and frame-based processing, Yousefzadeh, Orchard, Stromatias, Serrano-Gotarredona, and Linares-Barranco (2018) proposed a hybrid neural network that combines SNN and ANN features. Their implementation on an FPGA consumes 7 uJ per frame and obtains 97% accuracy on the MNIST database. In similar work, Losh and Llamocca (2019) designed spiking hybrid network (SHiNe), FPGA-based hardware that achieved reasonable accuracy (90%) for the MNIST data set. The SHiNe design has significantly lower FPGA resource utilization (about 35% less) due to two factors: the neural network (the SHiNe network is significantly simpler than the standard neural network, requiring only 1 bit per signal) and neuron implementation (each SHiNe neuron includes only a counter and a set of comparators). They have also implemented an approach named thrifting, which limits the number of allowed connections from neurons in one layer to a neuron in the next layer. Their FPGA designs on the Zynq XC7Z010 PSoC board consume far less power than the GPU or CPU implementations. Zhang et al. (2020) developed an FPGA-based SNN implementation that provides 908,578 times speed-up compared with software implementation. They reduced the consumption of hardware resources by using arithmetic shift instead of multiplication operations, which can speed up the training efficiency.

Han, Li, Zheng, and Zhang (2020) proposed an FPGA-based SNN hardware implementation that supports up to 16.384 neurons and 16.8 million synapses with 0.477 W power consumption. They used a hybrid updating algorithm that included a time-stepped and event-driven algorithm. In addition to on-chip block random access memory (RAM), they used an external DDR memory to optimize the latency of memory access.

Kuang et al. (2019) introduced a real-time FPGA-based implementation of SNNs that significantly reduces the cost of hardware resources with multiplier-less approximation. Their proposed systems are suitable for bio-inspired neuromorphic platform and online applications. An FPGA-based parallel neuromorphic processor for SNNs presented in Wang, Li, Shao, Dey, and Li (2017) successfully tackled several critical problems related to memory organization and parallel processing. A 59.4 times training speed-up was achieved by the 32-way parallel design and reduced up to 20% energy consumption by using the approximate multipliers in their processor design. An FPGA-based SNN hardware implementation with biologically realistic neuron and synapse proposed by Fang, Shrestha, Zhao, Li, and Qiu (2019) applied a population encoding scheme to convert a continuous value into spike events. The FPGA implementation achieves 196

times lower power consumption and 10.1 times speed-up compared to a GPU implementation. Their experiments demonstrate that temporal SNN is 8.43 times speed-up compared with rate SNN on FPGA platform.

Table 6 presents the performance of recent FPGA-based implementations of SNNs in terms of network configuration, system performance, and target device.

6 Challenges and Future Research Directions

Spiking neural networks are capable of modeling information processing in the brain, such as pattern recognition. They offer promising event-driven processing, fast inferences, and low power consumption. Spiking CNNs offer a high potential for classification tasks in low-power neuromorphic hardware, as they combine both spike-based computing of SNNs and high CNN accuracy s (Diehl et al., 2015). Additionally, deep SNN offers a promising computational paradigm for improving energy efficiency and reducing classification latency. However, training spiking CNNs and deep SNNs remains challenging because of nondifferentiable spiking dynamics. To tackle this problem, we provided an overview of the state-of-the-art learning rules for SNNs in the section 3. One solution is direct supervised learning, which takes advantage of reducing power consumption and a straightforward technique. This strategy is based on the backpropagation-like technique (Lee et al., 2016) and conventional gradient descent. However, direct training-based strategies still provide less efficiency and stability in coping with a complex database. An alternative technique to direct supervised learning is converting a trained CNN to a SNN by transferring the CNN operations directly into a SNN equivalent. Various approaches have been employed to convert CNNs to SNNs, such as threshold rescaling (Xu et al., 2017), n-scaling weight mapping (Yang et al., 2020), and weights-thresholds balance (Wang, Xu, Yan, & Tang, 2020).

The conversion rule has solved the learning issue for deep SNNs. However, it is not apparent that the conversion method can scale to deeper architectures and address the complex task. Furthermore, there is a possibility of accuracy loss during the conversion of CNNs to SNNs. Other hardware-friendly approaches are local learning rules, such as STDP (Kheradpisheh, Ganjtabesh, Thorpe, & Masquelier, 2018).

This method can be a suitable option design and a biologically plausible learning algorithm for hardware implementation. Additionally, STDP is a good choice for online learning, which allows a fast real-time learning process and reduces the computational complexity.

Spiking neural networks are poorly served by classical von Neumann computing architectures due to the dynamic nature of neurons; in addition, the classic computing architecture requires an extreme amount of time and power. Thus, neuromorphic platforms are ideally suited for executing SNNs. These platforms offer better parallel implementation than that

Table 6: Summary of FGPA-Based Implementation of SNNs.

Reference	Network Type/Neuron and Encoding Model/Topology	Recognition CA (%)	FPGA Platform	Software Tool/ Language	System Performance
Corradi et al. (2021)	SNN based on conversion method	Optical radar 99.7	Trenz TE0820-03-4DE21FA	Vivado	Achieves an energy efficiency (nJ/SO) 0,151
Panchapakesan et al. (2021)	SNN based on conversion method (VGG-13)	CIFAR-10 90.79 SVHN 96	FPGA Xilinx ZCU102	Vivado HLS	Achieves 13,086 frames per second under 200 MHz
Aung et al. (2021)	Spiking CNN (AlexNet)	CIFAR-10 81.8 SVHN 93.1	Xilinx Virtex UltraScale+ FPGA VCU118	N/A	Achieves 28.3 kFPS running at 425 MHz
Nallathambi and Chandrachoodan (2020)	Spiking CNN based on conversion method	CIFAR10 76.87	Intel Cyclone V	N/A	Reduces the number of off-chip memory accesses by close to 90%
Hong et al. (2020)	Time-delay neural net (TDNN)	CIFAR-10 83.43	I Kintex7 325T	Verilog	Consumes 4.92 W at 160 MHz clock frequency
Fang et al. (2020)	SNN based on conversion rule/LIF neuron/population coding/ $784 \times 600 \times 10$	MNIST 97.7	Cyclone V	N/A	Obtains $10\times$ speedup and $196\times$ improvement in energy efficiency compared with GPU
Han (2020)	A feedforward SNN based on a hybrid of the time-stepped and event-driven updating algorithms/LIF neuron/Poisson encoding 784-1200-1200-10	MNIST 97.06	Xilinx ZC706	N/A	Achieves 161 FPS under 200 MHz clock frequency, and very low power consumption of 0.477 W

Table 6: Continued.

Reference	Network Type/Neuron and Encoding Model/Topology	Recognition CA (%)	FPGA Platform	Software Tool/ Language	System Performance
Ju et al. (2020)	Deep SNN based on conversion rule/IF neuron/fixed uniform encoding/ 28×28-64c5-2s-64c5-2s-128f-10o	MNIST 98.94	Xilinx Zynq ZCU102	N/A	Achieves 164 FPS under 150 MHz clock frequency and obtains 41× speed-up and 4.6 W power consumption.
Liu et al. (2019)	Liquid state machine (LSM) based on spike-timing-dependent-plasticity (STDP)	TI46 speech corpus 95	Xilinx Zync ZC-706	N/A	Consumes 237 mW at 100 MHz clock frequency
Losh and Llamocca (2019)	Spiking hybrid network (SHiNe) based on backpropagation learning. Integrate, rectification, and fire neuron/fixed-frequency duty-cycle encoding 196-64-10	MNIST 97.70	Xilinx Zynq XC7Z010-1CLG400C	Vivado 2016.3	Results in 65.536 μs per frame under 125 MHz clock rate and total 161 mW power consumption.
Guo (2019)	DNN to SNN conversion rule IF neuron/Poisson encoding 28×28-12c5-2s-64c5-2s-10 (CNN) 784-1200-1200-10 (FCN)	MNIST 98.98 98.84	Xilinx V7 690T	Vivado 2016.4	Consumes 0.745 W at 100 MHz clock frequency, using 32-bit fixed-point precision
Kuang et al. (2019)	Three-layer SNN based on STDP learning rules/LIF neuron and conductance-based synapse.	MNIST 93	Stratix III	Verilog HDL	N/A
Abderrahmane and Miramond (2019)	SNN based on backpropagation learning IF neuron/rate coding I784-300-300-10	MNIST 97.70	Intel Cyclone V	Quartus Prime Lite 18.10 VHDL	Achieves 256 FPS under 50 MHz clock frequency with time-multiplexed architecture and 70 K FPS with fully parallel architecture

Table 6: Continued.

Reference	Network Type/Neuron and Encoding Model/Topology	Recognition CA (%)	FPGA Platform	Software Tool/ Language	System Performance
Zhang et al. (2019)	SNN based on the backpropagation algorithm LIF neuron 256-256-10	MNIST 96.26	Terasic DE2-115	Quartus II Verilog	Obtains 10.7× speedup and 293 MW power at 100 MHz
Liu et al. (2018)	liquid state machine (LSM) based on spike-timing-dependent-plasticity (STDP)	TI46 speech corpus 93.1 CityScape 97.9	Xilinx Virtex-6	N/A	It is up to 29% more energy efficient for training and 30% more energy efficient for classifying than the baseline.
Yousefzadeh (2017)	Two-layer hybrid neural network LIF neuron/Poisson encoding	E-MNIST 97.09%	Xilinx SPARTAN-6	HDL	Achieves 58 K FPS under 220 MHz clock frequency and consumes 363 MW, which is equal to less than 7 uJ for each frame
Mostafa et al. (2017)	A feedforward SNN trained using backpropagation. LIF neuron/temporal encoding 748-600-10	MNIST 96.98	Xilinx Spartan6-LX150	N/A	N/A
Chung et al. (2015)	Time-delay neural network	MNIST 97.64	Xilinx Artix 7	Vivado	Processes an input image in 156.8 μs under 160 MHz
Neil and Liu (2014)	Spiking deep belief network LIF neuron 784-500-500-10	MNIST 92	Xilinx Spartan6-LX150	RTL	Achieves 152 ms processing time/image and 1.5 W power consumption at 75 MHz clock frequency

provided by CPUs and less power consumption than GPUs. FPGA offers a programmable and very flexible platform for SNN implementation. Compare to ASICs, FPGAs provide better stability, rapid design time, faster fabrication time, and higher flexibility. The FPGA implementation of SNNs achieves significantly lower power consumption than GPU implementation and better speed-up compared to the CPU implementation (Abderrahmane & Miramond, 2019; Nallathambi & Chandrachoodan, 2020).

In the case of the hardware implementation of deep SNNs, the number of neurons, connections, and weights can be very large, leading to an increase in the size of memory. FPGAs' on-chip memory is not sufficient to store all parameters of the network. Therefore, an external memory like SRAM is required next to the on-chip memory to store the parameters and data flow into the architecture. Thus, choosing a suitable information coding method and designing an effective architecture can reduce memory fetches. Different architectures have been used for FPGA-based implementation of SNNs such as fully parallel and time-multiplexed. Designing an FPGA architecture depends on the application target.

Focusing on the advancement of SNNs and their neuromorphic implementations, the following research aspects need to be considered, and more work is required to resolve the remaining challenges and limitations:

- One of the key challenges in developing SNNs is to deploy suitable training and learning algorithms, which profoundly affect application accuracy and execution cost.
- Another unsolved challenge is how information is encoded with spikes. Although neural coding has a remarkable effect on the fulfillment of SNNs, the questions remain as to what the best encoding approach is and how to develop a learning algorithm to be well matched by the encoding scheme. Designing a learning algorithm that is capable of training hidden neurons in an interconnected SNN has become a major challenge.

Neuromorphic computing is at an early stage, and much progress is needed in both algorithm and hardware that are capable of exhibiting human-like intelligence.

7 Conclusion

Spiking neural networks have been considered the third generation of neural network, offering a high-speed real-time implementation of a complex problem in a bio-inspired, power-efficient manner. This review offers an overview of recent strategies to train SNNs and highlights two popular deep learning methods: spiking CNNs and deep, fully connected SNNs, in terms of their learning rule, network architecture, and experiment and recognition accuracy. This review also discussed current SNN simulators, comparing three main approaches: clock-driven, event driven, and hybrid;

it presented a survey of the work done on hardware implementation of SNNs; and demonstrated that FPGAs are a promising candidate for acceleration of SNNs and can achieve better speed-up than CPUs and less energy consumption than GPUs.

References

Aamir, S. A., Stradmann, Y., Müller, P., Pehle, C., Hartel, A., Grübl, A., . . . Meier, K. (2018). An accelerated LIF neuronal network array for a large-scale mixed-signal neuromorphic architecture. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *65*(12), 4299–4312. 10.1109/TCSI.2018.2840718

Abderrahmane, N., & Miramond, B. (2019). Information coding and hardware architecture of spiking neural networks. In *Proceedings of the 22nd Euromicro Conference on Digital System Design* (pp. 291–298). Piscataway, NJ: IEEE.

Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A., & Arshad, H. (2018). State-of-the-art in artificial neural network applications: A survey. *Heliyon*, *4*(11), e00938. 10.1016/j.heliyon.2018.e00938, PubMed: 30519653

Ambroise, M., Levi, T., Joucla, S., Yvert, B., & Saïghi, S. (2013). Real-time biomimetic central pattern generators in an FPGA for hybrid experiments. *Frontiers in Neuroscience*, *7*, 215. 10.3389/fnins.2013.00215, PubMed: 24319408

Amir, A., Taba, B., Berg, D., Melano, T., McKinstry, J., & Di Nolfo, C. (2017). A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 7243–7252). Piscataway, NJ: IEEE.

Aung, M. T. L., Qu, C., Yang, L., Luo, T., Goh, R. S. M., & Wong, W. F. (2021). DeepFire: Acceleration of convolutional spiking neural network on modern field programmable gate arrays. In *Proceedings of the 31st International Conference on Field-Programmable Logic and Applications* (pp. 28–32). Piscataway, NJ: IEEE.

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T. C., Rasmussen, D., . . . Eliasmith, C. (2014). Nengo: A Python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, *7*, 48. 10.3389/fninf.2013.00048, PubMed: 24431999

Benjamin, B. V., Gao, P., McQuinn, E., Choudhary, S., Chandrasekaran, A. R., Bussat, J. M., . . . Boahen, K. (2014). Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. In *Proceedings of the IEEE*, *102*(5), 699–716. 10.1109/JPROC.2014.2313565

Beyeler, M., Carlson, K. D., Chou, T. S., Dutt, N., & Krichmar, J. L. (2015). A user-friendly and highly optimized library for the creation of neurobiologically detailed spiking neural networks. In *Proceedings of the International Joint Conference on Neural Networks*. Piscataway, NJ: IEEE.

Bohte, S. M., Kok, J. N., & La Poutre, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, *48*(14), 17–37. 10.1016/S0925-2312(01)00658-0

Borst, A., & Theunissen, F. E. (1999). Information theory and neural coding. *Nature Neuroscience*, *2*(11), 947–957. 10.1038/14731, PubMed: 10526332

Brette, R. (2015). Philosophy of the spike: Rate-based vs. spike-based theories of the brain. *Frontiers in Systems Neuroscience*, *9*, 151. 10.3389/fnsys.2015.00151, PubMed: 26617496

Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J. M., . . . Zirpe, M. (2007). Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience*, *23*(3), 349–398. 10.1007/s10827-007-0038-6, PubMed: 17629781

Burr, G. W., Narayanan, P., Shelby, R. M., Sidler, S., Boybat, I., di Nolfo, C., & Leblebici, Y. (2015). Large-scale neural networks implemented with non-volatile memory as the synaptic weight element: Comparative performance analysis (accuracy, speed, and power). In *Proceedings of the 2015 IEEE International Electron Devices Meeting*. Piscataway, NJ: IEEE.

Calimera, A., Macii, E., & Poncino, M. (2013). The human brain project and neuromorphic computing. *Functional Neurology*, *28*(3), 191. 24139655

Camuñas-Mesa, L. A., Linares-Barranco, B., & Serrano-Gotarredona, T. (2019). Neuromorphic spiking neural networks and their memristor-CMOS hardware implementations. *Materials*, *12*(17), 2745.

Caporale, N., & Dan, Y. (2008). Spike-timing-dependent plasticity: A Hebbian learning rule. *Annu. Rev. Neurosci.*, *31*, 25–46. 10.1146/annurev.neuro.31.060407.125639, PubMed: 18275283

Cerezuela-Escudero, E., Jimenez-Fernandez, A., Paz-Vicente, R., Dominguez-Morales, M., Linares-Barranco, A., & Jimenez-Moreno, G. (2015). Musical notes classification with neuromorphic auditory system using FPGA and a convolutional spiking network. In *Proceedings of the International Joint Conference on Neural Networks* (pp. 1–7). Piscataway, NJ: IEEE.

Chou, T. S., Kashyap, H. J., Xing, J., Listopad, S., Rounds, E. L., Beyeler, M., . . . Krichmar, J. L. (2018). CARLsim 4: An open source library for large scale, biologically detailed spiking neural network simulation using heterogeneous clusters. In *Proceedings of the International Joint Conference on Neural Networks* (pp. 1–8). Piscataway, NJ: IEEE.

Chung, J., Shin, T., & Kang, Y. (2015). *Insight: A neuromorphic computing system for evaluation of large neural networks*. arXiv:1508.01008.

Corradi, F., Adriaans, G., & Stuijk, S. (2021). Gyro: A digital spiking neural network architecture for multi-sensory data analytics. In *Proceedings of the 2021 Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools* (pp. 9–15). New York: ACM.

Cramer, B., Stradmann, Y., Schemmel, J., & Zenke, F. (2020). The Heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *PP*(99), 1–14.

Dan, Y., & Poo, M. M. (2006). Spike timing-dependent plasticity: From synapse to perception. *Physiol. Rev*, *86*, 1033–1048. 10.1152/physrev.00030.2005, PubMed: 16816145

Davies, M., Srinivasa, N., Lin, T. H., Chinya, G., Cao, Y., Choday, S. H., . . . Liao, Y. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, *38*(1), 82–99. 10.1109/MM.2018.112130359

Davison, A. P., Brüderle, D., Eppler, J. M., Kremkow, J., Muller, E., Pecevski, D., . . . Yger, P. (2009). PyNN: A common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, *2*, 11. 19194529

Diehl, P. U., & Cook, M. (2014). Efficient implementation of STDP rules on SpiNNaker neuromorphic hardware. In *Proceedings of the 2014 International Joint Conference on Neural Networks* (pp. 4288–4295). Piscataway, NJ: IEEE.

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S. C., & Pfeiffer, M. (2015). Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *Proceedings of the 2015 International Joint Conference on Neural Networks* (pp. 1–8). Piscataway, NJ: IEEE.

Dominguez-Morales, J. P., Liu, Q., James, R., Gutierrez-Galan, D., Jimenez-Fernandez, A., Davidson, S., & Furber, S. (2018). Deep spiking neural network model for time-variant signals classification: A real-time speech recognition approach. In *Proceedings of the 2018 International Joint Conference on Neural Networks* (pp. 1–8). Piscataway, NJ: IEEE.

Dora, S., Subramanian, K., Suresh, S., & Sundararajan, N. (2016). Development of a self-regulating evolving spiking neural network for classification problem. *Neurocomputing*, *171*, 1216–1229. 10.1016/j.neucom.2015.07.086

Fang, H., Shrestha, A., Zhao, Z., Li, Y., & Qiu, Q. (2019). An event-driven neuromorphic system with biologically plausible temporal dynamics. In *Proceedings of the 2019 IEEE/ACM International Conference on Computer-Aided Design* (pp. 1–8). Piscataway, NJ: IEEE.

Fidjeland, A. K., Roesch, E. B., Shanahan, M. P., & Luk, W. (2009). NeMo: A platform for neural modelling of spiking neurons using GPUs. In *Proceedings of the 20th IEEE International Conference on Application-Specific Systems, Architectures and Processors* (pp. 137–144). Piscataway, NJ: IEEE.

Fontaine, B., & Peremans, H. (2009). Bat echolocation processing using first-spike latency coding. *Neural Networks*, *22*(10), 1372–1382. 10.1016/j.neunet.2009.05.002, PubMed: 19481904

Fu, Q., & Dong, H. (2021). An ensemble unsupervised spiking neural network for objective recognition. *Neurocomputing*, *419*, 47–58. 10.1016/j.neucom.2020.07.109

Furber, S. (2016). Large-scale neuromorphic computing systems. *Journal of Neural Engineering*, *13*(5), 051001. 10.1088/1741-2560/13/5/051001, PubMed: 27529195

Furber, S. B., Galluppi, F., Temple, S., & Plana, L. A. (2014). The SpiNNaker project. In *Proceedings of the IEEE*, *102*(5), 652–665. 10.1109/JPROC.2014.2304638

Gavrilov, A. V., & Panchenko, K. O. (2016). Methods of learning for spiking neural networks. A survey. In *Proceedings of the 13th International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering* (vol. 2, pp. 455–460). Piscataway, NJ: IEEE.

Gerstner, W., & Kistler, W. M. (2002). *Spiking neuron models: Single neurons, populations, plasticity.* Cambridge: Cambridge University Press.

Gerstner, W., Kistler, W. M., Naud, R., & Paninski, L. (2014). *Neuronal dynamics: From single neurons to networks and models of cognition.* Cambridge: Cambridge University Press.

Gewaltig, M. O., & Diesmann, M. (2007). Nest (neural simulation tool). *Scholarpedia*, *2*(4), 1430. 10.4249/scholarpedia.1430

Gu, W., Valavanis, K. P., Rutherford, M. J., & Rizzo, A. (2019). A survey of artificial neural networks with model-based control techniques for flight control of unmanned aerial vehicles. In *Proceedings of the 2019 International Conference on Unmanned Aircraft Systems* (pp. 362–371). Piscataway, NJ: IEEE.

Guerra-Hernandez, E. I., Espinal, A., Batres-Mendoza, P., Garcia-Capulin, C. H., Romero-Troncoso, R. D. J., & Rostro-Gonzalez, H. (2017). A FPGA-based neuromorphic locomotion system for multi-legged robots. *IEEE Access*, *5*, 8301–8312. 10.1109/ACCESS.2017.2696985

Guo, S., Wang, L., Wang, S., Deng, Y., Yang, Z., Li, S., . . . Dou, Q. (2019). A systolic SNN inference accelerator and its co-optimized software framework. In *Proceedings of the 2019 on Great Lakes Symposium on VLSI* (pp. 63–68). New York: ACM.

Guo, W., Yantir, H. E., Fouda, M. E., Eltawil, A. M., & Salama, K. N. (2021). Toward the optimal design and FPGA implementation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*.

Han, J., Li, Z., Zheng, W., & Zhang, Y. (2020). Hardware implementation of spiking neural networks on FPGA. *Tsinghua Science and Technology*, *25*(4), 479–486. 10.26599/TST.2019.9010019

Hazan, H., Saunders, D. J., Khan, H., Patel, D., Sanghavi, D. T., Siegelmann, H. T., & Kozma, R. (2018). BindsNET: A machine learning-oriented spiking neural networks library in Python. *Frontiers in Neuroinformatics*, *12*, 89. 10.3389/fninf.2018.00089, PubMed: 30631269

Hines, M. L., & Carnevale, N. T. (1997). The NEURON simulation environment. *Neural Computation*, *9*(6), 1179–1209. 10.1162/neco.1997.9.6.1179, PubMed: 9248061

Hodgkin, A. L., & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *Journal of Physiology*, *117*(4), 500–544. 10.1113/jphysiol.1952.sp004764

Hofmann, J, (2019). *An improved framework for and case studies in FPGA-based application acceleration computer vision, in-network processing and spiking neural networks*. PhD diss., Technische Universität Darmstadt.

Hong, T., Kang, Y., & Chung, J. (2020). InSight: An FPGA-based neuromorphic computing system for deep neural networks. *Journal of Low Power Electronics and Applications*, *10*(4), 36. 10.3390/jlpea10040036

Hongmin Li, Liu, H., Ji, X., Li, G., & Luping Shi, L. (2017). CIFAR10-DVS: An event-stream dataset for object classification. *Frontiers in Neuroscience*, *11*, 309.

Hu, Y., Tang, H. & Pan, G. (2018). *Spiking deep residual network*. arXiv:1805.01352.

Iakymchuk, T., Rosado-Muñoz, A., Guerrero-Martínez, J. F., Bataller-Mompeán, M., & Francés-Víllora, J. V. (2015). Simplified spiking neural network architecture and STDP learning algorithm applied to image classification. *EURASIP Journal on Image and Video Processing*, *2015*(1), 1–11. 10.1186/s13640-015-0059-4

Indiveri, G., Linares-Barranco, B., Hamilton, T. J., Van Schaik, A., Etienne-Cummings, R., Delbruck, T., . . . & Boahen, K. (2011). Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience*, *5*, 73.

Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, *14*(6), 1569–1572. 10.1109/TNN.2003.820440, PubMed: 18244602

Izhikevich, E. M. (2004). Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, *15*(5), 1063–1070. 10.1109/TNN.2004.832719, PubMed: 15484883

Jang, H., Simeone, O., Gardner, B., & Gruning, A. (2019). An introduction to probabilistic spiking neural networks: Probabilistic models, learning rules, and applications. *IEEE Signal Processing Magazine*, *36*(6), 64–77. 10.1109/MSP.2019.2935234

Ji, Y., Zhang, Y., Li, S., Chi, P., Jiang, C., Qu, P., . . . Chen, W. (2016). NEUTRAMS: Neural network transformation and co-design under neuromorphic hardware constraints. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture* (pp. 1–13). Piscataway, NJ: IEEE.

Jin, Y., Liu, Y., & Li, P. (2016). SSO-LSM: A sparse and self-organizing architecture for liquid state machine based neural processors. In *Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures* (pp. 55–60). Piscataway, NJ: IEEE.

Ju, X., Fang, B., Yan, R., Xu, X., & Tang, H. (2020). An FPGA implementation of deep spiking neural networks for low-power and fast classification. *Neural Computation*, *32*(1), 182–204. 10.1162/neco_a_01245, PubMed: 31703174

Kasabov, N. K. (2014). NeuCube: A spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data. *Neural Networks*, *52*, 62–76. 10.1016/j.neunet.2014.01.006, PubMed: 24508754

Kayser, C., Montemurro, M. A., Logothetis, N. K., & Panzeri, S. (2009). Spike-phase coding boosts and stabilizes information carried by spatial and temporal spike patterns. *Neuron*, *61*(4), 597–608. 10.1016/j.neuron.2009.01.008, PubMed: 19249279

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., & Masquelier, T. (2018). STDP-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, *99*, 56–67. 10.1016/j.neunet.2017.12.005, PubMed: 29328958

Kiselev, M. (2016). Rate coding vs. temporal coding: Is optimum between? In *Proceedings of the International Joint Conference on Neural Networks* (pp. 1355–1359). Piscataway, NJ: IEEE.

Kuang, Z., Wang, J., Yang, S., Yi, G., Deng, B., & Wei, X. (2019). Digital implementation of the spiking neural network and its digit recognition. In *Proceedings of the Chinese Control and Decision Conference* (pp. 3621–3625). Piscataway, NJ: IEEE.

Kugele, A., Pfeil, T., Pfeiffer, M., & Chicca, E. (2020). Efficient processing of spatio-temporal data streams with spiking neural networks. *Frontiers in Neuroscience*, *14*, 439. 10.3389/fnins.2020.00439, PubMed: 32431592

Kulkarni, S. R., & Rajendran, B. (2018). Spiking neural networks for handwritten digit recognition: Supervised learning and network optimization. *Neural Networks*, *103*, 118–127. 10.1016/j.neunet.2018.03.019, PubMed: 29674234

Kuutti, S., Fallah, S., & Bowden, R. (2020). Training adversarial agents to exploit weaknesses in deep control policies. In *Proceedings of the IEEE International Conference on Robotics and Automation* (pp. 108–114). Piscataway, NJ: IEEE.

Lee, C., Panda, P., Srinivasan, G., & Roy, K. (2018). Training deep spiking convolutional neural networks with STDP-based unsupervised pre-training followed by supervised fine-tuning. *Frontiers in Neuroscience*, *12*, 435.

Lee, J. E., Lee, C., Kim, D.W., Lee, D.& Seo, Y. H. (2020). An on-chip learning method for neuromorphic systems based on non-ideal synapse devices. *Electronics*, *9*(11), 1946. 32051761

Lee, J. H., Delbruck, T., & Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, *10*, 508.

Lisitsa, D., & Zhilenkov, A. A. (2017). Prospects for the development and application of spiking neural networks. In *Proceedings of the IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering* (pp. 926–929). Piscataway, NJ: IEEE.

Liu, Y., Jin, Y., & Li, P. (2018). Online adaptation and energy minimization for hardware recurrent spiking neural networks. *ACM Journal on Emerging Technologies in Computing Systems*, *14*(1), 1–21.

Liu, Y., Yenamachintala, S. S., & Li, P. (2019). Energy-efficient FPGA spiking neural accelerators with supervised and unsupervised spike-timing-dependent-plasticity. *ACM Journal on Emerging Technologies in Computing Systems*, *15*(3), 1–19.

López-Vázquez, G., Ornelas-Rodríguez, M., Espinal, A., Soria-Alcaraz, J. A., Rojas-Domínguez, A., Puga-Soberanes, H. J., . . . Rostro-González, H. (2019). Evolving random topologies of spiking neural networks for pattern recognition. *Computer Science and Information Technology*, *9*(7), 41–56.

Losh, M., & Llamocca, D. (2019). A low-power spike-like neural network design. *Electronics*, *8*(12), 1479. 10.3390/electronics8121479

Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, *10*(9), 1659–1671. 10.1016/S0893-6080(97)00011-7

Markram, H., Gerstner, W., & Sjöström, P. J. (2011). A history of spike-timing-dependent plasticity. *Front. Synaptic Neurosci.*, *3*, 4. 10.3389/fnsyn.2011.00004, PubMed: 22007168

Mashford, B. S., Yepes, A. J., Kiral-Kornek, I., Tang, J., & Harrer, S. (2017). Neural-network-based analysis of EEG data using the neuromorphic TrueNorth chip for brain-machine interfaces. *IBM Journal of Research and Development*, *61*(2/3), 7–1.

Mead, C. (1990). Neuromorphic electronic systems. In *Proceedings of the IEEE*, *78*(10), 1629–1636. 10.1109/5.58356

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., . . . Brezzo, B. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, *345*(6197), 668–673. 10.1126/science.1254642, PubMed: 25104385

Mirsadeghi, M., Shalchian, M., Kheradpisheh, S. R., & Masquelier, T. (2021). STiDi-BP: Spike time displacement based error backpropagation in multilayer spiking neural networks. *Neurocomputing*, *427*, 131–140. 10.1016/j.neucom.2020.11.052

Molin, J. L., Figliolia, T., Sanni, K., Doxas, I., Andreou, A., & Etienne-Cummings, R. (2015). FPGA emulation of a spike-based, stochastic system for real-time image dewarping. In *Proceedings of the IEEE 58th International Midwest Symposium on Circuits and Systems* (pp. 1–4). Piscataway, NJ: IEEE.

Mostafa, H. (2017). Supervised learning based on temporal coding in spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(7), 3227–3235. 28783639

Mostafa, H., Pedroni, B. U., Sheik, S., & Cauwenberghs, G. (2017). Fast classification using sparsely active spiking networks. In *2017 IEEE International Symposium on Circuits and Systems* (pp. 1–4).

Mozafari, M., Ganjtabesh, M., Nowzari-Dalini, A., Thorpe, S. J., & Masquelier, T. (2018). *Combining STDP and reward-modulated STDP in deep convolutional spiking neural networks for digit recognition*. arXiv:1804.00227.

Mozafari, M., Kheradpisheh, S. R., Masquelier, T., Nowzari-Dalini, A., & Ganjtabesh, M. (2018). First-spike-based visual categorization using reward-modulated STDP. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(12), 6178–6190. 10.1109/TNNLS.2018.2826721, PubMed: 29993898

Nallathambi, A., & Chandrachoodan, N. (2020). *Probabilistic spike propagation for FPGA implementation of spiking neural networks.* arXiv:2001.09725.

Naveros, F., Garrido, J. A., Carrillo, R. R., Ros, E., & Luque, N. R. (2017). Event- and time-driven techniques using parallel CPU-GPU co-processing for spiking neural networks. *Frontiers in Neuroinformatics*, *11*, 7. 10.3389/fninf.2017.00007, PubMed: 28223930

Neil, D., & Liu, S. C. (2014). Minitaur, an event-driven FPGA-based spiking network accelerator. *IEEE Transactions on Very Large-Scale Integration Systems*, *22*(12), 2621–2628. 10.1109/TVLSI.2013.2294916

Neil, D., & Liu, S. C. (2016). Effective sensor fusion with event-based sensors and deep network architectures. In *Proceedings of the 2016 IEEE International Symposium on Circuits and Systems* (pp. 2282–2285). Piscataway, NJ: IEEE.

Nitzsche, S., Pachideh, B., Luhn, N., & Becker, J. (2021). Digital hardware implementation of optimized spiking neurons. In *Proceedings of the 2021 International Conference on Neuromorphic Computing* (pp. 126–134). New York: ACM.

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. 10.3389/fnins.2015.00437

Osswald, M., Ieng, S. H., Benosman, R., & Indiveri, G. (2017). A spiking neural network model of 3D perception for event-based neuromorphic stereo vision systems. *Scientific Reports*, *7*(1), 1–12. 10.1038/s41598-016-0028-x, PubMed: 28127051

Panchapakesan, S., Fang, Z., & Li, J. (2021). SyncNN: Evaluating and accelerating spiking neural networks on FPGAs. In *Proceedings of the 31st International Conference on Field-Programmable Logic and Applications* (pp. 286–293). Piscataway, NJ: IEEE.

Paugam-Moisy, H. (2006). *Spiking neuron networks a survey*. paugam-idiap-rr-06-11.pdf

Paugam-Moisy, H., & Bohte, S. M. (2012). Computing with spiking neuron networks. In G. Rozenberg, T. Bäck, & J. N. Kok, (Eds.), *Handbook of natural computing* (vol. 1, pp. 1–47). Berlin: Springer.

Pavlidis, N. G., Tasoulis, O. K., Plagianakos, V. P., Nikiforidis, G., & Vrahatis, M. N. (2005). Spiking neural network training using evolutionary algorithms. In *Proceedings of the 2005 IEEE International Joint Conference on Neural Networks* (vol. 4, pp. 2190–2194). Piscataway, NJ: IEEE. 10.1109/IJCNN.2005.1556240

Perez-Peña, F., Cifredo-Chacon, M. A., & Quiros-Olozabal, A. (2020). Digital neuromorphic real-time platform. *Neurocomputing*, *371*, 91–99.

Pfeiffer, M., & Pfeil, T. (2018). Deep learning with spiking neurons: Opportunities and challenges. *Frontiers in Neuroscience*, *12*, 774. 10.3389/fnins.2018.00774, PubMed: 30410432

Qi, Y., Zhang, B., Taha, T. M., Chen, H., & Hasan, R. (2014). FPGA design of a multicore neuromorphic processing system. In *Proceedings of the IEEE National Aerospace and Electronics Conference* (pp. 255–258). Piscataway, NJ: IEEE.

Qu, L., Zhao, Z., Wang, L., & Wang, Y. (2020). Efficient and hardware-friendly methods to implement competitive learning for spiking neural networks. *Neural Computing and Applications*, *32*(17), 13479–13490.

Rahman, T. (2017). *Classification of roadside material using convolutional neural network and a proposed implementation of the network through Zedboard Zynq 7000 FPGA*. Purdue University.

Rudolph-Lilith, M., Dubois, M., & Destexhe, A. (2012). Analytical integrate- and-fire neuron models with conductance-based dynamics and realistic postsynaptic potential time course for event-driven simulation strategies. *Neural Computation*, *24*(6), 1426–1461. 10.1162/NECO_a_00278, PubMed: 22364504

Rueckauer, B., Lungu, I. A., Hu, Y., Pfeiffer, M., & Liu, S. C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, *11*, 682. 10.3389/fnins.2017.00682, PubMed: 29375284

Saleh, A. Y., Hameed, H., Najib, M.& Salleh, M. (2014). A novel hybrid algorithm of differential evolution with evolving spiking neural network for pre-synaptic neurons optimization. *International Journal of Advances in Soft Computing and Its Applications*, *6*(1), 1–16.

Schaffer, J. D. (2015). Evolving spiking neural networks: A novel growth algorithm corrects the teacher. In *Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications* (pp. 1–8). Piscataway, NJ: IEEE.

Schemmel, J., Brüderle, D., Grübl, A., Hock, M., Meier, K., & Millner, S. (2010). A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proceedings of the IEEE International Symposium on Circuits and Systems* (pp. 1947–1950). Piscataway, NJ: IEEE.

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*. arXiv:1705.06963.

Sengupta, A., Ye, Y., Wang, R., Liu, C., & Roy, K. (2018). *Going deeper in spiking neural networks: VGG and residual architectures.* arXiv:1802.02627.

Seo, J. S., & Seok, M. (2015). Digital CMOS neuromorphic processor design featuring unsupervised online learning. In *Proceedings of the IFIP/IEEE International Conference on Very Large Scale Integration* (pp. 49–51). Piscataway, NJ: IEEE.

Shahid, N., Rappon, T., & Berta, W. (2019). Applications of artificial neural networks in health care organizational decision-making: A scoping review. *PLOS One*, *14*(2), e0212356.

Shrestha, S. B., & Orchard, G. (2018). Slayer: Spike layer error reassignment in time. *Advances in neural information processing systems*, *31*.

Simeone, O. (2018). *Neuromorphic computing and learning: A stochastic signal processing perspective.* https://nms.kcl.ac.uk/osvaldo.simeone/SNN_v30.pdf

Sironi, A., Brambilla, M., Bourdis, N., Lagorce, X., & Benosman, R. (2018). HATS: Histograms of averaged time surfaces for robust event-based object classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* Piscataway, NJ: IEEE. 10.1109/CVPR.2018.00186

Stimberg, M., Brette, R., & Goodman, D. F. (2019). Brian 2, an intuitive and efficient neural simulator. *eLife*, *8*, e47314. 10.7554/eLife.47314, PubMed: 31429824

Stimberg, M., Goodman, D. F., & Nowotny, T. (2020). Brian2GeNN: Accelerating spiking neural network simulations with graphics hardware. *Scientific Reports*, *10*(1), 1–12. 10.1038/s41598-019-54957-7, PubMed: 31913322

Stromatias, E., Soto, M., Serrano-Gotarredona, T., & Linares-Barranco, B. (2017). An event-driven classifier for spiking neural networks fed with synthetic or dynamic vision sensor data. *Frontiers in Neuroscience*, *11*, 350. 10.3389/fnins.2017.00350, PubMed: 28701911

Tanaka, G., Yamane, T., Héroux, J. B., Nakane, R., Kanazawa, N., Takeda, S., . . . Hirose, A. (2019). Recent advances in physical reservoir computing: A review. *Neural Networks*, *115*, 100–123. 10.1016/j.neunet.2019.03.005, PubMed: 30981085

Tang, G., & Michmizos, K. P. (2018). Gridbot: An autonomous robot controlled by a spiking neural network mimicking the brain's navigational system. In *Proceedings of the International Conference on Neuromorphic Systems* (pp. 1–8). New York: ACM.

Tavanaei, A., Kirby, Z., & Maida, A. S. (2018). Training spiking ConvNets by STDP and gradient descent. In *Proceedings of the International Joint Conference on Neural Networks* (pp. 1–8). Piscataway, NJ: IEEE.

Thiele, J. C. (2019). *Deep learning in event-based neuromorphic systems*. PhD diss., Université Paris Saclay.

Thorpe, S., Delorme, A., & Van Rullen, R. (2001). Spike-based strategies for rapid processing. *Neural Networks*, *14*(6–7), 715–725. 10.1016/S0893-6080(01)00083-1, PubMed: 11665765

Tkačik, G., Prentice, J. S., Balasubramanian, V., & Schneidman, E. (2010). Optimal population coding by noisy spiking neurons. In *Proceedings of the National Academy of Sciences*, *107*(32), 14419–14424.

Vazquez, R. A. (2010). Izhikevich neuron model and its application in pattern recognition. *Australian Journal of Intelligent Information Processing Systems*, *11*, 35–40.

Vázquez, R. A., & Garro, B. A. (2011). Training spiking neurons by means of particle swarm optimization. In *Proceedings of the International Conference in Swarm Intelligence* (pp. 242–249). Berlin: Springer.

Vitay, J., Dinkelbach, H. Ü., & Hamker, F. H. (2015). ANNarchy: A code generation approach to neural simulations on parallel hardware. *Frontiers in Neuroinformatics*, *9*, 19. 10.3389/fninf.2015.00019, PubMed: 26283957

Walter, F., Röhrbein, F., & Knoll, A. (2015). Neuromorphic implementations of neurobiological learning algorithms for spiking neural networks. *Neural Networks*, *72*, 152–167. 10.1016/j.neunet.2015.07.004, PubMed: 26422422

Wang, Q., Jin, Y., & Li, P. (2015). General-purpose LSM learning processor architecture and theoretically guided design space exploration. In *Proceedings of the 2015 IEEE Biomedical Circuits and Systems Conference* (pp. 1–4). Piscataway, NJ: IEEE.

Wang, Q., Li, Y., Shao, B., Dey, S., & Li, P. (2017). Energy efficient parallel neuromorphic architectures with approximate arithmetic on FPGA. *Neurocomputing*, *221*, 146–158. 10.1016/j.neucom.2016.09.071

Wang, Y., Xu, Y., Yan, R., & Tang, H. (2020). Deep spiking neural networks with binary weights for object recognition. *IEEE Transactions on Cognitive and Developmental Systems*, *13*(3), 514–523. 10.1109/TCDS.2020.2971655

Wei, H., Bu, Y., & Dai, D. (2017). A decision-making model based on a spiking neural circuit and synaptic plasticity. *Cognitive Neurodynamics*, *11*(5), 415–431. 10.1007/s11571-017-9436-2, PubMed: 29067130

Wu, S., Amari, S. I., & Nakahara, H. (2002). Population coding and decoding in a neural field: A computational study. *Neural Computation*, *14*(5), 999–1026. 10.1162/089976602753633367, PubMed: 11972905

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*, 331.

Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., & Shi, L. (2019). Direct training for spiking neural networks: Faster, larger, better. In *Proceedings of the AAAI Conference on Artificial Intelligence* (vol. 33, pp. 1311–1318). Menlo Park, CA: AAAI Press. 10.1609/aaai.v33i01.33011311

Xu, Q., Peng, J., Shen, J., Tang, H., & Pan, G. (2020). Deep CovDenseSNN: A hierarchical event-driven dynamic framework with spiking neurons in noisy environment. *Neural Networks*, *121*, 512–519. 10.1016/j.neunet.2019.08.034, PubMed: 31733521

Xu, Y., Tang, H., Xing, J., & Li, H. (2017). Spike trains encoding and threshold rescaling method for deep spiking neural networks. In *Proceedings of the 2017 IEEE Symposium Series on Computational Intelligence* (pp. 1–6). Piscataway, NJ: IEEE.

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*, *43*, 99–113. 10.1016/j.neunet.2013.02.003, PubMed: 23500504

Yang, X., Zhang, Z., Zhu, W., Yu, S., Liu, L., & Wu, N. (2020). Deterministic conversion rule for CNNs to efficient spiking convolutional neural networks. *Science China Information Sciences*, *63*(2), 1–19.

Yao, M., Gao, H., Zhao, G., Wang, D., Lin, Y., Yang, Z., & Li, G. (2021). Temporal-wise attention spiking neural networks for event streams classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 10221–10230). Piscataway, NJ: IEEE

Yavuz, E., Turner, J., & Nowotny, T. (2016). GeNN: A code generation framework for accelerated brain simulations. *Scientific Reports*, *6*(1), 1–14. 10.1038/srep18854, PubMed: 28442746

Yi, Y., Liao, Y., Wang, B., Fu, X., Shen, F., Hou, H., & Liu, L. (2016). FPGA based spike-time dependent encoder and reservoir design in neuromorphic computing processors. *Microprocessors and Microsystems*, *46*, 175–183. 10.1016/j.micpro.2016.03.009

Yousefzadeh, A., Orchard, G., Stromatias, E., Serrano-Gotarredona, T., & Linares-Barranco, B. (2018). Hybrid neural network, an efficient low–power digital hardware implementation of event-based artificial neural network. In *Proceedings of the IEEE International Symposium on Circuits and Systems* (pp. 1–5). Piscataway, NJ: IEEE.

Yousefzadeh, A., Serrano-Gotarredona, T., & Linares-Barranco, B. (2015). Fast pipeline 128 $\times$ 128 pixel spiking convolution core for event-driven vision processing in FPGAs. In *Proceedings of the International Conference on Event-Based Control, Communication, and Signal Processing* (pp. 1–8). Piscataway, NJ: IEEE.

Yusuf, Z. M., Hamed, H. N. A., Yusuf, L. M., & Isa, M. A. (2017). Evolving spiking neural network (ESNN) and harmony search algorithm (HSA) for parameter optimization. In *Proceedings of the Sixth International Conference on Electrical Engineering and Informatics* (pp. 1–6). Piscataway, NJ: IEEE.

Zhang, C. M., Qiao, G. C., Hu, S. G., Wang, J. J., Liu, Z. W., Liu, Y. A., . . . & Liu, Y. (2019). A versatile neuromorphic system based on simple neuron model. *AIP Advances*, *9*(1), 015324.

Zhang, G., Li, B., Wu, J., Wang, R., Lan, Y., Sun, L., . . . & Chen, Y. (2020). A low-cost and high-speed hardware implementation of spiking neural network. *Neurocomputing*, *382*, 106–115. 10.1016/j.neucom.2019.11.045

Zhang, Z., & Kouzani, A. Z. (2020). Implementation of DNNs on IoT devices. *Neural Computing and Applications*, *32*(5), 1327–1356. 10.1007/s00521-019-04550-w

Zheng, H., Wu, Y., Deng, L., Hu, Y. & Li, G. (2020). *Going deeper with directly trained larger spiking neural networks.* arXiv:2011.05280.

Zheng, N., & Mazumder, P. (2018a). Online supervised learning for hardware-based multilayer spiking neural networks through the modulation of weight-dependent spike-timing-dependent plasticity. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(9), 4287–4302. 10.1109/TNNLS.2017.2761335, PubMed: 29990088

Zheng, N., & Mazumder, P. (2018b). A low-power hardware architecture for on-line supervised learning in multi-layer spiking neural networks. In *Proceedings of the 2018 IEEE International Symposium on Circuits and Systems* (pp. 1–5). Piscataway, NJ: IEEE.

Zhou, S., Chen, Y., Ye, Q., & Li, J. (2019). *Direct training based spiking convolutional neural networks for object recognition*. arXiv:1909.10837.

Received August 1, 2021; accepted January 18, 2022.