A Survey on Spiking Neural Networks

Chan Sik Han and Keon Myung Lee

Department of Computer Science, Chungbuk National University, Cheongju, Korea

Abstract

Spiking neural networks (SNNs) have attracted attention as the third generation of neural networks for their promising characteristics of energy-efficiency and biological plausibility. The diversity of spiking neuron models and architectures have made various learning algorithms developed. This paper provides a gentle survey of SNNs to give an overview of what they are and how they are trained. It first presents how biological neurons works and how they are mathematically modelled specially in differential equations. Next it categorizes the learning algorithms of SNNs into groups and presents how their representative algorithms work. Then it briefly describe the neuromorphic hardware on which SNNs run.

Keywords: Spiking neural network, Deep learning, Neural network, Machine learning, Learning algorithms

1. Introduction

Deep neural networks (DNNs) have made great successes in various formerly difficult tasks such as vision, speech, natural language, and games [1–5]. They have many layers and parameters and thus require huge computing resources, resulting in high energy consumption. Spiking neural networks (SNNs) are computation models which mimic biological neural networks in a more similar way than artificial neural networks (ANNs) [6]. In SNNs, neurons called spiking neurons receive spikes from other neurons and generates spikes. The SNNs can be implemented in an energy-efficient way since they are operated according to spikes sparsely generated. With this expectation, machine learning people have made efforts to develop SNNs with comparable performance to DNNs [7]. On the other hand, in neuroscience, researchers have been interested in simulating brain-scale SNNs to study brain functions. The diversity of neuron models for biological neurons, spike coding methods, and network architectures have made various learning algorithms developed for SNNs.

This survey focuses on the SNNs in the perspective of machine learning. In Section 2, for the better understanding of SNNs, we first explain the structure of biological neurons and their behaviors, and then several mathematical models of spiking neurons. In conventional ANNs and DNNs, a simple neuron model is used of which operation is expressed as the application of activation function to the weighted sum of its input. On the contrary, in SNNs, spike neurons can be modelled in various ways using differential equations with several hyperparameters like threshold level, time constant, refractory time, latency, and so on. We also present the notion of synaptic plasticity which is used to model local learning occurred in SNNs. Spiking neurons receive and generate a spike or a spike train. Hence all input and output for training should be encoded in spikes.

Received: Nov. 14, 2021

Revised : Nov. 14, 2021

Accepted: Dec. 7, 2021

Correspondence to: Keon Myung Lee
 (kmllee@cbnu.ac.kr)

©The Korean Institute of Intelligent Systems

©This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (<http://creativecommons.org/licenses/by-nc/3.0/>) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

We present the neuron coding methods for input and output of SNNs. Then we describe how to simulate SNNs in software and what issues there are in the simulation.

In Section 3, we deal with learning methods for SNNs. They can be grouped into the unsupervised methods and the supervised methods. The unsupervised methods include some local learning algorithms. The supervised methods can be categorized into the direct methods, the ANN-SNN conversion methods, and the hybrid methods. The direct methods train directly the SNNs despite of the nonlinearity of the SNN activation functions using some surrogate functions or imposed restrictions. The ANN-SNN methods first train an ANN, and then convert the trained ANN into an SNN with the same architecture as the ANN. The hybrid methods first apply an ANN-SNN method and then fine-tune the converted SNN with a direct method.

Section 4 briefly reviews the neuromorphic hardware which have been developed for neuroscience studies and machine learning. Despite of various publications of neuromorphic hardware, there are few neuromorphic hardware on the market. In Section 5, we draw the conclusions along with the current research trends.

2. Biological Neurons and Spiking Neuron Models

This section presents the behaviors of biological neurons, the computational models of spiking neurons, the notion of synaptic plasticity for brain learning, the architectures of SNNs, neural coding methods for input/output representations, and simulation of SNNs.

2.1 Biological Neuron and Its Behaviors

Brains consist of a large number of nerve cells called neurons (e.g., approximately 86 billions in the human brain) which are heavily interconnected each other [8]. Neurons communicate with each other via spikes also known as action potentials [9]. Action potentials are the electrical impulses which have a short duration of around 10 ms in brain. A neuron has three main components: dendrites, an axon, and a cell body called soma [10]. Dendrites are tree-shaped component which receive signals from axon of other neurons. Upon arriving spikes through dendrites, a soma increases or decreases its membrane potential and sends an action potential down an axon when the membrane potential reaches its threshold level. Axons are long nerve fiber which conducts action potentials away from the soma. When an action potential is transmitted through an axon,

Figure 1. Phase shifts of membrane potential [13].

synaptic vesicles move towards an axon terminal and release neurotransmitters in the synaptic cleft [11]. The synaptic cleft is the small gap that separates the presynaptic and postsynaptic neurons. A synapse is the junction between the axon of one neuron and a dendrite of another, through which the two neurons communicate with neurotransmitters. Neurotransmitters are chemicals that influence a receiving neuron either to promote the generation of spikes or to prevent it by binding with their corresponding receptors of the dendrite of the receiving neuron [12]. Such bindings open or close the ion channels which cause the membrane potential to change. Synaptic weight refers to the connection strength at a synapse which is determined by the amount of released neurotransmitters, the amount of receptors to absorb, and the signal propagation resistance in the axons and the dendrite.

Membrane potential experiences a sequence of state phases when generating an action potential: resting state, depolarization, repolarization, and hyperpolarization, as shown in Figure 1 [13]. There exists the imbalance of electrical charges between the interior of neurons and their surroundings in a resting state. The resting state is the ground value of trans-membrane voltage which is negatively charged potential, approximately -70 mV. When neurotransmitter glutamate binds with the AMPA receptor, the sodium channels open, which results in a large influx of sodium ions. It causes a rapid rise of the membrane potential and when the membrane potential reaches the threshold level, the action potential starts to be generated. This phase of a rise of potential from negative towards positive is called depolarization. When membrane potential increases sufficiently enough to open the potassium channels, potassium ions start to move out of the neuron, which results in a rapid drop of membrane potential. The phase of such potential drop is called repolarization be-

cause membrane potential gets negated charged back. Action potential is shaped during the phase shift from depolarization and repolarization. Slow close of potassium channels make an undershoot of membrane potential, which results in a period of potential lower than the potential of resting potential. The phase of such lower potential is called hyperpolarization. The period of hyperpolarization is called refractory time. The hyperpolarized potential gradually returns to the resting potential state by ion movement through the membrane.

Membrane potential increases when a spike is received through an excitatory synapse, whereas it decreases when a spike is received through an inhibitory synapse. Membrane potential leaks away exponentially over time.

2.2 Spiking Neuron Models

Several mathematical models for spiking neurons have been developed to describe the characteristics of membrane potential change as shown in Figure 1. A spiking neuron receives spikes, accumulates them into its membrane potential, generates a spike when its potential reaches the threshold, and consumes its potential. In the spiking neuron models, there are Hodgkin-Huxley model, leaky integrate-and-fire (LIF) model, integrate-and-fire (IF) model, spike response model (SRM), Izhikevich's model, FitzHugh-Nagumo (FHN) model, and so on.

The Hodgkin-Huxley model [14] describes membrane potential in terms of sodium channel potential, potassium channel potential, and leak potential. For a neuron with sodium and potassium channels, it models the total current I passing through the membrane as follows:

$$I = C_m \frac{dV_m}{dt} + g_K(V_m - V_K) + g_{Na}(V_m - V_{Na}) + g_L(V_m - V_L), \quad (1)$$

where C_m is the membrane capacitance, g_K and g_{Na} are the

Figure 2. Hodgkin-Huxley model.

Figure 3. Leaky integrate-and-fire (LIF) model.

potassium and sodium conductance, respectively, V_K and V_{Na} are the potassium and sodium potentials, respectively, and g_L and V_L are the leak conductance and leak potential, respectively. The model can be expressed in the electric circuit shown in Figure 2. It is complicated and computationally expensive due to its differential equation and parameters.

LIF model [15] describes the membrane potential without introducing channel potentials in a simpler way than the Hodgkin-Huxley model. The LIF model integrates its injected current into membrane potential but allows membrane potential to slowly leak over time.

The LIF model can be expressed in the electric circuit shown in Figure 3. Its membrane potential V_m can be characterized by the following differential equation:

$$C_m \frac{dV_m}{dt} = -\frac{V_m - V_{rest}}{R_m} + I, \quad (2)$$

where I is the injected current, V_{rest} is the resting potential, R_m is the resistance, and C_m is the capacity. When $\tau_m = C_m R_m$ is used, Eq. (2) can be expressed as follows:

$$\tau_m \frac{dV_m}{dt} = V_{rest} - V_m + R_m I, \quad (3)$$

where τ_m is called the time constant, which controls how quickly the membrane potential changes. A solution to the above equation is as follows:

$$V(t) = V_{rest} + \frac{1}{C_m} \int_0^t e^{(s-t)/\tau_m} I(s) ds. \quad (4)$$

Due to the integral term of Eq. (4), the following difference equation is usually used in its implementation.

$$V[t+1] = V[t] + \Delta t \left(\frac{V_{rest} - V[t] + R_m I[t]}{\tau_m} \right). \quad (5)$$

The IF model is a simplified version of the LIF model, in

which the leakage of membrane potential is ignored. It is somewhat weak in expressing the behavior of membrane potential in biological neurons, yet its simplicity can be beneficial in the computational and implementation aspect, especially in hardware. The difference equation for the IF model is as follows:

$$V[t+1] = V[t] + \Delta t (V_{rest} - V[t] + R_m I[t]). \quad (6)$$

The SRM is a generalization of the LIF model of which equation is formulated using filters instead of differential equation, as follows [16]:

$$V(t) = \eta(t - \hat{t}) + \int_0^\infty \kappa(t - \hat{t}, s) I(t - s) ds, \quad (7)$$

where $\hat{t}$ is the firing time of the last spike, η describes the shape of the action potential and its after-spike potential, κ is the linear response to an input pulse, and $I(t)$ is an injected current. Here the functions η and κ are called kernel. The model allows to model refractoriness which controls the inactiveness after spike generation.

The Izhikevich model [17] is a generalized neuron model that can generate most recognized firing patterns of biological neurons, which is as biologically plausible as the Hodgkin–Huxley model, yet as computationally efficient as the IF model. The model is expressed in the following two differential equations:

$$\frac{dv}{dt} = 0.04v^2(t) + 5v(t) + 140 - u(t) + I(t), \quad (8)$$

$$\frac{du}{dt} = a(bv(t) - u(t)), \quad (9)$$

with the auxiliary after-spike resetting as follows:

$$\text{if } v \geq 30 \text{ mV, then } \begin{cases} v \leftarrow c, \\ u \leftarrow u + d, \end{cases}$$

where $v(t)$ is the membrane potential, $u(t)$ is a membrane recovery variable, a , b , c , and d are the parameters that control the shape of membrane potential as shown in Figure 4. a controls the decay rate of $u(t)$, b controls the sensitivity of $u(t)$ to the subthreshold fluctuations of $v(t)$, c is the after-spike reset value of $v(t)$, and d controls the after-spike reset value of $u(t)$.

The FHN model is a simplified model of the Hodgkin-Huxley model, which is expressed in the following differential equation

Figure 4. Izhikevich model [18].

tions [18]:

$$\frac{dv}{dt} = v(t) - \frac{v^3(t)}{3} - w(t) + I_{ext}, \quad (10)$$

$$\frac{dw}{dt} = v(t) + a - bw(t), \quad (11)$$

where $v(t)$ is the membrane potential, $w(t)$ is an auxiliary function, I_{ext} is the injected current, and a and b are the parameters to control the shape of membrane potential.

There are variants of the above-mentioned neuron models and other models. They usually use Hodgkin-Huxley model, Izhikevich's model, FHN model, and their variants in neuroscience, whereas they usually use LIF mode, IF model, and SRM in machine learning.

2.3 Synaptic Plasticity

Synaptic strengths, i.e., synaptic weights, between neurons strongly affect the behaviors of the brain. Synaptic plasticity is the ability of synapses to strengthen or weaken over time, which is widely believed to contribute to learning and memory in the brain. Spike-timing-dependent plasticity (STDP) is a biological process that adjusts synaptic weight by tight temporal correlations between the spikes of presynaptic and postsynaptic neurons [19]. There are two major phenomena in synaptic plasticity: long-term potentiation (LTP) and long-term depression (LTD) [20]. LTP is a persistent strengthening of synapses, whereas LTD is a persistent weakening of synapses, on recent patterns of spikes between presynaptic and postsynaptic neurons.

According to STDP, repeated presynaptic spike arrival a few milliseconds prior to postsynaptic spikes leads to LTP of the synapses in many synapse types. On the other hand, repeated presynaptic spike arrival after postsynaptic spikes leads to LTD of the synapses. Figure 5 shows an STDP function that plots the change of synaptic weight as a function of the relative timing of presynaptic and postsynaptic spikes. In the figure, x -axis indicates a value of relative timing between presynaptic spike arrival and postsynaptic spike firing, and y -axis indicates the

Figure 5. An spike-timing-dependent plasticity (STDP) function.

relative change of synaptic weight.

Neuroscientists have paid attention to how to model the STDP because STDP seems to play key roles in learning and information storage in the brain. There have been proposed several mathematical models for STDP. Machine learning people have also tried to apply STDP to train SNN models.

2.4 Spiking Neural Networks

SNNs are ANNs that consist of spiking neurons, which can more closely mimic biological neural networks. SNNs incorporate the concept of time into their operations in which operations are carried out in a spiking time-dependent manner. In conventional ANNs, neurons transmit information at each propagation cycle regardless of their activation value. On the contrary, in SNNs, neurons transmit information (i.e., spikes) to postsynaptic neurons only when their membrane potential reaches the threshold level. That is, only when a spike is generated at a neuron, it propagates the spike into its postsynaptic neurons. This makes it possible to save the energy consumption because only pulse-shaped spikes are propagated only when spikes are generated. SNNs are called the third generation neuron networks, which have attracted attention as an energy-efficient neural network model in engineering aspects [21].

In SNNs, spiking neurons are connected with synapses by which the transmitted spikes are either amplified or attenuated. There are two types of synapses: excitatory and inhibitory synapses [22]. When a spike is propagated through an excitatory synapse, the membrane potential of the receiving neuron is increased. When a neuron receives a spike through an inhibitory synapse, its membrane potential is decreased.

SNNs can be organized into layered architectures, recurrent architectures, or hybrid architectures [23]. In a hybrid architec-

Figure 6. A Synfire chain architecture [24].

Figure 7. A liquid state machine architecture [25].

ture, some subpopulations are layered, and others are recurrent. As the hybrid architectures, there are Synfire chain, liquid state machine (LSM), and so on. A Synfire chain has a multi-layered architecture each of which layer is organized into a recurrent network, as shown in Figure 6 [24]. A LSM is a large, recurrent network of spiking neurons some of which receive inputs and some of which read out their value as the output values as shown in Figure 7 [25]. In an LSM, the weights connected to output neurons are to be trained and all other weights are initialized and fixed.

Neuroscientists have had interest in building nearly human brain-sized neural networks and analyzing their behaviors to understand the mechanisms of various functions in the brain [26]. They have dealt with complex SNNs which are organized into recurrent architectures or hybrid architectures. For such complex networks, there are not yet successful learning algorithms except the STDP or its variants. STDP and its variants are not powerful enough to train complex SNNs. Hence, neuroscientists are usually not so much interested in learning the SNNs. They simulate the operations of huge SNNs and analyze their characteristic patterns. Machine learning model developers are interested in developing SNN models with high accuracy and low energy consumption. They have been usually dealing with layered SNN models and have developed various learning algorithms for SNNs.

2.5 Neural Coding Methods

In SNNs, spiking neurons receive and produce spike signals. Hence all inputs to SNNs should be encoded into a spike or a spike train which is a sequence of spikes spread over the time dimension. On the other hand, the outputs of SNNs are a spike or a spike train. Such outputs are also needed to be decoded into an understandable format like scalar values. These kinds of encoding and decoding are called neural coding.

A spike occurring at time point t is mathematically expressed as Dirac delta function $\delta(t)$ which has the following properties: $\delta(x) = 0$ if $x \neq t$, and $\int_{t-\epsilon}^{t+\epsilon} x dx = 1$ for $\epsilon > 0$. A spike train $S_i(t)$ for a neuron i is expressed in a sequence of the occurrence time points of spikes as follows:

$$S_i(t) = \sum_f \delta(t - t_i^{(f)}), \quad (12)$$

where $t_i^{(f)}$ indicates the occurrence time point of the f -th spike for the neuron i .

The neural encoding methods can be classified into rate coding, temporal coding, population coding, and direct input coding [23]. In rate coding, a value is represented as a spike train of which firing rate is proportional to the value [27]. To generate a spike train of a specific firing rate, we can distribute uniformly as many spikes as (the firing rate) $\times$ (the latency) with some random perturbation to the occurrence time of spikes, over the time span of the latency. The latency indicates the number of time steps during which a spike train is presented. To generate a spike train, we can also use a Poisson distribution of which mean corresponds to the firing rate [28]. In addition, we can use a stochastic encoding method which normalizes input values into the interval $[0, 1]$ and uses the normalized values as the probability to generate a spike at each time step.

The temporal coding methods generate a single spike at a specific time point to represent an input value. In the temporal coding methods, there are time-to-first spike code, rank code, and so on [23]. The time-to-first code (a.k.a, latency code) represents a value as such a spike that the larger the input value, the earlier the onset of the induced spike. In the rank code (a.k.a. spike-order code), all the values are first sorted in the decreasing order, and then each of those values is assigned a spike time in a way that the order of the assigned spike times follows the order of those value. The population code makes a population (i.e., set) of input neurons take care of an input value together [29]. Each neuron has its own receptive field of which sensitivity is somewhat a Gaussian function-based as shown

Figure 8. Population coding.

in Figure 6. Once an input value is given, each input neuron generates a spike at the time corresponding to its Gaussian-like function value to the input.

In direct input coding [30], we use the input value as a constant current to its corresponding spiking neuron without generation of spikes. The input values are usually normalized into a specific interval like $[0, 1]$.

When the neurons of the output layer come to have the final membrane potentials, they produce spike(s) as the output. To interpret the output spike(s), we use the decoding methods such as spike counting, rate-based coding, temporal coding, and so on. For the SNNs of regression tasks, the spike counting method regards the count of spikes in the output nodes as the estimated value. For the SNNs of classification tasks, the rate-based decoding method selects the node label with the maximum frequency of spikes as the output. For SNNs of which the output neurons generate at most one spike, the node label with the earliest firing time is selected as the class label. Some SNNs use as the output value the membrane potential of output neurons without generating spikes [31,32]. They directly apply the softmax function to the membrane potential values in order to get the probability of classes.

When a spike train is fed into an LIF neuron, the membrane potential of the neuron is expressed in the following differential equation:

$$C_m \frac{dV_m}{dt} = -\frac{V_m - V_{rest}}{R_m} + \sum_i w_i \sum_f \delta_i(t - t_i^{(f)}), \quad (13)$$

where w_i is the synaptic weight for the connection with a presynaptic neuron i , $t_i^{(f)}$ is a firing time of the presynaptic neuron, and the conductance for the spikes is assumed to be 1 G (siemens). Compared to Eq. (2), Eq. (13) tells that neurons receive the weighted sum of spikes as the external current.

2.6 Simulation of Spiking Neural Networks

SNNs can be executed on a neuromorphic hardware which is specialized for SNN model execution, or simulated in software. Only a few neuromorphic hardware are available on the market. Most of them support a limited architecture of SNNs. Hence, in the training and testing phases, software simulations are widely used. The simulation methods can be categorized into synchronous simulation and asynchronous simulation.

In synchronous simulations, also known as clock-driven simulations, all neurons are updated simultaneously at every tick of a clock. In asynchronous simulations, also known as event-driven simulations, neurons are updated only when they receive or produce a spike [33]. In the simulations, spikes are represented as 0 and 1, and neurons have a variable to keep the value of their membrane potential which is updated by the differential or difference equations corresponding to the adopted neuron model.

The differential equations can be rather simple as in Eq. (3), but they can be very complicated to describe the detailed behaviors of ion channels in the neurons. In the simulations of neuroscience studies, they use such complicated differential equations as in the Hodgkin-Huxley model to update the membrane potential of neurons. On the other hand, in the simulations of machine learning applications, such simpler differential equations as in the LIF model are used.

When an SNN processes its input, the input is fed into an SNN with a specified latency. A spike pattern for the input is produced by the adopted neural encoding scheme. In synchronous simulations, it is usually assumed that at each time step (i.e., a tick of clock) an input signal passes through the entire SNN from the input neurons to the output neurons. Machine learning applications usually use the synchronous simulations for training SNN models, whereas neuroscience studies mainly use asynchronous simulations for investigating the brain functions. Input to an SNN is given in a spike or spike train generated by the neural encode scheme. Such spike or spike train is generated either earlier on or on the fly depending on the adopted encoding scheme. When a uniform or Poisson distribution is used to generate a spike train, it is needed to generate the entire spike train earlier on to feed the input to an SNN. When a stochastic encoding method is used, the spikes of a spike train can be, on the fly, generated and fed into the input of an SNN.

When an SNN model is organized for a simulation, the ensemble and connection paradigm can be used. The ensemble component is used to represent a group of neurons that operate

together. The connection component is used to connect an ensemble to another or to the same ensemble. When a connection is made from an ensemble to itself, the ensemble becomes a recurrent network. When ensembles are arranged in a chain with connections, a multi-layered SNN is formed.

For the simulations of SNNs, there are various hyperparameters to control the behaviors of spiking neurons as follows: resting potential, minimum potential, threshold level, spike potential, refractory period, membrane time constant, latency, and so on. Simpler models for machine learning applications have a few hyperparameters, while complicated models for neuroscience studies have more hyperparameters. As the markup languages for exchanging the SNN models across the platforms, there are NeuroML, ONNX, NNEF, and so on. As the simulator-independent languages for designing and simulating SNNs, there are PyNN, EDLUT, and Nengo. There are also domain-specific languages such as OptiML and Corelet.

3. Learning Methods for Spiking Neural Networks

The learning methods for SNNs can be roughly categorized into unsupervised learning and supervised learning.

3.1 Unsupervised Learning

The typical unsupervised learning method for an SNN is a local training method such as STDP method. As shown in Figure 5, the STDP algorithm adjusts synaptic weights in such a way that synaptic plasticity is controlled by timing difference between presynaptic and postsynaptic neurons' spike times. There are several variants of STDP algorithms. The vanilla STDP method uses the following update quantity Δw_{ji} for the weight w_{ji} from the presynaptic neuron j to the postsynaptic neuron i :

$$\Delta w_{ji} = \sum_{f=1}^F \sum_{n=1}^N W(t_i^{(n)} - t_j^{(f)}), \quad (14)$$

where $t_i^{(n)}$ is the n -th spiking time for neuron i , $t_j^{(f)}$ is the f -th spiking time for neuron j , and $W()$ is an STDP function of Figure 5 which is defined as follows:

$$W(x) = \begin{cases} A_+ \exp(-x/\tau_+), & \text{for } x > 0, \\ A_- \exp(x/\tau_-), & \text{otherwise,} \end{cases} \quad (15)$$

where A_+ and A_- are constants that control the height of the curves in Figure 5, τ_+ and τ_- are time constants to control the

stiffness of the function.

There are several modifications for the vanilla STDP method. The following one is the weight change by a modified STDP method which does not use the exponential function of Eq. (15) [34]:

$$\Delta w = \eta(x_{pre} - x_{tar})(w_{max} - w)^\mu, \quad (16)$$

where x_{pre} is the presynaptic trace which models the recent presynaptic spike history of which value is increased by 1 at the arrival of a presynaptic spike, decreased exponentially for no arrival of spike. x_{tar} is the target value of the presynaptic trace at the time of a postsynaptic spike where the higher the target value, the lower the synaptic will be. η is the learning rate, w_{max} is the maximum weight, and μ determines the dependence of the update on the previous weight.

There are other variants of the STDP method as follows [34]:

$$\Delta w = \eta_{post}(x_{pre} \exp(-\beta w) - x_{tar} \exp(-\beta(w_{max} - w))), \quad (17)$$

where β is a parameter to determine the strength of the weight dependence, and η_{post} is a learning rate. The next is another variant of the STDP method.

$$\Delta w = \eta_{pre} x_{post} w^\mu, \quad (18)$$

where η_{pre} is the learning rate, and x_{post} is the postsynaptic trace defined like x_{pre} .

The Bienenstock-Cooper-Munro (BCM) rule is an unsupervised learning method with which weights are modified depending on the rates of the presynaptic and postsynaptic spikes [35]. Its update rule can be expressed as follows:

$$\Delta w = \rho_{pre} \phi(\rho_{post}, \theta), \quad (19)$$

where ρ_{pre} and ρ_{post} are the presynaptic and postsynaptic rate, respectively and θ is some threshold. The update rule decreases the synaptic weights when $\phi(\rho_{post} < \theta, \theta)$, increases the weights when $\phi(\rho_{post} > \theta, \theta)$, and makes no change when $\phi(0, \theta) = 0$. On the meanwhile, the update rule depends linearly on the presynaptic rate, but nonlinearly on the postsynaptic spike rate.

Both STDP and BCM rules are biologically-plausible unsupervised training algorithms and relatively easy to implement, yet usually not easy to apply to train high accuracy models having multiple layers.

Diehl and Cook [34] applied the STDP rule to train a handwritten digit recognition SNN which consists of one excitatory neuron layer and an inhibitory neuron layer. Each excitatory neuron receives the spikes for all the pixels as the input, that were encoded using the Poisson distribution-based method. An excitatory neuron has a connection to only one inhibitory neuron while an inhibitory neuron is connected to all excitatory neurons except the one with incoming connection to itself. All synaptic weights from input neurons to excitatory neurons are learned using STDP of Eq. (16). Their model uses the LIF neuron model and the time constant for excitatory neurons is longer than that of inhibitory neurons. Excitatory neurons are labelled with classes after training, based on their highest average response to a digit class over the entire training set.

Kheradpisheh et al. [36] used the STDP rule to train a spiking deep convolutional neural networks for object recognition. Their network has a multi-layered architecture in which convolution layer and pooling layer are interleaved, and the feature vector of the last pooling is used as the input to an support vector machine (SVM) classifier model. For an input image, there is a temporal coding cell for each pixel location. The temporal coding cells first apply the difference of Gaussian (DoG) filter to the input image, and then converts each computed contrast into a spike according to the rank order encoding method. The learning for convolutional layers is carried out layer-by-layer using the STDP rule. The last layer is the global max pooling layer applied to each channel of its preceding convolutional layer. The results of global max pooling are used as the input to a linear SVM classifier model.

3.2 Supervised Learning

The supervised learning methods can be categorized into direct training, ANN-SNN conversion, and hybrid training methods. In the direct training approach, the training methods use differentiable surrogate function in place of the discrete activation function during the training phase, and apply a gradient-based optimization technique with the surrogate function. In the ANN-SNN conversion approach, a conventional ANN is first trained for the given training data, and its weights are then used to set the weights of an SNN of the same architecture. In the hybrid approach, we first use the ANN-SNN approach to initialize the weights of an SNN, and then fine-tune the SNN with a direct training method.

Table 1. Direct training algorithms

Algorithm	Neuron model	Architecture	Input encoding	Output decoding	Features
SpikeProp (2000, [37])	SRM	Shallow network	Population code	Time-to-first code	Surrogate gradient; multiple delayed synaptic terminals
ReSuMe (2005, [38])	don't care	(FF, RNN, LSM)+ trainable single layer	Spike train	Spike train	Train the weights for the last layer; STDP & anti-STDP
PES (2011, [40])	IF/LIF model	Two-layered network	Spike train (firing rate)	Spike train (firing rate)	MSE loss for decoded value
STBP (2018, [42])	LIF	Shallow network	Spike train (rate code)	Spike train (firing rate)	BPTT-like over spatial & time domains
BP-STDP (2019, [32])	LIF	Deep network	Spike train (spike count)	Direct output (spike count)	Backpropagation + STDP
SBBP (2019, [43])	IF/LIF	Deep network	Spike train (rate code)	Direct output (membrane potential)	Surrogate gradient

3.2.1 Direct Training

The direct training methods make use of inherent characteristics of spike neurons such as spike timing. Table 1 summarizes some direct training methods in terms of their neuron type, architecture, input encoding method, output decoding method, and unique features. In the early days of SNN studies, the direct methods had attracted attention which try to mimic biological behaviors. Later algorithms has paid more attention to apply conventional neural networks' techniques like gradient-based optimization to SNNs.

SpikePro [37] is a training algorithm for a shallow SNN in which input is encoded using the population code, output is a single spike coded by the time-to-first code, the neurons are the SRM type, and each connection is made of multiple synaptic paths with different fixed delays and trainable weights as shown in Figure 9. Because the output is given in a single spike, the objective of training is to make the actual spike time as close as possible to the desired spike time. Hence the loss function is defined as $E = \frac{1}{2} \sum_j (t_j^o - t_j^d)^2$, where t_j^o is the spike time of the SNN output for the j -th training data, and t_j^d is the desired spike time for the j -th training data. To adjust the connection weights, SpikeProp uses a gradient-descent method for the loss function E . On computing the gradient, it is needed to get the derivative $\partial t_j / \partial u_j(t)$ of the spike time t_j with respect to

Figure 9. An SNN with multiple synaptic connections [37].

membrane potential $u_j(t)$, but the derivative is not directed computed. It is clear that increase in membrane potential results in earlier spike generation, hence $\partial t_j / \partial u_j(t) < 0$. SpikeProp uses a surrogate gradient for $\partial t_j / \partial u_j(t)$ as follows:

$$\frac{\partial t}{\partial u_j(t)} = -1 / \left(\frac{\partial u_j(t)}{\partial t} \right). \quad (20)$$

A remote supervised method (ReSuMe) [38] is a training algorithm for an SNN which consists of a front subnetwork and a following output layer, where the front subnetwork can be either feedforward, recurrent, or hybrid network like LSM shown in Figure 7. In ReSuMe, an SNN receives and generates a spike train, and interestingly there are teacher neurons each of which provides the information of desired spike timings for

Figure 10. An SNN for ReSuMe training [38].

its corresponding output neuron. The weights for the output neurons are trained and the teacher neurons are not connected into the SNN although they provide such supervising information as shown in Figure 10. ReSuMe adjusts the weights so as to make the spike train generated by the SNN similar to the spike train presented by the teacher neurons. For a connection weight w from a presynaptic neuron (i.e., an output neuron of the frontend subnetwork) to a postsynaptic neuron (i.e., an output neuron of the SNN), ReSuMe uses the following update rule which takes into account the correlations between spike trains:

$$\frac{dw(t)}{dt} = (S^d(t) - S^l(t)) \left(a + \int_0^\infty W(s) S^{in}(t-s) ds \right), \quad (21)$$

where $S^d(t)$, $S^{in}(t)$ and $S^l(t)$ indicate target (i.e., teacher), presynaptic, and postsynaptic spike trains, respectively, a is the parameter for the amplitude of the non-correlation contribution, $W(s)$ is a learning window defined over a time delay s between the occurring spikes. For excitatory synapses, the parameter a is positive and the window $W(s)$ has a similar shape to that of STDP. For inhibitory synapses, a is negative, and $W(s)$ has a shape similar to the anti-STDP rule of which the function is the negative of the STDP function.

The neural engineering framework (NEF) is a general methodology for building large-scale, biologically plausible, neural models of cognition [39]. It represents a vector $\mathbf{x}$ in n -dimensional vector using a population of neurons each of which activity is expressed as $a_i = G[\alpha \mathbf{e}_i \cdot \mathbf{x} + J_i]$, where $\mathbf{e}_i$ is a randomly initialized vector called encoder vector, α is a scaling factor, J_i is the constant value called the background current, and $G[\cdot]$ is a nonlinear neural activity function which computes the firing rate. The input vector $\mathbf{x}$ can be estimated from the

recent activities $a_i(t)$ of the neurons using the properly selected n -dimensional decoders $\mathbf{d}_i$ by $\hat{\mathbf{x}}(t) = \sum_i a_i(t) \mathbf{d}_i$. The decoding vectors $\mathbf{d}_i$ is determined to minimize $E = \int (\mathbf{x} - \hat{\mathbf{x}})^2 / 2$. The derivative $\partial E / \partial \mathbf{d}_i$ of E with respect to $\mathbf{d}_i$ is as follows:

$$\frac{\partial E}{\partial \mathbf{d}_i} = a_i \mathbf{E}, \quad (22)$$

where $\mathbf{E} = \mathbf{x} - \hat{\mathbf{x}}$.

Two populations of neurons can be connected to do linear or nonlinear transformation of a vector represented by the preceding population into a vector represented by the following population. When such a connection is made, the connection weight w_{ij} between a presynaptic neuron i and a postsynaptic neuron j is expressed as $w_{ij} = \alpha_j \mathbf{e}_j \cdot \mathbf{d}_i$. Prescribed error sensitivity (PES) [40] is a learning algorithm of NEF for two-layered SNNs where the desired output y^* is approximated by $y = \sum_i a_i \mathbf{d}_i$ of the last layer.

Under the assumption that $\partial E / \partial \mathbf{d}_i$ is a constant, we can convert Eq. (22) to the following standard delta rule form along with the learning rate μ :

$$\Delta \mathbf{d}_i = \mu a_i \mathbf{E}. \quad (23)$$

When $\alpha_j \mathbf{e}_j$ is multiplied to both sides of Eq. (23), we get $\Delta \mathbf{d}_i \cdot \alpha_j \mathbf{e}_j = \mu a_i \mathbf{E} \cdot \alpha_j \mathbf{e}_j$. Because $w_{ij} = \alpha_j \mathbf{e}_j \cdot \mathbf{d}_i$, we can get the PES learning rule as follows:

$$\Delta w_{ij} = \mu a_i \mathbf{E} \cdot \alpha_j \mathbf{e}_j. \quad (24)$$

Backpropagation spike-timing-dependent plasticity (BP-STDP) [32] is an STDP-based training algorithm for multilayered SNNs which may contain convolutional layers as shown in Figure 11. For the SNNs, input vectors are encoded into spike trains in which the numbers of spikes (i.e., spike counts) correspond to scalar values in the input vector. The output of the SNNs is a spike train of which spike count is the output value. The loss function for BP-STDP is the mean square error for the differences of the output spike counts of an SNN and the desired output values. BP-STDP assumes the time step size of duration ϵ in which at most a spike occurs, and trains an SNN to generate a spike only for the time step at which the target spike train has a spike.

BP-STDP uses the following weight change $\Delta w_{ih}(t)$ for the output layer neurons:

$$\Delta w_{ih}(t) = \mu \varepsilon_i(t) \sum_{t'=t-\epsilon}^t s_h(t'), \quad (25)$$

Figure 11. An SNN for BP-STDP training [32]. L_i is the target spike train, G_i is the generated spike train at node i of the output layer, G_h is the generated spike train at node h of a hidden layer, and G_j is the spike train of node j at the input layer which encodes a scalar input value.

where μ is the learning rate, and $\varepsilon_i(t)$ is defined to have the behaviors of STDP for weight updates as follows:

$$\varepsilon_i(t) = \begin{cases} 1, & \text{if } z_i(t) = 1, r_i \neq 1 \text{ in } [t - \epsilon, t], \\ -1, & \text{if } z_i(t) = 0, r_i = 1 \text{ in } [t - \epsilon, t], \\ 0, & \text{otherwise.} \end{cases} \quad (26)$$

In the above equation, $\varepsilon_i(t)$ makes the weight $w_{ih}(t)$ increase when the desired output $z_i(t)$ is 1 (i.e., presence of a spike) but the output of the SNN $r_i(t) \neq 1$ in the time step t for $[t - \epsilon, t]$. On the other hand, it makes $w_{ih}(t)$ decrease when $z_i(t) = 1$ and $r_i = 1$ in the time step t . Its update is similar to STDP. The weight change $\Delta w_{hj}(t)$ for the hidden layer neurons is as follows:

$$\Delta w_{hj}(t) = \begin{cases} \mu \varepsilon_h \sum_{t'=t-\epsilon}^t s_j(t'), & \text{if } s_h = 1 \text{ in } [t - \epsilon, t], \\ 0, & \text{otherwise,} \end{cases} \quad (27)$$

where $\varepsilon_h = \sum_i w_{ih} \cdot \varepsilon_i$ which is similar to the backpropagated error term in the error backpropagation algorithm for the conventional multilayer perceptron.

Spatio-temporal backpropagation (STBP) [42] is a direct training algorithm for a shallow fully connected or convolutional SNN with LIF neurons, which receives and generates spike trains and regards the firing rate of output spikes at the output layer as the inferred output value. It pays an attention to the spatial and temporal domains in the execution of an

Figure 12. Surrogate gradient functions which $\lim_{a_i \rightarrow 0^+} h_i(u) = dg(u)/du$ where $g(u)$ is an activation function [42].

SNN. In the spatial domain, an SNN processes its incoming spike signals from the preceding layer in a layer-by-layer manner. In the temporal domain, an SNN repeatedly updates the states of neurons during the execution latency. This temporal domain aspect is closely related to the execution of recurrent neural networks (RNNs). STBP is a gradient-based training rule derived in a similar manner to backpropagation through time (BPTT) for RNNs. It uses some surrogate gradients for the non-differentiable activation function $g(u)$ which generates a spike when its membrane potential reaches the threshold level. Figure 12 shows the surrogate functions used in STBP.

Spike-based backpropagation (SBBP) [43] is a training algorithm for an LIF-based SNN model which has no bias in its neurons, receives spike trains generated by the Poisson distribution-based rate coding method, and uses the average membrane potentials of the output neurons as the output values. Such SNNs consist of front-end convolutional layers with average pooling, and back-end fully-connected layers. Each convolution operation generates a spike only when the computed membrane potential is greater than or equal to the specified threshold level. Only when an average pooling result is greater than or equal to the specified threshold level, a spike is generated as the pooling value. Because the output of the SNNs is a scalar value (i.e., average membrane potential), the loss function E is defined as the mean square error (MSE), $E = \sum_j (o_j - d_j)^2/2$, where o_j is the output and d_j is the desired output. The derivative $\partial E / \partial w_{ij}^{l-1}$ is computed using the chain rule as follows:

$$\frac{\partial E}{\partial w_{ij}^{l-1}} = \frac{\partial E}{\partial a_j^l} \frac{\partial a_j^l}{\partial net_j^l} \frac{\partial net_j^l}{\partial w_{ij}^{l-1}}, \quad (28)$$

where a_j^l is an output of the j -th neuron of layer l , net_j^l is the incoming current to the neuron, and w_{ij}^{l-1} is the weight from i -th neuron of the preceding layer to the neuron. SBBP uses the following surrogate derivatives for $\partial a_j^l / \partial net_j^l$:

$$\frac{\partial a_j^l}{\partial net_j^l} = \begin{cases} \frac{1}{V_{th}}, & \text{for output layer,} \\ \frac{1}{V_{th}} \left(1 + \frac{1}{\gamma} \frac{\partial f(t)}{\partial t} \right), & \text{for hidden layer,} \end{cases} \quad (29)$$

where $f(t) = \sum_k \exp(-(t - t_k)/\tau_m)$ and τ_m is the time constant.

3.2.2 ANN-SNN Conversion

The ANN-SNN conversion methods first train an ANN model and then fine-tune an SNN model of the same architecture as the trained ANN model, of which weights are initialized with the weights of the ANN model. Table 2 summarizes the characteristics of some ANN-SNN conversion methods.

Hunsberger and Eliasmith's method [44] first trains an ANN which may contain convolutional layers and average pooling, and consists of its neurons with no bias terms. The ANN uses the so-called soft-LIF activation function instead of ReLU. The

Figure 13. Firing rate functions for soft-LIF (dotted curve) and LIF (solid curve) [44].

soft-LIF is a firing rate function of input current similar to that of LIF as shown in Figure 13. The soft-LIF firing rate function is differentiable while the LIF firing rate function is not. The soft-LIF function is defined as follows:

$$r(i) = 1 / \left(\tau_{ref} - \tau_{RC} \left(1 - \frac{V_{th}}{i} \right) \right), \quad (30)$$

Table 2. ANN-SNN conversion algorithms

Algorithm	Neuron model	Architecture	Input encoding	Output decoding	Features
soft-LIF (2015, [44])	soft-LIF (ANN) LIF (SNN)	Deep network	Spike train (rate code)	Spike train (firing rate)	Use soft-LIF in ANN for LIF
Cao et al. (2015, [45])	ReLU (ANN) IF (SNN)	Shallow network	Spike train (rate code)	Spike train (firing rate)	Constrained arch.; avg. pooling, no bias
Diehl et al. (2015, [46])	ReLU (ANN) IF (SNN)	Shallow network	Spike train (rate code)	Spike train (firing rate)	Constrained arch.; weight normalization
Rueckauer et al. (2017, [30])	ReLU (ANN) IF (SNN)	Deep network	Direct input	Spike train (firing rate)	Constrained arch.; batch norm.; softmax
Whetstone (2018, [47])	bReLU (ANN) IF (SNN)	Deep network	Spike train (rate code)	Spike train (firing rate)	Adaptive sharpening of activation function
Sengupta et al. (2019, [48])	ReLU (ANN) IF (SNN)	Deep network	Spike train (rate code)	Spike train (firing rate)	Normalization in SNN; Spike-Norm
RMP-SNN (2020, [49])	ReLU (ANN) IF (SNN)	Deep network	Spike train (rate code)	Spike train (firing rate)	IF with soft-reset; control threshold range; threshold balancing
Deng et al. (2021, [50])	thr. ReLU (ANN) IF (SNN)	Deep network	Spike train (rate code)	Spike train (firing rate)	Conversion loss-aware bias adaptation; threshold ReLU; shifted bias
Ding et al. (2021, [51])	RNL (ANN) IF (SNN)	Deep network	Spike train (rate code)	Spike train (rate code)	Optimal scaling factors for threshold balancing
Patel et al. (2021, [52])	mod. ReLU (ANN) IF (SNN)	Scaled-down U-Net	Spike train (rate code)	Spike train (rate code)	image segmentation Loihi deployment

where τ_{ref} and τ_{RC} are time constants for refractory period and resistor-capacitor component in Figure 3, respectively, and i is the input current. Once an ANN with the soft-LIF activation function is trained, the soft-LIF is replaced with LIF, and input and output are expressed in spikes, to get an SNN corresponding to the ANN.

Cao et al.'s method [45] first trains a tailored CNN model in which all layers produce positive values, all neurons at convolutional layers and fully connected layers have no biases, and average pooling is used instead of max pooling. For the tailed CNN model, an SNN is organized with the same architecture, which uses IF neurons with no bias terms, receives and generates spike trains, and uses average pooling, if any. Once the CNN model is trained, its weights are used to initialize the weights of the organized SNN.

Diehl et al.'s method [46] is an improvement of Cao et al.'s method [45] which first trains an ANN model, and normalizes its weights before deploying them into an SNN model of the same architecture. When the weights of a trained ANN model are directly used as those of its corresponding SNN model, the neurons of the SNN model may get insufficient membrane potential to reach the threshold level. In addition, some membrane potentials are too large to generate just a single spike. To handle these issues, the weight normalization techniques have been developed. Among them, there are the model-based normalization and the data-based normalization [46]. In the model-based method, weights are normalized in a layer-wise manner, which are divided by the maximum of the positive weights. In the data-based method, we choose as the scaling factors the maximum of the activation of training data for each neuron of the trained ANN model. Then we divide the weights to each neuron by its scaling factor or use the scaling factor as the threshold level of the corresponding neuron for the SNN model. Those weight normalization methods have shown that such a weight-normalized SNN model gives better performance than the baseline model with no weight normalization.

In [30], the authors proposed an algorithm to first train an ANN having some architectural restrictions, and then convert it into an SNN model of which neurons are IF neurons with bias terms. It has established a theoretic foundation for the relationship between LIF activation and firing rate of spiking neurons. It supports the following two reset modes for membrane potential at spike generation: reset-to-zero mode and reset-by-subtraction mode. The transformed SNN model allows its neurons to have bias terms, uses the input as an input current to the neurons of the first layer, uses max-pooling by using a gating function,

Figure 14. Sharpening of bReLU function at the Whetstone method.

and generates spikes according to the softmax probability at the output layer. When weights are transferred from the trained ANN model to the SNN, it uses a slightly-modified data-based weight normalization method.

Whetstone method [47] is a process to train binary, threshold-activation SNNs using the existing deep learning methods. It first trains an ANN until performance makes no improvement. Then, it progressively sharpens the activation function toward a step activation at each layer one at a time, beginning from the input layer, while managing performance. The sharpening process is automated with an adaptive sharpening schedule. As the activation function, it uses the bounded rectified linear unit (bReLU) $h_{\alpha,\beta}(x)$ defined as follows:

$$h_{\alpha,\beta}(x) = \begin{cases} 1, & \text{if } x \geq \beta, \\ (x - \alpha)/(\beta - \alpha), & \text{if } \alpha \leq x < \beta, \\ 0, & \text{if } x < \alpha. \end{cases} \quad (31)$$

As α approaches β , the function gets sharper. During the sharpening process, the input does not need to be encoded into spike or spike train. The training is conducted in the same way as in conventional ANN training. Figure 14 shows a process to sharpen the bReLU function in the Whetstone method.

Sengupta et al.'s method [48] is an ANN-SNN conversion method of which SNN models have IF neurons with no bias terms, and may include the average pooling and the identity skip connections for deeper networks. The method first trains an ANN, next initializes an SNN of the same architecture with the trained weights, then do the threshold balancing to adjust the threshold level of spiking neurons. For threshold balancing, it uses Spike-Norm [48] which can be regarded as an improvement of Diehl et al.'s normalization method [46]. Spike-Norm chooses as the scaling factor for each neuron the maximum membrane potential for the training data in the converted SNN. Then it uses the scaling factor as the threshold value of spiking neurons.

RMP-SNN method [49] is an ANN-SNN conversion method where an SNN model uses IF neurons with soft-reset [57]. Hard-reset indicates a mechanism to reset the membrane potential

of a spike neuron to a pre-specified low potential just after generating a spike when the membrane potential reaches the threshold level. On the other hand, soft-reset (a.k.a., reset by subtraction) is a mechanism that keeps the residual potential above the firing threshold just after generating a spike.

The trained ANN uses ReLU activation function, and its weights are transferred to an SNN of the same architecture. A neuron with ReLU function produces the output proportional to its weighted input sum, but a spiking neuron with hard-reset usually does not produce spikes of which rate is proportioned to its membrane potential. It is assumed that the conversion loss of ANN-SNN conversion is caused by the nonlinearity between membrane potential and the spiking rate in an IF neuron with the hard-reset. In the RMP-SNN method, an SNN consists of IF neurons with soft-reset, which receives spike trains and generates spike trains. To guarantee the linearity between membrane potential V_{in} and output firing rate f_{out} , the operating range of the threshold level V_{th} is maintained to hold the following condition:

$$f_{in} V_{in} \leq V_{th} \leq V_{in}, \quad (32)$$

where f_{in} and V_{in} are average input rate and voltage amplitude, respectively.

Deng and Gu [50] paid attention to the conversion loss from an ANN to its corresponding SNN in terms of activation function and reset operation after spike generation. To begin with, they adopt IF neurons with soft-reset [57] to reduce the loss. In addition, they observed that the shifted threshold ReLU of shift $V_{th}/2T$ has less differences from SNN than ReLU as shown in Figure 15, where V_{th} is the threshold level and T is the number of time steps, i.e., latency. In their method, they first train an ANN with the threshold ReLU activation function of shift 0, and then convert it into an SNN of which weights are initialized with the weights of the trained ANN, and the threshold level V_{th} of each layer is set to the maximum activation of the ANN at the corresponding layer. In addition, the biases of spike neurons in SNN are set to the corresponding biases of the trained ANN which are added by $V_{th}^l/2T$.

Ding et al. [51] introduced a weight normalization method called rate normalization. The method adjusts the threshold level θ_l of each layer with a trainable parameter p_l which is used to clip the activation value of the neurons and to scale the maximum activation as follows:

$$\theta_l = p_l \cdot \max(W_{l-1} \mathbf{r}_{l-1} + \mathbf{b}_{l-1}), \quad (33)$$

Figure 15. Activation functions of ReLU, threshold ReLU, and SNN.

$$\mathbf{z}_l = \text{clip}(W_{l-1} \mathbf{r}_{l-1} + \mathbf{b}_{l-1}, 0, \theta_l), \quad (34)$$

$$\mathbf{r}_l = \frac{\mathbf{z}_l}{\theta_l}, \quad (35)$$

where $\mathbf{r}_l$ is the firing rate and $\mathbf{z}_l$ is the clipped activation which is bounded by 0 and θ_l . They first train an ANN so as to minimize the difference between the ANN's output and the desired output for the training data. Then they train the scaling parameter p_l to minimize the difference between the ANN output and the SNN output.

Patel et al. [52] applied an ANN-SNN conversion method to develop an SNN-based U-Net [53] for 2D image segmentation. When they train a U-Net, they use a modified ReLU function as shown in Figure 16, which is defined as follows:

$$f(x) = \begin{cases} (p(x))^{-1}, & \text{if } x > 0, \\ 0, & \text{otherwise,} \end{cases} \quad p(x) = \Delta t \left\lceil \frac{1}{x \Delta t} \right\rceil, \quad (36)$$

where Δt indicates the simulation time step. In the training phase, they insert into input, noises sampled from a uniform distribution from zero to one. The derivative of the above modified ReLU is given as follows:

$$f'(x) = \begin{cases} \left(\frac{\Delta t}{2} + \frac{1}{2} \right)^{-1}, & \text{if } x > 0, \\ 0, & \text{otherwise.} \end{cases} \quad (37)$$

Once the U-Net is trained, it is converted into an multi-layered SNN. They use a percentile-based loss function which regularize the maximum firing rate of each neuron across all example in the batch to be between a minimum and a maximum value. Once an SNN is obtained, they apply a quantization method and a partitioning method to the SNN so as to deploy it onto the Loihi chip [60] which is a neuromorphic chip.

Figure 16. A modified ReLU function and its derivative.

3.2.3 Hybrid Methods

The hybrid training methods first initialize weights of an SNN model with those trained by the ANN-SNN conversion method, and then fine-tune the SNN model with a direct learning method. Table 3 summarizes the characteristics of some hybrid methods.

Rathi et al. method [54] first uses an ANN-SNN conversion method similar to Diehl et al.'s method [46], to get an SNN which uses LIF neurons with no biases, uses average pooling for pooling operation, receives spike trains generated by the Poisson rate coding method, has the output neurons with no leakage and no spike generation, and applies the softmax function to accumulated membrane potential of output neurons so as to get classification probabilities. For fine-tuning the SNN, it uses the cross-entropy as the loss function L . It applies the STDB [42]-like algorithm to fine-tune the SNN, where a surrogate gradient for $\partial o_i^t / \partial u_i^t$ is defined as follows:

$$\frac{\partial o_i^t}{\partial u_i^t} = \alpha \exp(-\beta \Delta t), \quad \Delta t = t - t_s, \quad (38)$$

where o_i^t indicates the occurrence of a spike at time t at neuron i , u_i^t is the membrane potential at time t at neuron i , and Δt indicate the difference between the current time step t and the last time step t_s in which the neuron generated a spike. The

surrogate gradient is used in computing the gradient of L with respect to parameters.

Direct input encoding with leakage and threshold optimization in deep spiking neural networks (DIET-SNN) algorithm [55] first trains an ANN, next converts the trained ANN to an SNN, and then fine-tunes the SNN using a surrogate gradient. For an ANN, it uses the ReLU activation function and no bias terms for neurons, does not apply the batch normalization method, and uses the average pooling, if needed. In the ANN-SNN conversion, the converted SNN consists of IF neurons, input is fed directly into the neurons of the SNN without any encoding, and a threshold balancing method is applied. In the fine-tuning phase, the SNN consists of LIF neurons, receives the input as the input current directly to the neurons of the input layer, and produces probabilities computed by applying the softmax function to the accumulated membrane potentials of the output layer. As the loss function, the cross-entropy is used. On computing the gradient of the loss function with respect to weights, the surrogate of $\partial o_i^t / \partial z_i^t$ is used, which is defined as follows:

$$\frac{\partial o_i^t}{\partial z_i^t} = \gamma \max\{0, 1 - |z_i^t|\}, \quad z_i^t = \frac{u_i^t}{V_i^{th}}, \quad (39)$$

where u_i is the membrane potential of neuron i , o_i^t is the occurrence of spike at time t at neuron i , and V_i^{th} is the threshold level for neuron i . It optimizes weights, leakage parameters, and thresholds all together as the algorithm name implies.

Takuka et al.' method [58] uses a knowledge distillation technique which effectively learns a small student model from a large trained teacher model. As shown in Figure 17, the method first trains a large ANN which generates the class probabilities, and next trains a small ANN from the trained large ANN using a knowledge distillation technique which uses the following

Table 3. Hybrid training algorithms

Algorithm	Neuron model	Architecture	Input encoding	Output decoding	Features
Rathi et al. (2020, [54])	ReLU (ANN) LIF (SNN)	Deep network	Spike train (rate coding)	Direct output (membrane potential)	ANN-SNN conv. + STDB; ST-based surrogate gradient
DIET-SNN (2020, [55])	ReLU (ANN) IF/LIF (SNN)	Deep network	Direct input	Direct output	Trainable leakage and threshold in LIF
Takuya et al. (2021, [58])	ReLU (ANN) LIF (SNN)	Deep network	Direct input	Direct output (membrane potential)	Knowledge distillation for conv. ; fine-tuning

Figure 17. Knowledge distillation-based SNN training [58].

loss function L_{kd} :

$$L_{kd} = - \sum_{\mathbf{x} \in D_x} \sum_{i=1}^C p_{AS}^i(\mathbf{x}) \log \frac{p_{AS}^i(\mathbf{x})}{p_{AL}^i(\mathbf{x})}, \quad (40)$$

where $p_{AS}^i(\mathbf{x})$ and $p_{AL}^i(\mathbf{x})$ indicate the probability of class i into which the small ANN and the large ANN classify the input $\mathbf{x}$, respectively. Then, the weights of the small ANN are transferred into those of an SNN. After that, the SNN is fine-tuned with reference to the large ANN and the small ANN using a knowledge distillation technique with the following loss function:

$$L = \lambda_1 L_{ce} + \lambda_2 L_{kd1} + \lambda_3 L_{kd2}, \quad (41)$$

where λ_1 , λ_2 , and λ_3 are the hyperparameters to control the contribution of the corresponding loss functions, and L_{ce} is the loss function of the SNN itself, L_{kd1} is the loss function of the knowledge distillation from the large ANN to the SNN, and L_{kd2} is the loss function of the knowledge distillation from the small ANN to the SNN. Those loss functions are defined as follows:

$$L_{ce} = - \sum_{\mathbf{x} \in D_x} \sum_{i=1}^C p_{SN}^i(\mathbf{x}; 1) \log(p_{SN}^i(\mathbf{x}; 1)), \quad (42)$$

$$L_{kd1} = -\tau^2 \sum_{\mathbf{x} \in D_x} \sum_{i=1}^C p_{SN}^i(\mathbf{x}) \log \frac{p_{SN}^i(\mathbf{x})}{p_{AL}^i(\mathbf{x})}, \quad (43)$$

$$L_{kd2} = -\tau^2 \sum_{\mathbf{x} \in D_x} \sum_{i=1}^C p_{SN}^i(\mathbf{x}) \log \frac{p_{SN}^i(\mathbf{x})}{p_{AS}^i(\mathbf{x})}. \quad (44)$$

The SNN has the same architecture with the small ANN. In

the SNN, each input neuron receives directly the input value as input current, and the output neurons do not generate spikes and produce the class probabilities by applying the softmax function to the membrane potentials.

4. Neuromorphic Hardware

Neuromorphic hardware are specialized hardware to simulate SNNs very fast and efficiently. Various neuromorphic hardware have been developed in processor cluster, FPGA, or chips. Some neuromorphic hardware support only some specific hard-wired SNN architectures, and others allow the architecture of SNNs to be configured.

Some neuromorphic hardware such as SpiNNaker, BrainScaleS, and Neurogrid have been developed for neuroscience simulations to study the brain [59]. SpiNNaker uses a network of ARM processors tightly connected to local memory as the building blocks which is housed in 10 racks, with each rack holding over 100,000 cores. It supports several spiking neuron model including LIF and Izhikevich model, and has some software tools for learning SNNs. BrainScaleS is constructed with several wafers interconnected together, each wafer consisting of 384 cores, 200K neurons, and 45M synapses. It is used to simulate brain-scale neural networks. Neurogrid consists of 16 Neurocores, each of which has 65,536 neurons, and allows to simulate one million neurons and six billion synapses in real time.

A few neuromorphic chips such as TrueNorth and Loihi have been developed, which targets low power large scale SNN evaluation. TrueNorth is a chip which consists of 4,096 cores with 1 million neurons and 256 million synapses. Its neuron is a modified LIF neuron. Loihi is a chip with a manycore mesh comprising 128 neuromorphic cores, 3 embedded x86 processor cores, and off-chip communication interfaces that hierarchically extend the mesh in 4 planar directions to other chips, up to 16,384 chip. It supports LIF neuron model which can be used as IF neuron when leakage is set to zero. Both TrueNorth and Loihi are not yet being sold for general research and development at the moment of writing [60].

Several SNN FPGA boards such as PYNQ-Z1 and DE1-SoC are on the market. Such boards support SNNs with a fixed number of layers like 1 to 4 layers, and limited types of neuron models. Some of them support on-line learning like PES learning or STDP learning. There are some analog-digital chips designed to support a fixed SNN architecture [61]. In neuromorphic chips, the SNN models are usually trained offline

and later the trained modes are downloaded into them. Due to complexity of circuits, online learning algorithms except STDP are usually not supported in the neuromorphic chips.

In the semiconductor sectors, there have been designed and experimentally fabricated the neuron models and synaptic connections using CMOS and Memristor Technologies [62]. They have just shown the possibility of energy-efficient neuromorphic chips. There are yet no widely accessible neuromorphic hardware on which SNNs are deployed and executed at the moment of writing.

5. Conclusion

Spiking neural networks have attracted attention for their energy efficient operation and biological plausibility. Neuroscientists are interested in simulating brain-scale SNNs to study brain functions. In machine learning, DNNs have been writing stories of success in formerly difficult tasks like vision, speech, and natural language processing. Machine learning people have worked on how to develop SNNs which are as good as DNNs. The performances of SNNs are approaching those of DNNs, but are not yet enough to replace DNNs.

The paper have addressed SNNs mainly in the perspective of machine learning. Various SNN learning algorithms have been developed and being developed. The direct learning algorithms yet seem to be difficult to train a deep SNN. The ANN-SNN conversion algorithms seem to be a best way to build deep SNNs. There will be more efforts to be exerted to reduce the conversion loss from an ANN to its corresponding SNN. The hybrid learning methods will take further advantage of both the ANN-SNN conversion algorithms and the direct training algorithms as each of them makes advances.

An ANN takes just a single cycle from the input layer to the output layer, but an SNN has to experience multiple cycles to get a stable output. Hence, one research direction in SNNs is to reduce the latency of SNN execution.

Most SNN machine learning studies have been conducted in software simulation. The accessibility to neuromorphic hardware is yet limited. Once some low-cost neuromorphic hardware are available, SNN models are expected to be widely deployed in edge devices of the IoT environments due to their energy efficiency.

Conflict of Interest

No potential conflict of interest relevant to this article was reported.

Acknowledgements

This research was partly supported by the MSIT (Ministry of Science and ICT), Korea, under the Grand Information Technology Research Center support program (IITP-2021-0-01462) supervised by the IITP (Institute for Information & communications Technology Planning & Evaluation), and partly by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-00708, IDE for Autonomic IoT Applications based on Neuromorphic Architecture) and Korea Evaluation Institute of Industrial Technology (KEIT) grant funded by the Korea government (MOTIE) (No. 2019-0-00708, IDE for Autonomic IoT Applications based on Neuromorphic Architecture).

References

- [1] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M. L. Shyu, S. C. Chen, and S. S. Iyengar, "A survey on deep learning: algorithms, techniques, and applications," *ACM Computing Surveys*, vol. 51, no. 5, pp. 1-36, 2019. <https://doi.org/10.1145/3234150>
- [2] N. Chauhan and B. J. Choi, "DNN based classification of ADHD fMRI data using functional connectivity coefficient," *International Journal of Fuzzy Logic and Intelligent Systems*, vol. 20, no. 4, pp. 255-260, 2020. <https://doi.org/10.5391/IJFIS.2020.20.4.255>
- [3] U. Erdenebayar, Y. Kim, J. U. Park, S. Lee, and K. J. Lee, "Automatic classification of sleep stage from an ECG signal using a gated-recurrent unit," *International Journal of Fuzzy Logic and Intelligent Systems*, vol. 20, no. 3, pp. 181-187, 2020. <https://doi.org/10.5391/IJFIS.2020.20.3.181>
- [4] K. I. Kim and K. M. Lee, "Convolutional neural network-based gear type identification from automatic identification system trajectory data," *Applied Sciences*, vol. 10, no. 11, article no. 4010, 2020. <https://doi.org/10.3390/app10114010>

- [5] K. M. Lee, K. S. Park, K. S. Hwang, and K. I. Kim, "Deep neural network model construction with interactive code reuse and automatic code transformation," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 18, article no. e5480, 2020. <https://doi.org/10.1002/cpe.5480>
- [6] H. Jang, O. Simeone, B. Gardner, and A. Gruning, "An introduction to probabilistic spiking neural networks: probabilistic models, learning rules, and applications," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 64-77, 2019. <https://doi.org/10.1109/MSP.2019.2935234>
- [7] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: opportunities and challenges," *Frontiers in Neuroscience*, vol. 12, article no. 774, 2018. <https://doi.org/10.3389/fnins.2018.00774>
- [8] S. Herculano-Houzel, "The human brain in numbers: a linearly scaled-up primate brain," *Frontiers in Human Neuroscience*, vol. 3, article no. 31, 2009. <https://doi.org/10.3389/neuro.09.031.2009>
- [9] S. M. Schuetze, "The discovery of the action potential," *Trends in Neurosciences*, vol. 6, pp. 164-168, 1983. [https://doi.org/10.1016/0166-2236\(83\)90078-4](https://doi.org/10.1016/0166-2236(83)90078-4)
- [10] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge, UK: Cambridge University Press, 2002.
- [11] R. Jahn and T. C. Sudhof, "Synaptic vesicles and exocytosis," *Annual Review of Neuroscience*, vol. 17, no. 1, pp. 219-246, 1994. <https://doi.org/10.1146/annurev.ne.17.030194.001251>
- [12] T. C. Sudhof, "Neurotransmitter release," in *Pharmacology of Neurotransmitter Release*. Heidelberg, Germany: Springer, 2008, pp. 1-21. https://doi.org/10.1007/978-3-540-74805-2_1
- [13] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge, UK: Cambridge University Press, 2014.
- [14] A. L. Hodgkin, "A quantitative description of ion currents and its application and excitation in nerve membranes," *Journal of Physiology*, vol. 117, pp. 500-544, 1952. <https://doi.org/10.1113/jphysiol.1952.sp004764>
- [15] R. B. Stein, "Some models of neuronal variability," *Biophysical Journal*, vol. 7, no. 1, pp. 37-68, 1967. [https://doi.org/10.1016/S0006-3495\(67\)86574-3](https://doi.org/10.1016/S0006-3495(67)86574-3)
- [16] W. Gerstner, "Time structure of the activity in neural network models," *Physical Review E*, vol. 51, no. 1, pp. 738-758, 1995. <https://doi.org/10.1103/PhysRevE.51.738>
- [17] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569-1572, 2003. <https://doi.org/10.1109/TNN.2003.820440>
- [18] E. M. Izhikevich, "FitzHugh-Nagumo model," *Scholarpedia*, vol. 1, no. 9, article no. 1349, 2006. <https://doi.org/10.4249/scholarpedia.1349>
- [19] J. Lisman and N. Spruston, "Questions about STDP as a general model of synaptic plasticity," *Frontiers in Synaptic Neuroscience*, vol. 2, article no. 140, 2010. <https://doi.org/10.3389/fnsyn.2010.00140>
- [20] R. C. Malenka and M. F. Bear, "LTP and LTD: an embarrassment of riches," *Neuron*, vol. 44, no. 1, pp. 5-21, 2004. <https://doi.org/10.1016/j.neuron.2004.09.012>
- [21] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659-1671, 1997. [https://doi.org/10.1016/S0893-6080\(97\)00011-7](https://doi.org/10.1016/S0893-6080(97)00011-7)
- [22] C. L. Tao, Y. T. Liu, R. Sun, B. Zhang, L. Qi, S. Shivakoti, et al., "Differentiation and characterization of excitatory and inhibitory synapses by cryo-electron tomography and correlative microscopy," *Journal of Neuroscience*, vol. 38, no. 6, pp. 1493-1510, 2018. <https://doi.org/10.1523/JNEUROSCI.1548-17.2017>
- [23] F. Ponulak and A. Kasinski, "Introduction to spiking neural networks: information processing, learning and applications," *Acta Neurobiologiae Experimentalis*, vol. 71, no. 4, pp. 409-433, 2011.
- [24] Y. Ikegaya, G. Aaron, R. Cossart, D. Aronov, I. Lampl, D. Ferster, and R. Yuste, "Synfire chains and cortical songs: temporal modules of cortical activity," *Science*, vol. 304, no. 5670, pp. 559-564, 2004. <https://doi.org/10.1126/science.1093173>
- [25] W. Maass, T. Natschlager, and H. Markram, "Real-time computing without stable states: a new framework for

- neural computation based on perturbations,” *Neural Computation*, vol. 14, no. 11, pp. 2531-2560, 2002. <https://doi.org/10.1162/089976602760407955>
- [26] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen, “A large-scale model of the functioning brain,” *Science*, vol. 338, no. 6111, pp. 1202-1205, 2012. <https://doi.org/10.1126/science.1225266>
- [27] F. Rieke, D. Warland, R. D. R. Van Steveninck, and W. Bialek, *Spikes: Exploring the Neural Code*. Cambridge, MA: MIT Press, 1999.
- [28] D. Heeger, “Poisson model of spike generation,” 2000 [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.6580>.
- [29] S. Wu, S. I. Amari, and H. Nakahara, “Population coding and decoding in a neural field: a computational study,” *Neural Computation*, vol. 14, no. 5, pp. 999-1026, 2002. <https://doi.org/10.1162/089976602753633367>
- [30] B. Rueckauer, I. A. Lungu, Y. Hu, M. Pfeiffer, and S. C. Liu, “Conversion of continuous-valued deep networks to efficient event-driven networks for image classification,” *Frontiers in Neuroscience*, vol. 11, article no. 682, 2017. <https://doi.org/10.3389/fnins.2017.00682>
- [31] C. Lee, S. S. Sarwar, P. Panda, G. Srinivasan, and K. Roy, “Enabling spike-based backpropagation for training deep neural network architectures,” *Frontiers in Neuroscience*, vol. 14, article no. 119, 2020. <https://doi.org/10.3389/fnins.2020.00119>
- [32] A. Tavanaei and A. Maida, “BP-STDP: approximating backpropagation using spike timing dependent plasticity,” *Neurocomputing*, vol. 330, pp. 39-47, 2009. <https://doi.org/10.1016/j.neucom.2018.11.014>
- [33] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. M. Bower, et al., “Simulation of networks of spiking neurons: a review of tools and strategies,” *Journal of Computational Neuroscience*, vol. 23, no. 3, pp. 349-398, 2007. <https://doi.org/10.1007/s10827-007-0038-6>
- [34] P. U. Diehl and M. Cook, “Unsupervised learning of digit recognition using spike-timing-dependent plasticity,” *Frontiers in Computational Neuroscience*, vol. 9, article no. 99, 2015. <https://doi.org/10.3389/fncom.2015.00099>
- [35] M. R. Azghadi, S. Al-Sarawi, N. Iannella, and D. Abbott, “Design and implementation of BCM rule based on spike-timing dependent plasticity,” in *Proceedings of the 2012 International Joint Conference on Neural Networks (IJCNN)*, Brisbane, Australia, 2012, pp. 1-7. <https://doi.org/10.1109/IJCNN.2012.6252778>
- [36] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, “STDP-based spiking deep convolutional neural networks for object recognition,” *Neural Networks*, vol. 99, pp. 56-67, 2018. <https://doi.org/10.1016/j.neunet.2017.12.005>
- [37] S. M. Bohte, J. N. Kok, and J. H. La Poutre, “Spike-prop: backpropagation for networks of spiking neurons,” in *Proceedings of the 8th European Symposium on Artificial Neural Networks*, Bruges, Belgium, 2000, pp. 17-37.
- [38] F. Ponulak, “ReSuMe: new supervised learning method for spiking neural networks,” Institute of Control and Information Engineering, Poznan University of Technology, Poznan, Poland, 2005.
- [39] C. Eliasmith and C. H. Anderson, *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. Cambridge, MA: MIT Press, 2003.
- [40] D. MacNeil and C. Eliasmith, “Fine-tuning and the stability of recurrent neural networks,” *PloS One*, vol. 6, no. 9, article no. e22885, 2011. <https://doi.org/10.1371/journal.pone.0022885>
- [41] T. Bekolay, C. Kolbeck, and C. Eliasmith, “Simultaneous unsupervised and supervised learning of cognitive functions in biologically plausible spiking neural networks,” in *Proceedings of the Annual Meeting of the Cognitive Science Society*, Berlin, Germany, 2013.
- [42] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, “Spatio-temporal backpropagation for training high-performance spiking neural networks,” *Frontiers in Neuroscience*, vol. 12, article no. 331, 2018. <https://doi.org/10.3389/fnins.2018.00331>
- [43] C. Lee, S. S. Sarwar, P. Panda, G. Srinivasan, and K. Roy, “Enabling spike-based backpropagation for training deep neural network architectures,” 2019 [Online]. Available: <https://arxiv.org/abs/1903.06379>.

- [44] E. Hunsberger and C. Eliasmith, "Spiking deep networks with LIF neurons," 2015 [Online]. Available: <https://arxiv.org/abs/1510.08829>.
- [45] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54-66, 2015. <https://doi.org/10.1007/s11263-014-0788-3>
- [46] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *Proceedings of 2015 International Joint Conference on Neural Networks (IJCNN)*, Killarney, Ireland, 2015, pp. 1-8. <https://doi.org/10.1109/IJCNN.2015.7280696>
- [47] W. Severa, C. M. Vineyard, R. Dellana, S. J. Verzi, and J. B. Aimone, "Whetstone: a method for training deep artificial neural networks for binary communication," 2018 [Online]. Available: <https://arxiv.org/abs/1810.11521>.
- [48] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: VGG and residual architectures," *Frontiers in Neuroscience*, vol. 13, article no. 95, 2019. <https://doi.org/10.3389/fnins.2019.00095>
- [49] B. Han, G. Srinivasan, and K. Roy, "RMP-SNN: residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Seattle, WA, 2020, pp. 13558-13567.
- [50] S. Deng and S. Gu, "Optimal conversion of conventional artificial neural networks to spiking neural networks," 2021 [Online]. Available: <https://arxiv.org/abs/2103.00476>.
- [51] J. Ding, Z. Yu, Y. Tian, and T. Huang, "Optimal ANN-SNN conversion for fast and accurate inference in deep spiking neural networks," 2021 [Online]. Available: <https://arxiv.org/abs/2105.11654>.
- [52] K. Patel, E. Hunsberger, S. Batir, and C. Eliasmith, "A spiking neural network for image segmentation," 2021 [Online]. Available: <https://arxiv.org/abs/2106.08921>.
- [53] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention*. Cham, Switzerland: Springer, 2015, pp. 234-241. https://doi.org/10.1007/978-3-319-24574-4_28
- [54] N. Rathi and K. Roy, "DIET-SNN: direct input encoding with leakage and threshold optimization in deep spiking neural networks," 2020 [Online]. Available: <https://arxiv.org/abs/2008.03658>.
- [55] N. Rathi, G. Srinivasan, P. Panda, and K. Roy, "Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation," 2020 [Online]. Available: <https://arxiv.org/abs/2005.01807>.
- [56] L. Deng, Y. Wu, Y. Hu, L. Liang, G. Li, X. Hu, Y. Ding, P. Li, and Y. Xie, "Comprehensive SNN compression using ADMM optimization and activity regularization," *IEEE Transactions on Neural Networks and Learning Systems*, 2021. <https://doi.org/10.1109/TNNLS.2021.3109064>
- [57] B. Rueckauer, I. A. Lungu, Y. Hu, and M. Pfeiffer, "Theory and tools for the conversion of analog to spiking convolutional neural networks," 2016 [Online]. Available: <https://arxiv.org/abs/1612.04052>.
- [58] S. Takuya, R. Zhang, and Y. Nakashima, "Training low-latency spiking neural network through knowledge distillation," in *Proceedings of 2021 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, Tokyo, Japan, 2021, pp. 1-3. <https://doi.org/10.1109/COOLCHIPS52128.2021.9410323>
- [59] M. Bouvier, A. Valentian, T. Mesquida, F. Rummens, M. Reyboz, E. Vianello, and E. Beigne, "Spiking neural networks hardware implementations and challenges: a survey," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 15, no. 2, article no. 22, 2019. <https://doi.org/10.1145/3304103>
- [60] M. Davies, N. Srinivasa, T. H. Lin, G. Chinya, Y. Cao, C. H. Choday, et al., "Loihi: a neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82-99, 2018. <https://doi.org/10.1109/MM.2018.112130359>
- [61] M. S. Asghar, S. Arslan, and H. Kim, "A low-power spiking neural network chip based on a compact LIF neuron and binary exponential charge injector synapse circuits," *Sensors*, vol. 21, no. 13, article no. 4462, 2021. <https://doi.org/10.3390/s21134462>

- [62] J. K. Han, J. Oh, G. J. Yun, D. Yoo, M. S. Kim, J. M. Yu, S. Y. Choi, and Y. K. Choi, “Co-integration of single transistor neurons and synapses by nanoscale CMOS fabrication for highly scalable neuromorphic hardware,” *Science Advances*, vol. 7, 2021. <https://10.1126/sciadv.abg8836>

Chan Sik Han is a Ph.D. candidate at Department of Computer Science, Chungbuk National University, Korea. He received his bachelor's and master's degrees at the same department. He has been working on research related to machine learning,

deep learning, and spiking neural networks.

E-mail: chatterboy@cbnu.ac.kr

Keon Myung Lee is a professor in the Department of Computer Science, Chungbuk National University, Korea. He received his B.S., M.S., and Ph.D. degrees in computer science from KAIST, Korea and was a postdoctorate fellow at INSA de Lyon, France. He was a visiting professor at the University of Colorado at Denver and a visiting scholar at Indiana University, USA. His principal research interests are machine learning, deep learning, soft computing, data science, and intelligent service systems.

E-mail: kmlee@cbnu.ac.kr