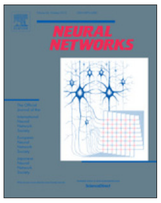
Neural Networks 125 (2020) 258–280

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet



2020 Special Issue

Supervised learning in spiking neural networks: A review of algorithms and evaluations

Xiangwen Wang, Xianghong Lin [*], Xiaochao Dang

College of Computer Science and Engineering, Northwest Normal University, Lanzhou, 730070, People's Republic of China

A R T I C L E I N F O

Article history:
Available online 25 February 2020

Keywords:
Spiking neural network
Spike train
Spiking neuron
Supervised learning
Performance evaluation

A B S T R A C T

As a new brain-inspired computational model of the artificial neural network, a spiking neural network encodes and processes neural information through precisely timed spike trains. Spiking neural networks are composed of biologically plausible spiking neurons, which have become suitable tools for processing complex temporal or spatiotemporal information. However, because of their intricately discontinuous and implicit nonlinear mechanisms, the formulation of efficient supervised learning algorithms for spiking neural networks is difficult, and has become an important problem in this research field. This article presents a comprehensive review of supervised learning algorithms for spiking neural networks and evaluates them qualitatively and quantitatively. First, a comparison between spiking neural networks and traditional artificial neural networks is provided. The general framework and some related theories of supervised learning for spiking neural networks are then introduced. Furthermore, the state-of-the-art supervised learning algorithms in recent years are reviewed from the perspectives of applicability to spiking neural network architecture and the inherent mechanisms of supervised learning algorithms. A performance comparison of spike train learning of some representative algorithms is also made. In addition, we provide five qualitative performance evaluation criteria for supervised learning algorithms for spiking neural networks and further present a new taxonomy for supervised learning algorithms depending on these five performance evaluation criteria. Finally, some future research directions in this research field are outlined.

1. Introduction

Artificial neural networks (ANNs) are abstractions and simulations of the structure and function of the biological nervous system (Prieto et al., 2016). Traditional ANNs encode neural information by the spike firing rate of the biological neurons (Haykin, 2009), in which the inputs and outputs of neurons are generally expressed as analog variables. However, experimental evidence from neuroscience suggests that biological nervous systems encode information through the precise timing of spikes, not only through the neuronal firing rate (Bohte, 2004; Borst & Theunissen, 1999; Walter, Röhrbein, & Knoll, 2016; Whalley, 2013). Using a biologically plausible spiking neuron model (Gerstner & Kistler, 2002; Izhikevich, 2004; Lin & Zhang, 2009) as the basic unit for constructing spiking neural networks (SNNs), they encode and process neural information through precisely timed spike trains. SNNs are often referred to as the third generation of neural networks (Ghosh-Dastidar & Adeli, 2009; Maass, 1997a). They have a greater computing capacity to simulate a variety of neuronal signals and approximate any continuous function (Maass, 1996, 1997b), and have been shown to be suitable tools for processing spatiotemporal information (Beyeler, Dutt, & Krichmar, 2013; Gütig, 2014; Kulkarni & Rajendran, 2018).

Supervised learning in ANNs involves a mechanism of providing the desired outputs with the corresponding inputs (Almási, Woźniak, Cristea, Leblebici, & Engbersen, 2016; Denéve, Alemi, & Bourdoukan, 2017). Experimental studies have shown that supervised learning exists in the biological nervous system (Glasera, Benjamina, Farhoodia, & Kordinga, 2019; Knudsen, 1994), but there is no clear conclusion to explain how biological neurons realize this process. For SNNs, the internal state variable of a neuron and the error function between the actual and desired output spike trains no longer satisfy the property of continuous differentiability. Consequently, the supervised learning algorithms for the traditional ANNs, such as the error back-propagation algorithm (Chauvin & Rumelhart, 1995; Rummelhart, 1986), cannot be used directly for SNNs.

Supervised learning for SNNs is a significant research field. Researchers have conducted many studies on supervised learning for SNNs and achieved some results (Kasiński & Ponulak, 2006; Lin, Wang, Zhang, & Ma, 2015). The supervised learning algorithms for SNNs proposed in recent years can be divided

* Corresponding author.
E-mail address: linxh@nwnu.edu.cn (X. Lin).

https://doi.org/10.1016/j.neunet.2020.02.011
0893-6080/© 2020 Elsevier Ltd. All rights reserved.

- supervised learning algorithms for SNNs
  - network architecture
    - single layer
    - multilayer feed-forward
    - recurrent
  - running mode
    - online learning
    - offline learning
  - information encoding
    - single-spike learning
    - spike train learning
  - structural dynamics
    - learning in fixed SNN structures
    - learning in evolving SNN structures
  - knowledge representation
    - non-knowledge-based learning
    - learning for knowledge representation

Fig. 1. Categories of supervised learning algorithms for spiking neural networks (SNNs).

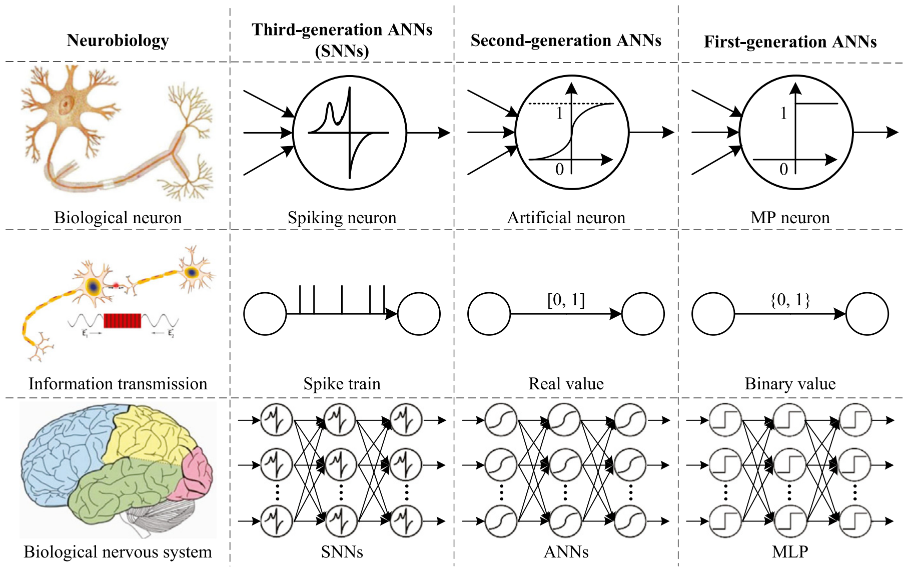


Fig. 2. Three generations of artificial neural networks (ANNs). MLP, multilayer perceptron; MP, McCulloch–Pitts; SNNs, spiking neural networks.

into several categories from different perspectives, as shown in Fig. 1. From the perspective of applicability to network architecture, they can be divided into supervised learning algorithms for single-layer SNNs, multilayer feed-forward SNNs, and recurrent SNNs. From the perspective of the running mode, they can be divided into online learning algorithms and offline learning algorithms (or batch learning algorithms) (Lobo, Del Ser, Bifet, & Kasabov, 2020). From the perspective of information encoding, they can be divided into producing a single spike and producing a spike train as outputs in response to input temporal or spatiotemporal data. From the perspective of structural dynamics, they can be divided into learning in fixed SNN structures and learning in evolving SNN structures (Kasabov, 2014, 2018; Kasabov, Dhoble, Nuntalid, & Indiveri, 2013). From the perspective of knowledge representation, they can be divided into non-knowledge-based learning and learning for knowledge representation (Kasabov, 2018).

This article provides a comprehensive review of supervised learning algorithms for SNNs from the perspectives of the applicability to SNN architectures and the inherent mechanisms of supervised learning algorithms. It further gives a performance evaluation and taxonomy for supervised learning algorithms, and finally it outlines future directions. The rest of this article is organized as follows. In Section 2, a comparison between SNNs and traditional ANNs is provided. In Section 3, some basic theories of supervised learning for SNNs are introduced, including the general framework and some related theories. In Sections 4–6, the state-of-the-art supervised learning algorithms for different SNN architectures are reviewed. A performance comparison of spike train learning of some representative algorithms is done in the corresponding sections. Section 7 presents the qualitative performance evaluation and taxonomy of supervised learning algorithms for SNNs. Section 8 outlines some future research directions in this research field. The conclusions for this article are presented in Section 9.

2. Comparison between SNNs and traditional ANNs

ANNs are abstractions and simulations of the structure and function of the biological nervous system, and play important roles in information processing and pattern recognition. An ANN is a computational model consisting of neurons as basic computational units. Information exchange between neurons is accomplished through synapses. According to their computational units, ANN models can be divided into three different generations (Maass, 1997a), as shown in Fig. 2.

First-generation ANNs use McCulloch–Pitts neurons (McCulloch & Pitts, 1943) as computational units, whose characteristic feature is that the output value is a binary variable. A McCulloch–Pitts neuron is the simplest neuron model, and can be described

Table 1
Comparison between spiking neural networks (SNNs) and traditional artificial neural networks (ANNs).

Characteristics		SNNs	Traditional ANNs
Information encoding and representation	Encoding scheme	Temporal encoding	Rate encoding
	Information representation	Spike trains	Scalars
	Dealing with temporal or spatiotemporal data	Excellent	Moderate
Computational unit and network simulation	Neuron model	Spiking neuron	Artificial neuron
	Computation mode	Differential equations	Activation function
	Network simulation	Clock-driven and event-driven	Step-by-step
Synaptic plasticity and learning	Plasticity mechanism	STDP rule	Hebb rule
	Designing supervised learning algorithms	Various mentalities	Derivative of loss function
Parallel and hardware implementation	Parallel computation	Massive	Moderate
	Hardware support	Neuromorphic VLSI	VLSI

STDP, spike-timing-dependent plasticity.

as

$$y=f\left(\sum_{i=1}^{N_I} w_i x_i-\theta\right)=\begin{cases}1 & \text{if } \sum_{i=1}^{N_I} w_i x_i-\theta\geq 0,\\ 0 & \text{if } \sum_{i=1}^{N_I} w_i x_i-\theta<0,\end{cases} \tag{1}$$

where $y\in\{0,1\}$ is the output of the McCulloch–Pitts neuron, $f(\cdot)$ is the activation function or state transition function, N_I is the number of input neurons, $x_i\in\mathbb{R}$ is the input of neuron i, $w_i\in\mathbb{R}$ is the synaptic weight between the input neuron i and the output neuron, and $\theta\in\mathbb{R}$ is the activation threshold. The representative computational model of first-generation ANNs is the perceptron (Minsky & Papert, 1969; Rosenblatt, 1958).

Second-generation ANNs use artificial neurons as computational units that apply a continuous function as the activation function of neurons to realize the processing of real numerical input and output. The artificial neuron can be described as

$$y=f\left(\sum_{i=1}^{N_I} w_i x_i+b\right), \tag{2}$$

where $b\in\mathbb{R}$ is the bias. This mode of information processing is a rate encoding scheme. The representative computational model is the feed-forward back-propagation neural network (Chauvin & Rumelhart, 1995; Rummelhart, 1986). Remarkable achievements in many fields have been made with second-generation ANNs. Sometimes traditional ANNs are referred to as *second-generation ANNs*. However, the computational power of the artificial neuron still does not reach its full potential because the temporal information on individual spikes is not represented.

Third-generation ANNs are SNNs that use biologically plausible spiking neurons as the basic computational units. The spiking neuron can be described as a hybrid system formalism (Brette, Rudolph, Carnevale, et al., 2007):

$$\begin{cases}\frac{d\boldsymbol{X}}{dt}=f(\boldsymbol{X}),\\ \boldsymbol{X}\leftarrow g_i(\boldsymbol{X}),\end{cases} \tag{3}$$

where $\boldsymbol{X}$ is a vector consisting of the state variables of the neuron, $f(\cdot)$ represents the differential equations for the evolution of the state variables, and $g_i(\cdot)$ represents the change of the state variables caused by spike events from synapse i. Neural information in the spiking neuron is transmitted and processed by the precisely timed spike trains. This is a temporal encoding scheme. Compared with the first- and second-generation ANN models, SNNs can describe the real biological nervous system more accurately, so as to achieve efficient information processing.

Table 1 provides a brief comparison between SNNs and traditional ANNs from the following four aspects (Kasabov, 2018):

1. Information encoding and representation. SNNs use the temporal encoding scheme to encode the information into spike trains, while ANNs use the rate encoding scheme to encode the information into scalars. This is one of the fundamental differences between SNNs and ANNs. Therefore, SNNs have more powerful information representation ability than ANNs, especially for complex temporal or spatiotemporal data.
2. Computational unit and network simulation. The basic computational unit for an SNN is the spiking neuron, which is expressed by differential equations, whereas the basic computational unit for an ANN is the artificial neuron, in which the inputs are processed by an activation function. This is another fundamental difference between SNNs and ANNs. The corresponding simulation strategies for SNNs are mainly clock-driven and event-driven, while the simulation strategy for ANNs is a step-by-step simulation process.
3. Synaptic plasticity and learning. The synaptic plasticity mechanism of SNNs emphasizes the spike-timing-dependent plasticity (STDP) between presynaptic and postsynaptic neurons, while the mechanism of ANNs generally satisfies the Hebb rule. When supervised learning algorithms are designed, SNNs generally have various mentalities (see Sections 4–6), while ANNs are mainly based on finding the derivative of the loss function.
4. Parallel and hardware implementation. SNNs can realize fast and massively parallel information processing, while ANNs are relatively weak. SNNs use the discrete spike train instead of analog signals to transmit information, which is more suitable for hardware implementation with lower energy consumption (Farsa, Ahmadi, Maleki, Gholami, & Rad, 2019).

3. Basic theories of supervised learning for SNNs

3.1. The general framework of supervised learning

Supervised learning for SNNs is implemented through learning the spatiotemporal patterns of spike trains. A spike train $s=\{t^f\in\Gamma:f=1,\ldots,F\}$ is the ordered sequence of spike times at which a spiking neuron fires in the time interval $\Gamma=[0,T]$, and can be expressed formally as

$$s(t)=\sum_{f=1}^{F}\delta\left(t-t^f\right), \tag{4}$$

where t^f is the fth spike time in $s(t)$, F is the number of spikes, and $\delta(\cdot)$ represents the Dirac delta function: $\delta(x)=1$ if $x=0$ and $\delta(x)=0$ otherwise.

Although the various supervised learning algorithms for SNNs are different, the overall goal of them is consistent: for a given set of input spike trains $\boldsymbol{S}_i$ and desired output spike trains $\boldsymbol{S}_d$,

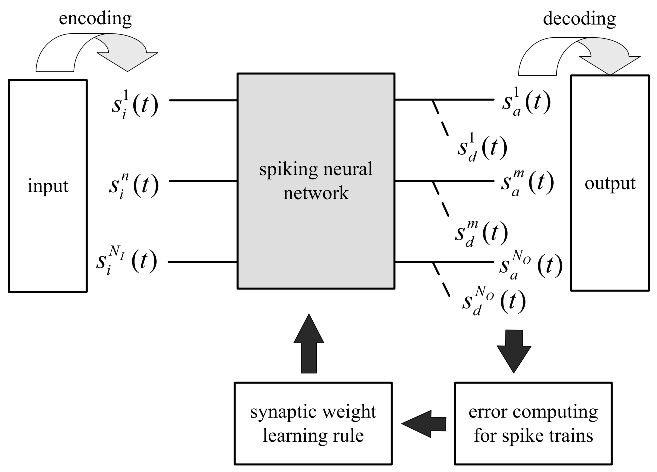


Fig. 3. The general framework of supervised learning for spiking neural networks.

finding the appropriate synaptic weight matrix $\boldsymbol{W}$ for SNNs to make the actual output spike trains $\boldsymbol{S}_a$ as similar as possible to the corresponding desired output spike trains $\boldsymbol{S}_d$; that is, the value of the error evaluation function $E(\boldsymbol{S}_a, \boldsymbol{S}_d)$ between them is a minimum. In the supervised learning process, the output spike trains $\boldsymbol{S}_a$ are determined by the input spike trains $\boldsymbol{S}_i$ and the synaptic weight matrix $\boldsymbol{W}$ connected to them, which can be expressed formally as

$$\boldsymbol{S}_a = F(\boldsymbol{S}_i, \boldsymbol{W}), \tag{5}$$

where F is a functional relationship between $\boldsymbol{S}_i$ and $\boldsymbol{W}$. Therefore, supervised learning for SNNs can be defined as

$$\min\ E(\boldsymbol{S}_a, \boldsymbol{S}_d) = \min \sum_{m=1}^{N_O} \left| s_d^m(t) - s_a^m(t) \right|, \tag{6}$$

where N_O is the number of output neurons, and $\left| s_d^m(t) - s_a^m(t) \right|$ is the similarity distance between $s_d^m(t)$ and $s_a^m(t)$. That means supervised learning is a process of minimizing the total error of the network, where the similarity measurement between spike trains can be used to calculate the error.

Assuming that an SNN contains N_I input neurons and N_O output neurons, the general framework of supervised learning for the SNN is shown in Fig. 3. Starting with the initial synaptic weight matrix $\boldsymbol{W}$ generated randomly, each supervised learning process for the SNN can be divided into the following four stages:

1. Encoding the input data into spike trains $s_i^n(t) \in \boldsymbol{S}_i$ through some specific coding methods.
2. Inputting the spike trains $s_i^n(t)$ into the SNN, and then using a certain simulation strategy to run the network. Finally, the actual output spike trains $s_a^m(t) \in \boldsymbol{S}_a$ can be obtained.
3. According to the desired output spike trains $s_d^m(t) \in \boldsymbol{S}_d$, computing the value of the error function $E(\boldsymbol{S}_a, \boldsymbol{S}_d)$ and adjusting the synaptic weights: $\boldsymbol{W} \leftarrow \boldsymbol{W} + \Delta\boldsymbol{W}$.
4. Determining whether the error of the SNN obtained by the learning process has reached a predetermined minimum error or the upper limit of learning epochs has been exceeded. If the termination condition is not satisfied, the learning process is repeated. After supervised learning, the output spike trains $\boldsymbol{S}_a$ of the network are decoded by a specific decoding method.

3.2. Related theories of supervised learning

The key to a supervised learning algorithm for SNNs is to construct the proper learning rule of synaptic weights. Meanwhile, it is also influenced by other related theories (Belatreche, Maguire, & McGinnity, 2007; Dorogyy & Kolisnichenko, 2016). These methods and techniques are discussed next.

3.2.1. Neural information encoding and decoding

Since the input and output of SNNs are spike trains, the analog quantities cannot be calculated directly. The first consideration is the encoding and decoding mechanism for neural information (Petro, Kasabov, & Kiss, 2020). The encoding of input and output spike trains has a great influence on the performance of supervised learning algorithms for SNNs (Taherkhani, Cosma, & McGinnity, 2019; Xu, Yang, & Zeng, 2019). *Encoding* refers to converting sample data or a stimulus signal into spike trains, while decoding is the reverse process of encoding (Brockmeier, Choi, Kriminger, Francis, & Principe, 2014), mapping the spike trains to output or a particular response. By referencing the encoding mechanism of biological neurons to specific stimulus signals, researchers have provided many temporal encoding strategies (Quiroga & Panzeri, 2013), such as the time-to-first-spike method (Tuckwell & Wan, 2005), the latency-phase method (Nadasdy, 2009), the population encoding method (Georgopoulos, Schwartz, & Kettner, 1986), and Ben's spiker algorithm (Schrauwen & Van Campenhout, 2003).

3.2.2. Computational models of spiking neurons

Spiking neurons are the basic computational units of SNNs. To simulate the function of the nervous system and solve practical problems using SNNs, it is of great importance to model suitable spiking neuron models. Depending on the complexity, the computational model of the spiking neuron can be divided into three categories (Lin & Gong, 2011): physiological models with biological plausibility, nonlinear models with a spiking mechanism, and linear models with a fixed threshold. Some spiking neuron models, such as the Hodgkin–Huxley model (Hodgkin & Huxley, 1952a, 1952b), are too complex, so it is difficult to realize large-scale SNN simulation with them, whereas some simple spiking neuron models, such as the integrate-and-fire neuron model (Burkitt, 2006a, 2006b) and the spike response model (SRM) (Gerstner, 2001; Gerstner & Kistler, 2002) with analytic expressions are commonly used in SNN simulation, especially in supervised learning.

3.2.3. Simulation strategy for SNNs

Since the spiking neuron is represented as a hybrid system through the continuous system and discrete spike events, simulation of SNNs is different from that of the traditional ANNs. Generally, two simulation strategies are used for SNNs: a clock-driven simulation strategy (Henker, Partzsch, & Schüffny, 2012) and an event-driven simulation strategy (Mattia & Del Giudice, 2000; Ros, Carrillo, Ortigosa, Barbour, & Agís, 2006). Some hybrid simulation strategies have also been proposed in recent years (D'Haene, Hermans, & Schrauwen, 2014; Kaabi, Tonnelier, & Martinez, 2011; Peng, Wang, Han, Song, & Ding, 2018; Zheng, Tonnelier, & Martinez, 2009). Research results show that different simulation strategies will affect the dynamic characteristics and learning performance of SNNs (Brette et al., 2007; Schæfer et al., 2002).

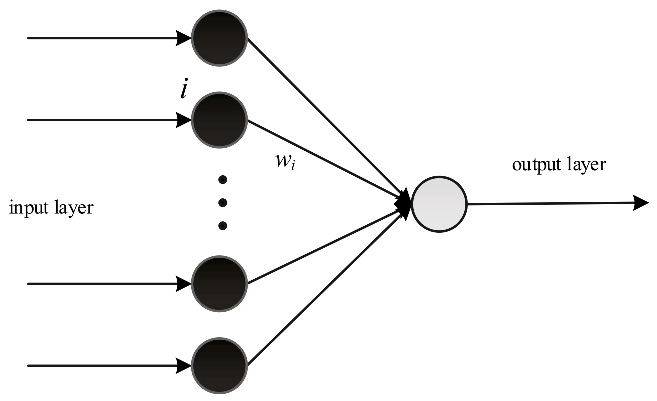


Fig. 4. Architecture of a single-layer spiking neural network.

3.2.4. *Similarity measure for spike trains*

Similarity measure for spike trains is a quantitative method to calculate the degree of similarity between spike trains. Researchers have already provided some different measurement methods (Dauwels, Vialatte, Weber, & Cichocki, 2008; Kreuz, Haas, Morelli, Abarbanel, & Politi, 2007; Naud, Gerhard, Mensi, & Gerstner, 2011). In the supervised learning process of SNNs, similarity measure between the actual and desired output spike trains is needed to calculate their errors. Through the calculation of spike train errors, on the one hand, the accuracy of supervised learning is calculated, and when the error is less than a given value, the process of learning iteration is ended. On the other hand, in some supervised learning algorithms (e.g., gradient-descent-based supervised learning algorithms), the defined specific error function is applied to the derivation of learning rules.

4. Supervised learning for single-layer SNNs

4.1. *The architecture of single-layer SNNs*

A single-layer SNN is a class of special feed-forward SNNs. It contains only one input layer and one output layer and does not contain any hidden layers. There is only one layer of synaptic weight in single-layer SNNs. Therefore, supervised learning for single-layer SNNs is actually for spiking neurons. Because of its simple architecture, a single-layer SNN is widely used in SNN simulation. Fig. 4 shows a single-layer SNN architecture with a single output neuron, where w_i is the connection weight of the ith synapse.

4.2. *Learning algorithms for single-layer SNNs*

4.2.1. *Perceptron-based algorithms*

Supervised learning for a spiking neuron is equivalent to correctly distinguishing the times of desired output spikes and the other times during the running process of the neuron by adjusting the synaptic weights. This is actually a classification problem, so the perceptron (Minsky & Papert, 1969; Rosenblatt, 1958) can be used to solve this problem (Du et al., 2015; Xu, Zeng, & Zhong, 2013).

Gütig and Sompolinsky (2006) proposed a biologically plausible supervised synaptic learning rule named *tempotron*, in which the spiking neurons realize the optimization of the synaptic weights by minimizing the error between the actual output potential and the desired output potential. The synaptic weight adjustment with the tempotron learning rule is expressed as

$$-\frac{dE_\pm}{dw_i} = \pm \sum_{t_i<t_{max}} K(\Delta t_i) \pm \frac{\partial V(t_{max})}{\partial t_{max}} \frac{dt_{max}}{dw_i}, \tag{7}$$

where t_i is the spike time of the ith afferent neuron, $K(t)$ is the normalized postsynaptic potential, t_{max} is the time at which the postsynaptic potential reaches its maximum value, and $\Delta t_i = t_{max} - t_i$ is the time difference between the activation of a synapse and the voltage maximum. By training synaptic weights of integrate-and-fire neurons in the input layer, the tempotron successfully implements the classification of single-spike spatiotemporal patterns.

Urbanczik and Senn (2009) derived another supervised learning rule for the tempotron task using the gradient descent rule. Experimental results showed that the new algorithm converges more quickly and reliably than the original tempotron rule, and does not rely on a specific reset mechanism. Yu, Tang, Tan, and Li (2013a) used the tempotron learning rule to recognize images of digits in the MNIST database. Zhao, Yu, Ding, Chen, and Tang (2014) simulated the tempotron with the event-driven method. Inspired by the tempotron, Roy, San, Hussain, Wei, and Basu (2016) proposed a morphological learning algorithm that can be used to find the optimal morphology of neurons with nonlinear dendrites and binary synapses.

Xu, Zeng, and Zhong (2013) proposed a perceptron-based spiking neuron learning rule (PBSNLR) with temporal encoding. It transforms supervised learning into a classification problem and then solves the problem by using the perceptron learning rule. The PBSNLR can be expressed as

$$w_i^{new} = \begin{cases} w_i^{old} + \eta P_i^{t_d} & \text{if } d^{t_d} = 1 \text{ and } a^{t_d} = 0, \\ w_i^{old} - \eta P_i^{t_d} & \text{if } d^{t_d} = 0 \text{ and } a^{t_d} = 1, \\ w_i^{old} & \text{if } d^{t_d} = a^{t_d}, \end{cases} \tag{8}$$

where w_i is the weight of the ith synapse, η is the learning rate, $P_i^{t_d}$ is the sum of postsynaptic potentials induced by all the spikes that have arrived through the ith synapse at t_d, and d^{t_d} and a^{t_d} are signals to represent whether t_d is the desired or actual output spike time, respectively. Qu, Xie, Liu, Zhang, and Lu (2015) further improved the PBSNLR by use of dynamic learning parameters with distance items and a better method for choosing negative samples. Then they used the improved PBSNLR for real-time user authentication. Zhang, Qu, Xie, and Kurths (2017) analyzed the strategies used to make spiking neurons robust to noise and proposed a training strategy that trains spiking neurons with a dynamic firing threshold named *noise threshold* to improve the noise tolerance of the PBSNLR. The noise threshold can be applied by other supervised learning methods for spiking neurons.

4.2.2. *Synaptic plasticity algorithms*

Synaptic plasticity plays an important role in learning and memory (Fusi, Annunziato, Badoni, Salamon, & Amit, 2000; Kiselev, 2017). Hebb (2005) first proposed a synaptic plasticity hypothesis, which emphasizes the importance of the synergistic activity between presynaptic and postsynaptic neurons on increasing synaptic potentiation (Kuriscak, Marsalek, Stroffek, & Toth, 2015). The mathematical model of the Hebb learning rule can be expressed as

$$\Delta w_{ij} \propto v_i v_j, \tag{9}$$

where v_i and v_j are neuronal activities of presynaptic and postsynaptic neurons, respectively. The spike train can not only cause persistent changes in synaptic strength but also satisfies the STDP learning rule (Caporale & Dan, 2008; Feldman, 2012). The STDP learning window for synaptic changes can be expressed mathematically as

$$W(s)^{STDP} = \begin{cases} +A_+ \exp(-\frac{s}{\tau_+}) & \text{if } s \geq 0, \\ -A_- \exp(\frac{s}{\tau_-}) & \text{if } s < 0, \end{cases} \tag{10}$$

where $s = t_{post} - t_{pre}$ is the time delay between the presynaptic spike arrival time t_{pre} and the postsynaptic firing time t_{post}, A_+ and A_- are the amplitudes, and τ_+ and τ_- are the time constants of the learning window.

On the basis of the STDP learning mechanism, researchers have proposed many biologically plausible supervised learning algorithms for single-layer SNNs. Ruf and Schmitt (1997) first provided a supervised Hebbian learning algorithm based on the spike firing time. However, this algorithm can learn only a single spike. Legenstein, Naeger, and Maass (2005) used the STDP mechanism to propose a supervised Hebbian learning algorithm for spiking neurons. In this algorithm, the external injected synaptic current is used as a teacher signal. They further provided tools for the analytic treatment of reward-modulated STDP (Frémaux, Sprekeler, & Gerstner, 2010), which allows us to predict under what conditions reward-modulated STDP will achieve the desired learning effect (Legenstein, Pecevski, & Maass, 2008a, 2008b). Similar work was presented in Bing et al. (2019). Gardner and Grüning (2013) implemented a stochastic neuron model and investigated its ability to learn a target spike train in response to a spatiotemporal spiking pattern through a reward-modulated STDP rule. Franosch, Urban, and van Hemmen (2013) proved that under very general conditions, supervised STDP converges to a stable configuration of synaptic weight leading to a reconstruction of primary sensory input. Jeyasothy, Sundaram, and Sundararajan (2019) presented a time-varying long-term synaptic efficacy function-based leaky integrate-and-fire (LIF) neuron model (SEFRON). Corresponding supervised learning rule based on STDP was also presented for pattern classification problems.

Ponulak and Kasiński (2010) represented the adjustment of synaptic weight as a combination of STDP and anti-STDP and proposed a remote supervised method (ReSuMe) for spiking neurons. The adjustment of synaptic weight can be expressed as

$$\Delta w_i(t) = [s_d(t) - s_a(t)] \left[a + \int_0^\infty W(s) s_i(t-s) ds \right], \tag{11}$$

where $s_i(t)$, $s_a(t)$, and $s_d(t)$ are the input spike train, the actual output spike train, and the desired output spike train, respectively, and a is the non-Hebbian item to accelerate the convergence of the training process. The integral kernel $W(s)$ defines the synaptic plasticity determined by the spike time correlation. The ReSuMe algorithm can be applied to various neuron models (Ponulak & Kasiński, 2008). However, the ReSuMe algorithm can learn the synaptic weight only of the output layer of the liquid state machine, in which the hidden layer with recurrent architecture is a static network. The ReSuMe algorithm has been used in various spatiotemporal pattern classification and recognition problems (Glackin, Maguire, McDaid, & Sayers, 2011; Hu, Tang, Tan, Li, & Shi, 2013).

In recent years, Taherkhani, Belatreche, Li, and Maguire (2015a) proposed a delay learning remote supervised method (DL-ReSuMe) for spiking neurons to merge the delay shift approach and the ReSuMe-based weight adjustment to improve the learning performance. In DL-ReSuMe, synaptic weights are updated by the delayed version of the ReSuMe rule, which is expressed as

$$\Delta w_i(t) = [s_d(t) - s_a(t)] \left[a + \int_0^\infty W(s) s_i(t - dt_i - s) ds \right], \tag{12}$$

where dt_i is the synaptic delay of the ith synapse. They further extended DL-ReSuMe to train multiple neurons to classify spatiotemporal spiking patterns and proposed the Multi-DL-ReSuMe algorithm (Taherkhani, Belatreche, Li, & Maguire, 2015b). Guo, Wang, Cabrerizo, and Adjouadi (2017) proposed a cross-correlated delay shift (CCDS) supervised learning method for spiking neurons, in which synapse delays and axonal delays are variants and are modulated together with weights during learning. The CCDS learning method is another improved version of ReSuMe achieved by integrating synaptic delays and axonal delays in the synaptic weight learning process. In addition, combining the ReSuMe algorithm and triplet-based STDP (Pfister & Gerstner, 2006), Lin, Chen, Wang, and Ma (2016) proposed a learning algorithm (T-ReSuMe) for spiking neurons.

4.2.3. Spike train convolution algorithms

Since the spike train is a set consisting of discrete spike events, to facilitate analysis and calculation, a symmetric and positive definite kernel function $\kappa(t)$ (Park, Seth, Rao, & Príncipe, 2012) can be chosen to convert the spike train to a unique continuous function by use of the convolution

$$f_s(t) = s(t) * \kappa(t) = \sum_{f=1}^{F} \kappa\left(t - t^f\right). \tag{13}$$

For any two given spike trains $s_i(t)$ and $s_j(t)$, the inner product of the corresponding continuous functions $f_{s_i}(t)$ and $f_{s_j}(t)$ can be defined in $L_2(\Gamma)$ space as (Paiva, Park, & Príncipe, 2009)

$$F\left(s_i(t), s_j(t)\right) = \left\langle f_{s_i}(t), f_{s_j}(t)\right\rangle_{L_2(\Gamma)} = \int_\Gamma f_{s_i}(t) f_{s_j}(t) dt. \tag{14}$$

Using the inner product representation of spike times, one can further rewrite Eq. (14) as the accumulation form of spike time pairs with order $O(F_iF_j)$:

$$\begin{aligned} F\left(s_i(t), s_j(t)\right) &= \sum_{m=1}^{F_i} \sum_{n=1}^{F_j} \int_\Gamma \kappa\left(t - t_i^m\right) \kappa\left(t - t_j^n\right) dt \\ &= \sum_{m=1}^{F_i} \sum_{n=1}^{F_j} \kappa\left(t_i^m, t_j^n\right), \end{aligned} \tag{15}$$

where t_i^m and t_j^n are spike times in the spike trains $s_i(t)$ and $s_j(t)$, respectively, and F_i and F_j are the numbers of spikes in $s_i(t)$ and $s_j(t)$, respectively.

Because $F(s_i(t), s_j(t))$ has symmetry and positivity properties, according to the Moore–Aronszajn theorem, there exists a reproducing kernel Hilbert space (RKHS) H_F induced by the spike train inner product F (Paiva et al., 2009). With use of the inner products of spike trains, a formal definition of the similarity measure of spike trains can be obtained (Park, Seth, Paiva, Li, & Principe, 2013), which is the basis of an error function for supervised learning in SNNs.

Carnell and Richardson (2005) used the linear algebra method to learn the spatiotemporal patterns of spike trains, in which the adjustment of synaptic weight is calculated by the projection defined through the Gram–Schmidt process (Leon, Björck, & Gander, 2013). Later, Mohemmed, Schliebs, Matsuda, and Kasabov (2012, 2013) proposed a spike pattern association neuron (SPAN) supervised learning algorithm for learning spatiotemporal spike patterns. The idea of the SPAN algorithm is to transform the spike trains during the learning phase into analog signals by an α-kernel function so that common mathematical operations can be performed on them. The adjustment of synaptic weight with the SPAN algorithm is expressed as

$$\Delta w_i(t) = \eta f_{s_i}(t) \left[f_{s_d}(t) - f_{s_a}(t) \right], \tag{16}$$

where $f_{s_i}(t)$, $f_{s_a}(t)$, and $f_{s_d}(t)$ are the convolved continuous functions corresponding to the input spike train, the actual output spike train, and the desired output spike train, respectively. The SPAN algorithm can also realize an offline or incremental learning process (Mohemmed & Kasabov, 2012). Inspired by the SPAN algorithm, Yu, Tang, Tan, and Li (2013b) proposed a precise-spike-driven (PSD) supervised learning rule that can be used to train neurons to associate an input spatiotemporal spike pattern with a desired spike train. The PSD learning rule convolves the input spike trains only by the double-exponential kernel function. In the experiment, the PSD algorithm was applied to optical character recognition and a more complex recognition problem involving handwritten digits (Yu, Tang, Tan, & Yu, 2014; Yu, Yan, Tang, Tan, & Li, 2016).

Recently, Lin, Wang, and Dang (2016) proposed a spike train kernel learning rule (STKLR) for spiking neurons. Various kernel functions are tested in the STKLR algorithm. The adjustment of synaptic weights in the STKLR algorithm is expressed as

$$\Delta w_i = \eta \left[\sum_{g=1}^{F_d} \sum_{f=1}^{F_i} \kappa\left(t_d^g, t_i^f\right) - \sum_{h=1}^{F_a} \sum_{f=1}^{F_i} \kappa\left(t_a^h, t_i^f\right) \right], \tag{17}$$

where η is the learning rate. t_i^f, t_a^h, and t_d^g are the spikes in the input spike train, the actual output spike train, and the desired output spike train, respectively, and κ is the kernel function. They then used the STKLR algorithm to solve an image recognition and classification problem (Ma, Lin, & Wang, 2018). Further, they proposed some other related supervised learning algorithms (Lin, Ning, & Wang, 2015; Wang, Lin, & Dang, 2019; Wang, Lin, Zhao, & Ma, 2016) based on the STKLR using the online learning mechanism, nonlinear spike train kernels, and the delay learning method.

4.2.4. Other supervised learning algorithms

Pfister, Toyoizumi, Barber, and Gerstner (2006) derived a supervised spike-based learning algorithm starting with statistical learning criteria and neural spike train analysis (Chen, 2013). They used a supervised learning paradigm to derive the synaptic weight updating rule that optimizes the likelihood of postsynaptic firing at one or several desired spike times by gradient ascent. On the basis of the statistical method of Pfister et al. (2006), Gardner and Grüning (2016) proposed two spike-based learning methods: one that relies on an instantaneous error signal to update synaptic weights in a network (INST rule) and one that relies on a filtered error signal for smoother synaptic weight modifications (FILT rule).

Inspired by learning rules for locally recurrent analog neural networks, Schrauwen and Van Campenhout (2006) presented a new learning rule for spiking neurons that uses the general population-temporal encoding model. The proposed method can learn very fast and can operate on a wide class of decoding schemes. Dora, Suresh, and Sundararajan (2014, 2015) presented a biologically inspired structure learning algorithm for single-layer SNNs to solve pattern classification problems using population encoding. The learning algorithm can choose either to add a new output neuron or to update the synaptic weight of existing output neurons. On the basis of a rank-order-based learning rule (Dhoble, Nuntalid, Indiveri, & Kasabov, 2012; Wysoski, Benuskova, & Kasabov, 2006), Wang, Belatreche, Maguire, and McGinnity (2017) presented an enhanced rank-order-based learning algorithm, called *SpikeTemp*, for single-layer SNNs with a dynamically adaptive structure.

Tapson et al. (2013) proposed a synaptic kernel inverse method (SKIM) for spiking neurons. The pseudo-inverse method or any similar convex optimization can be used to train the SKIM algorithm, so it may be an online, adaptable, and biologically plausible supervised learning algorithm. Lee, Kukreja, and Thakor (2017) proposed two novel convex-optimized synaptic efficacy (CONE) algorithms to estimate the weight for spiking neurons in a convex optimization framework, which is similar to the SKIM. Their main contribution is to convert the supervised learning problem for spiking neurons into a standard optimization problem.

Zhang, Qu, Belatreche, Chen, and Yi (2019) and Zhang, Qu, Belatreche, and Xie (2018) presented an efficient membrane-potential-driven supervised learning algorithm named *MemPo-Learn* for spiking neurons. At the desired output time, the gradient descent method is implemented to minimize the error function defined as the difference between the membrane potential and the firing threshold:

$$E = \frac{1}{2}\left[u_i(t) - V_{thresh}\right]^2, \tag{18}$$

where $u_i(t)$ is the membrane potential of the neuron and V_{thresh} is the firing threshold. At an undesired output time, synaptic weights are adjusted to make the membrane potential below the threshold, and the error function is defined as

$$E = \frac{1}{2}\left[u_i(t) - (V_{thresh} - p)\right]^2, \tag{19}$$

where p determines the magnitude of modification of the synaptic weight. Similar studies are presented in Lin, Ma, Meng, and Chen (2018), Luo, Qu, Zhang, and Chen (2019) and Zhang, Geng, et al. (2018). Yu, Li, and Tan (2019) proposed several threshold-driven plasticity rules to train neurons to fire the desired number of output spikes in response to input patterns, which can be used to process spike patterns that are encoded with both a rate coding and a temporal coding. Property proofs of robustness and convergence have also been provided. Motivated by the selective attention mechanism of the primate visual system, Xie, Qu, Yi, and Kurths (2017) proposed an accurate synaptic-efficiency adjustment (ASA) method to increase the efficiency of training SNNs. The adjustment of synaptic weights in the ASA algorithm combines the membrane-potential-driven mechanism with the STDP learning window.

4.3. Performance comparison of spike train learning

In this subsection, the spike train learning performances of some representative supervised learning algorithms for single-layer SNNs are compared. The ReSuMe (Ponulak & Kasiński, 2010), SPAN (Mohemmed et al., 2012), PSD (Yu et al., 2013b), and STKLR (Lin et al., 2016) algorithms are selected. In addition, the supervised learning algorithm for multilayer feed-forward SNNs proposed by Xu, Zeng, Han, and Yang (2013), which is a typical gradient-descent-based supervised learning algorithm, is considered. The weight updating rule of the output layer is used to train the single-layer SNNs. For convenience, it is called the *spike train SpikeProp (ST-SpikeProp) rule*. All of these algorithms can achieve spike train learning. There are 500 input neurons and one output neuron in the single-layer SNNs. The reference length of the spike trains is 400 ms and the reference firing rate of the spike trains is 60 Hz.

First, the learning performance of these algorithms with different lengths of spike trains was tested. The length of the spike trains was increased from 200 ms to 1000 ms in steps of 200 ms while the other settings remained the same. The learning rates of these algorithms corresponding to the different lengths of the spike trains are shown in Table 2. The learning results are shown in Fig. 5. Fig. 5(a) shows the average learning accuracy C after 1000 learning epochs. It can be seen that the learning accuracy C of all these algorithms decreases gradually with increasing length of the spike trains. The SPAN algorithm always has the highest learning accuracy. Fig. 5(b) shows the average number of learning epochs when the learning accuracy C reaches the maximum value. The learning epochs of these algorithms have no significant differences under the current conditions.

Furthermore, the learning performance of these algorithms with different firing rates of spike trains was tested. The firing rate of spike trains was increased from 20 Hz to 180 Hz in steps

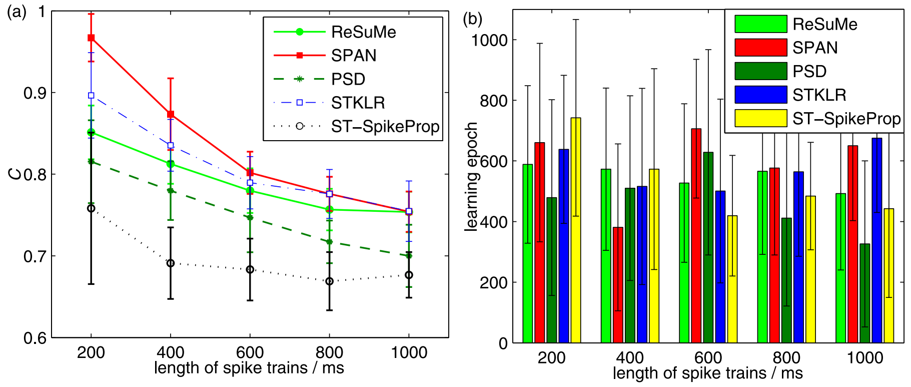


Fig. 5. Learning results with different lengths of spike trains for single-layer spiking neural networks after 1000 learning epochs. (a) The learning accuracy C. (b) The learning epochs when the learning accuracy C reaches the maximum value. PSD, precise spike driven; SPAN, spike pattern association neuron; STKLR, spike train kernel learning rule; ST-SpikeProp, spike train SpikeProp.

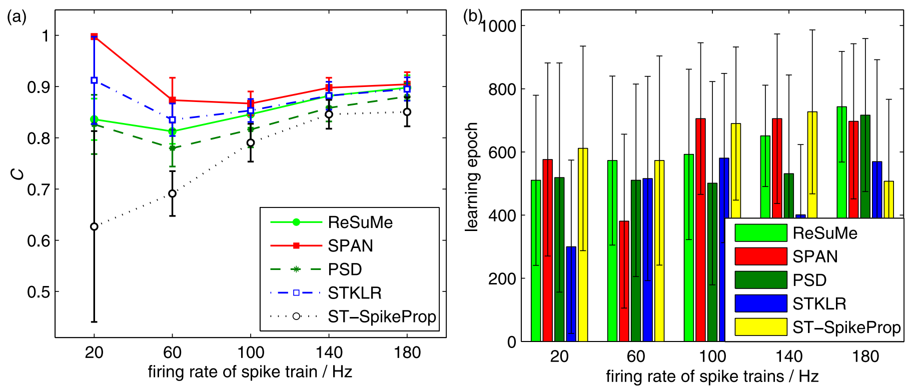


Fig. 6. Learning results with different firing rates of spike trains for single-layer spiking neural networks after 1000 learning epochs. (a) The learning accuracy C. (b) The learning epochs when the learning accuracy C reaches the maximum value. PSD, precise spike driven; SPAN, spike pattern association neuron; STKLR, spike train kernel learning rule; ST-SpikeProp, spike train SpikeProp.

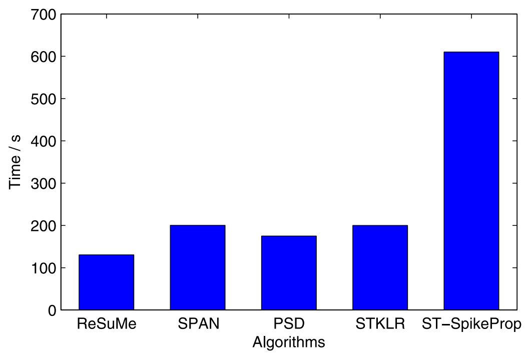


Fig. 7. Running time of one trial of selected algorithms. PSD, precise spike driven; SPAN, spike pattern association neuron; STKLR, spike train kernel learning rule; ST-SpikeProp, spike train SpikeProp.

of 40 Hz while the other settings remained the same. The learning rates of these algorithms corresponding to the different firing rates of spike trains are shown in Table 3. The learning results are shown in Fig. 6. Fig. 6(a) shows the average learning accuracy C after 1000 learning epochs, while Fig. 6(b) shows the average number of learning epochs when the learning accuracy C reaches the maximum value. Overall, the spike train learning performance of the SPAN algorithm is the best, while that of the ST-SpikeProp algorithm is the worst.

Table 2
Learning rates of selected algorithms with different lengths of spike trains.

Algorithm	200 ms	400 ms	600 ms	800 ms	1000 ms
ReSuMe	0.001	0.001	0.0005	0.0005	0.0005
SPAN	0.0001	0.00005	0.00001	0.00001	0.00001
PSD	0.001	0.001	0.001	0.0005	0.0005
STKLR	0.001	0.001	0.001	0.001	0.0001
ST-SpikeProp	0.000001	0.000001	0.0000001	0.00000005	0.00000001

PSD, precise spike driven; SPAN, spike pattern association neuron; STKLR, spike train kernel learning rule; ST-SpikeProp, spike train SpikeProp.

Table 3
Learning rates of selected algorithms with different firing rates of spike trains.

Algorithm	20 Hz	60 Hz	100 Hz	140 Hz	180 Hz
ReSuMe	0.001	0.001	0.0005	0.0001	0.00005
SPAN	0.0001	0.00005	0.00001	0.00001	0.00001
PSD	0.01	0.001	0.0005	0.0005	0.0001
STKLR	0.01	0.001	0.0005	0.0005	0.0005
ST-SpikeProp	0.000001	0.000001	0.0000005	0.0000005	0.0000001

PSD, precise spike driven; SPAN, spike pattern association neuron; STKLR, spike train kernel learning rule; ST-SpikeProp, spike train SpikeProp.

Fig. 7 shows the average running time of one trial of these algorithms under the reference conditions. The ST-SpikeProp algorithm has the longest running time because of its intrinsic complex mechanism. The running time of the other algorithms is relatively short.

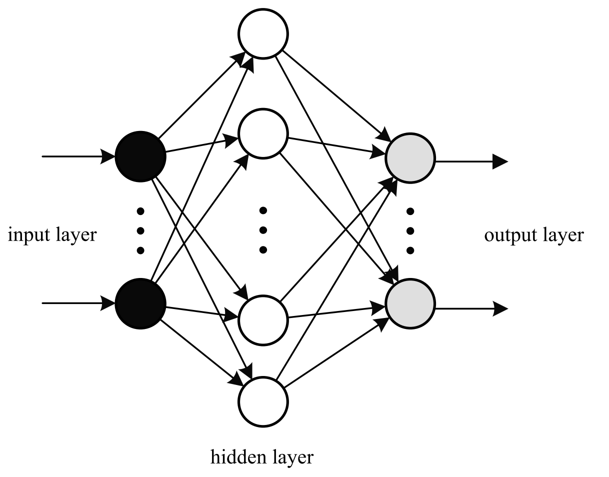


Fig. 8. Architecture of a multilayer feed-forward spiking neural network.

5. Supervised learning for multilayer feed-forward SNNs

5.1. The architecture of multilayer feed-forward SNNs

A multilayer feed-forward SNN is one of the commonly used SNN architecture. Neurons in a multilayer feed-forward SNN are hierarchically arranged. Each neuron is connected only to the neurons in the previous layer. The first layer is the input layer of the SNN. This layer does not contain any computation of neurons and represents the input of the SNN. The spike train of each neuron in the input layer represents the coding of the input data for the specific problem, which is input into the next layer of the SNN. The last layer is the output layer, and the spike train fired by each neuron in this layer constitutes the output of the network. There may be one or more hidden layers in between the input and output layers. An architecture of a multilayer feed-forward SNN is shown in Fig. 8.

5.2. Learning algorithms for multilayer feed-forward SNNs

5.2.1. Gradient descent algorithms

Back-propagation is an extremely important technique for designing learning algorithms and helping us to understand the brain (Lillicrap & Santoro, 2019; Whittington & Rafal, 2019). Supervised learning algorithms for SNNs based on the gradient descent rule use the gradient calculation and error back-propagation to adjust the synaptic weights, and ultimately minimize the error function that indicates the deviation between the actual and the desired output spikes. For supervised learning algorithms based on the gradient descent rule, the error function E between the actual and the desired output spike trains should be defined first, and then all synaptic weights are adjusted by the delta updating rule. The synaptic weight modification Δw is calculated according to

$$\Delta w = -\eta \nabla E = -\eta \frac{\partial E}{\partial w}, \tag{20}$$

where η is the learning rate and ∇E is the gradient of the error function E to the synaptic weight w.

Drawing lessons from the back-propagation algorithm (Rummelhart, 1986) in traditional ANNs, Bohte, Kok, and Poutré (2002) first proposed a back-propagation training algorithm for multilayer feed-forward SNNs, called *SpikeProp*, in which the SRM is used. To overcome the discontinuity of the internal state variable caused by spike firing, they approximated the thresholding function. All neurons in the SNNs are limited to fire only a single spike. Yang, Yang, and Wu (2012a) further proved that the assumption of Bohte et al. is mathematically correct. The error function of SNNs in the form of the least mean square is defined as

$$E = \frac{1}{2} \sum_{o=1}^{N_O} \left(t_o^a - t_o^d \right)^2, \tag{21}$$

where t_o^a and t_o^d are output and desired spikes of the oth output neuron in the output layer, respectively. For a multiple-synapse model, the adjustment of the kth synaptic weight between the presynaptic neuron i and the postsynaptic neuron j is expressed as

$$\Delta w_{ij}^k = -\eta \frac{\partial E}{\partial w_{ik}^k} = -\eta y_i^k (t_j^a) \delta_j, \tag{22}$$

where $y_i^k(t_j^a)$ is the postsynaptic potential without synaptic weight and δ_j represents the different gradient descent learning rule depending on in which layer the synapse is located. Experimental results showed that the SpikeProp algorithm can solve the nonlinear pattern classification problem.

In recent years, researchers have analyzed and improved the SpikeProp algorithm from different aspects. Xin and Embrechts (2001) presented a method with a simple momentum term that increases the convergence speed of the SpikeProp algorithm. The learning performance of the SpikeProp algorithm is further enhanced by adding learning rules based on gradient descent for parameters such as synaptic delays, the threshold of spike firing, and the time constant (Matsuda, 2016a, 2016b; Schrauwen & Van Campenhout, 2004a, 2004b; Thiruvarudchelvan, Moore, & Antolovich, 2013; Wu, McGinnity, Maguire, Glackin, & Belatreche, 2006). McKennoch, Liu, and Bushnell (2006) proposed the RProp and QuickProp algorithms, with faster convergence, and further extended SpikeProp to a class of nonlinear neuron models and constructed a back-propagation algorithm in theta neuron networks (McKennoch, Voegtlin, & Bushnell, 2009). Yang, Yang, and Wu (2012b) proposed a modified spiking neuron model that involves the derivative of the state function at spike firing time. The supervised learning algorithm for the modified spiking neuron was also given. Shrestha and Song (2015, 2016, 2017a, 2017b, 2018) improved the SpikeProp algorithm by use of an adaptive learning rate, adaptive delays, and an event-driven simulation strategy. Zhao, Zurada, Yang, and Wu (2018) further improved the SpikeProp algorithm by introducing the smoothing $L_{1/2}$ regularization term into the error function and proved convergence of this algorithm under some reasonable conditions. Mostafa (2018) defined the network input–output relation as locally linear after a transformation of the time variable and then trained the SNN through gradient descent directly. However, all of the above-mentioned algorithms encode information using a single spike, a limitation that means they cannot be effective for solving complex spatiotemporal problems.

A more important extension of the SpikeProp algorithm was presented by Booij and tat Nguyen (2005). Their algorithm allows the neurons in the input and hidden layers to fire multiple spikes, but only the first spike in the output layer is considered. The error function of the SNNs is the same as that of the SpikeProp algorithm, using Eq. (21). On the basis of Booij and tat Nguyen (2005), Fang and Wang (2008) derived an additional error back-propagation learning rule for the coefficient of the refractoriness function, in which the dependence of the postsynaptic potential on the firing times of the postsynaptic neuron is not ignored. Similarly, Ghosh-Dastidara and Adeli (2009) proposed a back-propagation learning algorithm named *Multi-SpikeProp*, with derivations of the learning rule based on the chain rule for a multiple spiking network model. Multi-SpikeProp was applied to the standard XOR problem and the Fisher *Iris* and

EEG classification problems. Experimental results showed that the Multi-SpikeProp algorithm has higher classification accuracy than the SpikeProp algorithm. Luo et al. (2017, 2018) used the dynamic momentum and learning rate adaption and antinoise learning rule to improve the learning performance of the extended SpikeProp algorithm presented by Booij and tat Nguyen (2005).

Supervised learning for multiple spikes is more complex than that for a single spike, in which the error function involves multiple spikes in computing. For any given input spike trains, the least mean square error function of SNNs for multiple spikes is no longer expressed by Eq. (21) but is defined as

$$E = \frac{1}{2}\sum_{o=1}^{N_O}\sum_{f=1}^{F_o}\left(t_o^f - \hat{t}_o^f\right)^2, \tag{23}$$

where t_o^f and $\hat{t}_o^f$ are output and desired spikes, respectively. Recently, Xu, Zeng, Han, and Yang (2013) extended the Multi-SpikeProp algorithm to allow neurons to fire multiple spikes in all layers; that is, the algorithm can implement the complex spatiotemporal pattern learning of spike trains. Gong (2013) further investigated the supervised learning algorithm based on multiple-spike back-propagation in combination with the long-term memory characteristics of the SRM. Gardner, Sporea, and Grüning (2015) introduced a new supervised learning rule named *MultilayerSpiker* that can train multilayer feed-forward SNNs to perform transformations between spatiotemporal input and output spike patterns. The stochastic spiking neuron model is used in this method. The learning rule is robust against input noise and could generalize well on an example dataset.

Banerjee (2016) derived a synaptic weight update rule for learning temporally precise spike-train-to-spike-train transformations in multilayer feed-forward networks of spiking neurons. The framework, aiming at seamlessly generalizing error back-propagation to the deterministic spiking neuron setting, is based strictly on spike timing and avoids invoking concepts about spike rates or probabilistic models of spiking. Wu, Deng, Li, Zhu, and Shi (2018) proposed a spatiotemporal back-propagation training framework for SNNs that combines both the spatial domain and the temporal domain in the training phase. Experimental results showed that the proposed method achieved high accuracy on either static or dynamic datasets. Zenke and Ganguli (2018) derived a nonlinear voltage-based three-factor learning rule named *SuperSpike* by using a surrogate gradient approach to train multilayer feed-forward SNNs. They also assessed the impact of different types of feedback credit assignment strategies for the hidden units.

5.2.2. Synaptic plasticity algorithms

Researchers have proposed various supervised learning algorithms for multilayer feed-forward SNNs combining synaptic plasticity with the gradient descent rule. Strain, McDaid, McGinnity, Maguire, and Sayers (2010) presented a cross-correlated training algorithm for SNNs that uses an extended form of the STDP rule. The input and output neurons are conductance-based LIF neurons, while the hidden neurons are dynamic threshold neurons in their model. By combination of the Bienenstock–Cooper–Munro learning rule (Bienenstock, Cooperk, & Munro, 1982; Izhikevich & Desai, 2003) with the STDP mechanism, a synaptic weight association training algorithm (Wade, McDaid, Santos, & Sayers, 2010) for SNNs was proposed that yields a unimodal synaptic weight distribution where weight stabilization is achieved with the sliding threshold associated with the Bienenstock–Cooper–Munro model after a period of training. Zheng and Mazumder (2018a, 2018b) formulated an online learning algorithm for hardware-based multilayer feed-forward SNNs through the modulation of weight-dependent STDP. The proposed learning method can estimate the gradient components with the help of the spike timings in an SNN.

Sporea and Grüning (2013) extended the ReSuMe algorithm (Ponulak & Kasiński, 2010) to multilayer feed-forward SNNs using back-propagation of the network error and proposed the Multi-ReSuMe algorithm. The Multi-ReSuMe algorithm can be applied to neurons firing multiple spikes in SNNs. Synaptic weight modification at time t for the output neurons is expressed as

$$\begin{aligned}\Delta w_{oh}(t) =& \frac{1}{N_H}s_h(t)\left\{\int_0^\infty W^{pre}(s)\left[s_d^o(t-s) - s_a^o(t-s)\right]ds\right\} \\ &+ \frac{1}{N_H}\left[s_d^o(t) - s_a^o(t)\right]\left[a + \int_0^\infty W^{post}(s)s_h(t-s)ds\right],\end{aligned} \tag{24}$$

where N_H is the number of hidden neurons, $s_h(t)$ is the spike trains fired by hidden neurons, and a is a non-Hebbian term. Synaptic weight modification at time t for the hidden neurons is expressed as

$$\begin{aligned}\Delta w_{hi}(t) =& \frac{1}{N_I N_H}s_i(t)\sum_{o=1}^{N_O}\left\{\int_0^\infty W^{pre}(s)\left[s_d^o(t-s) - s_a^o(t-s)\right]ds\right\}w_{oh} \\ &+ \frac{1}{N_I N_H}\sum_{o=1}^{N_O}\left[s_d^o(t) - s_a^o(t)\right]\left[a + \int_0^\infty W^{post}(s)s_i(t-s)ds\right]w_{oh},\end{aligned} \tag{25}$$

where N_I and N_O are the numbers of input neurons and output neurons, respectively. In addition, Taherkhani, Belatreche, Li, and Maguire (2018) extended their DL-ReSuMe (Taherkhani et al., 2015a) to train both synaptic weights and synaptic delays of a multilayer feed-forward SNN to fire multiple desired spikes.

5.2.3. Spike train convolution algorithms

Lin, Wang, and Hao (2017) extended their STKLR algorithm (Lin et al., 2016) to multilayer feed-forward SNNs and presented a new supervised learning algorithm for multilayer SNNs (Multi-STIP). They first constructed a novel error function of spike trains based on spike train kernels and then derived a synaptic weight learning rule combining the mechanism of error back-propagation. In Multi-STIP, the adjustment rule for synaptic weight between a neuron in the output layer and a neuron in the hidden layer is

$$\begin{aligned}\Delta w_{oh} &= -\eta\left[F(s_a^o(t), s_h(t)) - F(s_d^o(t), s_h(t))\right] \\ &= \eta\left[\sum_{g=1}^{F_d^o}\sum_{k=1}^{F_h}\kappa\left(t_d^g, t_h^k\right) - \sum_{j=1}^{F_a^o}\sum_{k=1}^{F_h}\kappa\left(t_a^j, t_h^k\right)\right],\end{aligned} \tag{26}$$

and the adjustment rule for synaptic weight between a neuron in the hidden layer and a neuron in the input layer is

$$\begin{aligned}\Delta w_{hi} &= -\eta\sum_{o=1}^{N_O}\left[F(s_a^o(t), s_i(t)) - F(s_d^o(t), s_i(t))\right]w_{oh} \\ &= \eta\sum_{o=1}^{N_O}\left[\sum_{g=1}^{F_d^o}\sum_{f=1}^{F_i}\kappa\left(t_d^g, t_i^f\right) - \sum_{j=1}^{F_a^o}\sum_{f=1}^{F_i}\kappa\left(t_a^j, t_i^f\right)\right]w_{oh},\end{aligned} \tag{27}$$

where N_O is the number of output neurons, t_i^f and t_h^k are spike firing times in the input spike train $s_i(t)$ and hidden neuron spike train $s_h(t)$, respectively, t_a^j and t_d^g are actual and desired spikes corresponding to an output neuron, and F_i, F_h, F_a^o, and F_d^o are the numbers of spikes in $s_i(t)$, $s_h(t)$, $s_a^o(t)$, and $s_d^o(t)$, respectively.

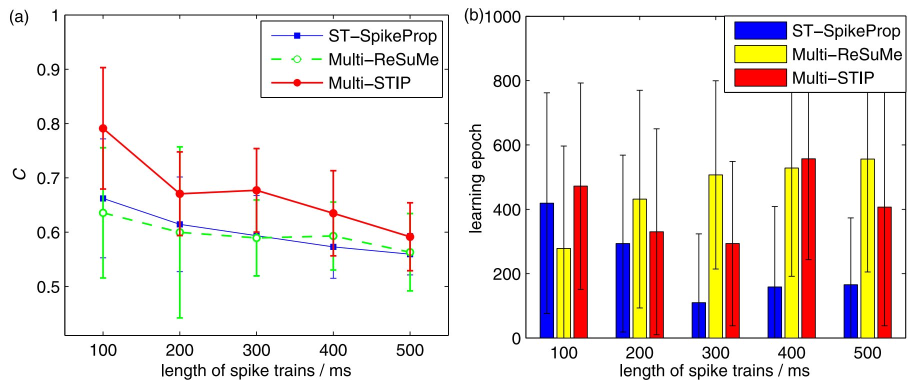


Fig. 9. Learning results with different lengths of spike trains for multilayer feed-forward spiking neural networks after 1000 learning epochs. (a) The learning accuracy C. (b) The learning epochs when the learning accuracy C reaches the maximum value. ST-SpikeProp, spike train SpikeProp.

Drawing on the SPAN algorithm (Mohemmed et al., 2012), Zhang and Lin (2015) proposed another supervised learning algorithm based on the convolution computing of spike trains for multilayer SNNs. This algorithm is the extension of the SPAN algorithm for multilayer feed-forward SNNs. In Yu, Tang, Hu, and Tan (2017), both the PSD rule (Yu et al., 2013b) and the tempotron rule (Gütig & Sompolinsky, 2006) were extended to multiple layers, leading to new rules of multilayer PSD (MutPSD) and multilayer tempotron (MutTmptr).

5.2.4. Other supervised learning algorithms

Belatreche, Maguire, McGinnity, and Wu (2003, 2006) investigated the viability of evolutionary strategies for supervised learning in SNNs. These algorithms can learn only a single spike, and the computation is very time-consuming. Similar work is described in Pavlidis, Tasoulis, Plagianakos, Nikiforidis, and Vrahatis (2005). There are other algorithms that use optimization algorithms to train SNNs (Abusnaina, Abdullah, & Kattan, 2019; Mohemmed, Matsuda, Schliebs, Dhoble, & Kasabov, 2011; Vázquez & Garro, 2011, 2015).

Wang, Belatreche, Maguire, and McGinnity (2014) proposed an online hybrid learning method for feed-forward SNNs with an adaptive structure that combines unsupervised and supervised learning rules. A growing and pruning strategy is used to adjust the structure of the hidden layer, and classification at the output layer is achieved through supervised learning. Similar studies are described in Dora, Sundaram, and Sundararajan (2015), Lin, Chang, Wang, Huang, and He (2018) and Takuya, Haruhiko, Hiroharu, and Shinji (2016).

Xie, Qu, Liu, and Zhang (2017) extended the PBSNLR algorithm (Xu, Zeng, & Zhong, 2013) to multilayer feed-forward SNNs and proposed a normalized perceptron-based learning rule. Differently from traditional methods, the normalized perceptron-based learning rule trains only the selected misclassified time points and the target ones, using the perceptron-based neuron. They also extended their ASA algorithm (Xie, Qu, Yi, & Kurths, 2017) to multilayer feed-forward SNNs and proposed a normalized spiking error back-propagation algorithm (Xie, Qu, Liu, Zhang, & Kurths, 2016).

5.3. Performance comparison of spike train learning

In this subsection, the learning performances of some representative supervised learning algorithms for multilayer feed-forward SNNs are compared. The ST-SpikeProp (Xu, Zeng, Han, & Yang, 2013), Multi-ReSuMe (Sporea & Grüning, 2013), and Multi-STIP (Lin et al., 2017) algorithms are selected. These three algorithms can achieve spike train learning. There are 50 input neurons, 30 hidden neurons, and one output neuron in multilayer feed-forward SNNs. The reference length of the spike trains is 200 ms and the reference firing rate of the spike trains is 40 Hz.

Table 4
Learning rates of selected algorithms with different lengths of spike trains.

Algorithm	100 ms	200 ms	300 ms	400 ms	500 ms
ST-SpikeProp	0.0000005	0.0000001	0.0000001	0.00000005	0.00000001
Multi-ReSuMe	0.01	0.005	0.001	0.0005	0.0001
Multi-STIP	0.01	0.01	0.005	0.005	0.001

ST-SpikeProp, spike train SpikeProp.

Table 5
Learning rates of selected algorithms with different firing rates of spike trains.

Algorithm	20 Hz	40 Hz	60 Hz	80 Hz	100 Hz
ST-SpikeProp	0.000001	0.0000001	0.0000001	0.0000001	0.00000005
Multi-ReSuMe	0.005	0.005	0.001	0.0005	0.0005
Multi-STIP	0.01	0.01	0.005	0.005	0.005

ST-SpikeProp, spike train SpikeProp.

Fig. 9 shows the learning results for these three algorithms with different lengths of spike trains. The length of the spike trains was increased from 100 ms to 500 ms in steps of 100 ms while the other settings remained the same. The learning rates of these algorithms corresponding to the different lengths of the spike trains are shown in Table 4. The average learning accuracy C after 1000 learning epochs is shown in Fig. 9(a). It can be seen that the Multi-STIP algorithm always has the highest learning accuracy. There is little difference in learning accuracy between ST-SpikeProp and Multi-ReSuMe. Fig. 9(b) shows the average number of learning epochs when the learning accuracy C reaches the maximum value. When the length of the spike trains is longer, the learning epoch of ST-SpikeProp is lower.

The learning results for these three algorithms with different firing rates of spike trains are shown in Fig. 10. The firing rate of spike trains was increased from 20 Hz to 100 Hz in steps of 20 Hz while the other settings remained the same. The learning rates of these algorithms corresponding to the different firing rates of spike trains are shown in Table 5. Fig. 10(a) shows the average learning accuracy C after 1000 learning epochs, from which it can be seen that the learning accuracy of the Multi-STIP algorithm is the highest. Fig. 10(b) shows the average number of learning epochs when the learning accuracy C reaches the maximum value. It can be seen that the learning epoch of ST-SpikeProp is relatively low.

Fig. 11 shows the average running time of one trial of these three algorithms under the reference conditions. The ST-SpikeProp algorithm has the longest running time. The running time of Multi-ReSuMe and Multi-STIP is relatively short. The

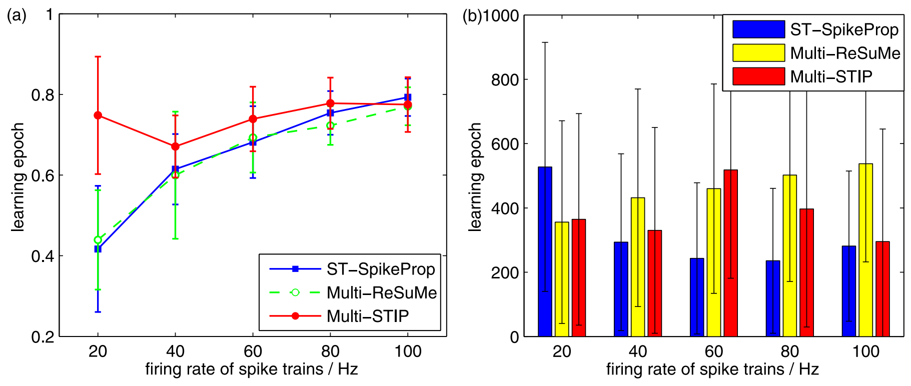


Fig. 10. Learning results with different firing rates of spike trains for multilayer feed-forward spiking neural networks after 1000 learning epochs. (a) The learning accuracy C. (b) The learning epochs when the learning accuracy C reaches the maximum value. ST-SpikeProp, spike train SpikeProp.

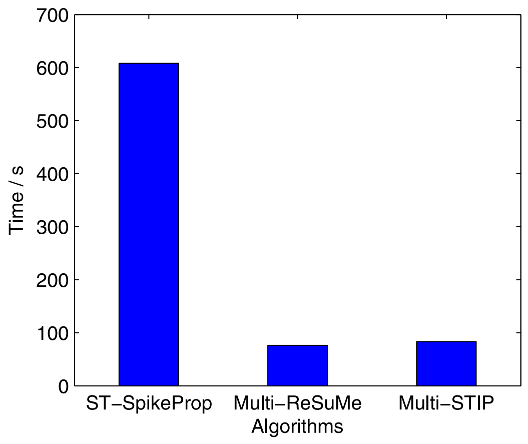


Fig. 11. Running time of one trial of selected algorithms. ST-SpikeProp, spike train SpikeProp.

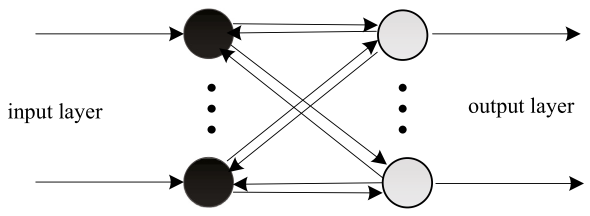


Fig. 12. Architecture of a fully recurrent spiking neural network.

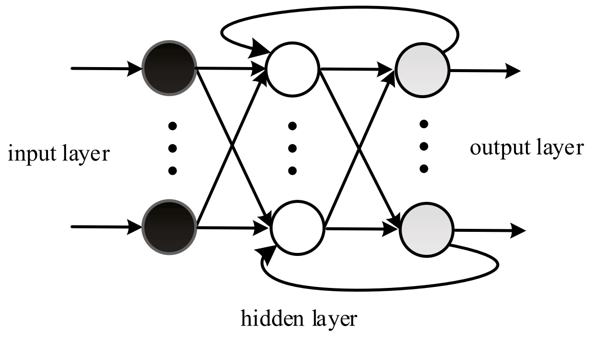


Fig. 13. Architecture of a recurrent spiking neural network with external feedbacks.

Multi-STIP algorithm is suitable for training large-scale feed-forward SNNs because of its high learning accuracy and short running time.

6. Supervised learning for recurrent SNNs

6.1. The architecture of recurrent SNNs

A recurrent SNN is an SNN that has feedback loops in the network. It is difficult to construct the learning algorithms and analyze the dynamics of recurrent SNNs (Amit & Brunel, 1997; Buesing, Bill, Nessler, & Maass, 2011; Pyle & Rosenbaum, 2017; Soula, Beslon, & Mazet, 2006). Maass (1996) and Maass and Natschläger (1997) proved that recurrent SNNs could simulate Turing machines and arbitrary Hopfield neural networks (Hopfield, 1982). There are various architectures of recurrent SNNs, but they can be divided roughly into two kinds: fully recurrent SNNs and locally recurrent SNNs. Because of the difference in architecture between these two types of recurrent SNNs, the construction of their learning algorithms and their dynamic performance are different.

Fully recurrent SNNs are usually single-layer SNNs with feedback connections between each neuron. In most cases, each neuron is connected to other neurons. Fig. 12 shows an architecture of a fully recurrent SNN.

Although fully recurrent SNNs have rich dynamic behavior, their architecture is so complex that they are difficult to analyze and train. Therefore, it is often necessary to simplify the architecture of fully recurrent SNNs in practical applications. A common simplification method is to introduce feedback connections to multilayer feed-forward SNNs. The resulting network is called a *locally recurrent SNN*, in which the primary connections are forward and a set of feedback connections are contained. According to the connection mode of feedbacks, the locally recurrent SNN can be divided into a recurrent SNN with external feedbacks and a recurrent SNN with internal feedbacks.

A recurrent SNN with external feedbacks is called a *Jordan recurrent neural network*, in which the connections between the neurons in the output layer and the hidden layer constitute the feedback loops. Fig. 13 shows an architecture of a recurrent SNN with external feedbacks.

A recurrent SNN with internal feedbacks is called a *neural Moore machine* or an *Elman recurrent neural network*, in which the output of neurons in the hidden layer is not only inputted forward to the output layer but is also inputted back to the hidden layer itself. Fig. 14 shows an architecture of a recurrent SNN with internal feedbacks.

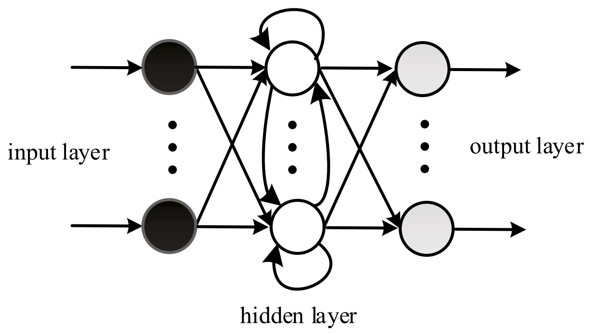


Fig. 14. Architecture of a recurrent spiking neural network with internal feedbacks.

6.2. Learning algorithms for recurrent SNNs

6.2.1. Gradient descent algorithms

Back-propagation through time (BPTT) (Werbos, 1990) is one of classical supervised learning algorithms for recurrent ANNs. In the spirit of BPTT, Tiňo and Mills (2006) extended SpikeProp (Bohte et al., 2002) to recurrent network topologies and proposed a SpikeProp through time (SPTT) supervised learning algorithm for recurrent SNNs. They turned the feed-forward SNN into a recurrent SNN by extending the feed-forward architecture with feedback connections. Similarly, Diehl, Zarrella, Cassidy, Pedroni, and Neftci (2016) presented a train-and-constrain method that trains recurrent ANNs using BPTT, then discretizes the weights, and finally converts them to recurrent SNNs by matching the responses of artificial neurons with those of the spiking neurons. Huh and Sejnowski (2018) presented another gradient descent method for optimizing recurrent SNN dynamics on the timescale of individual spikes by introducing a differentiable formulation of SNNs and deriving the exact gradient calculation. Then they applied their method to predictive coding tasks and a delayed-memory XOR task. Experimental results showed that this algorithm can optimize spiking networks to perform nonlinear computations over an extended time.

6.2.2. Kullback–Leibler divergence algorithms

On the basis of Pfister et al. (2006), Brea, Senn, and Pfister (2011, 2013) proposed a generic supervised learning rule for recurrent SNNs. The learning rule is matched to the neural dynamics, and it can be adapted to a wide range of neuron models. They first defined the Kullback–Leibler divergence from the distribution $P^{*(v)}$ of desired output spike trains to the distribution $P_w(v)$ of spike trains produced by the network:

$$\mathcal{D}(P^*(v) \parallel P_w(v)) = \left\langle \log \frac{P^*(v)}{P_w(v)} \right\rangle_{P^*(v)}. \tag{28}$$

Then they minimized the upper bound on the Kullback–Leibler divergence and derived the learning rule for synaptic weight using the gradient descent method:

$$\Delta w_{ij} \propto \left\langle \frac{\partial}{\partial w_{ij}} \log P_w(v, h) \right\rangle_{P_w(h|v)P^*(v)}. \tag{29}$$

Both offline and online learning rules were given. The derived learning rule is consistent with STDP and can be matched to the voltage-triplet rule (Clopath, Büsing, Vasilaki, & Gerstner, 2010). Rezende and Gerstner (2014), Rezende, Wierstra, and Gerstner (2011) derived a similar learning algorithm for recurrent SNNs combining principles from variational learning and reinforcement learning. The resulting online supervised learning algorithm is a Hebbian-type trace modulated by a global novelty signal. Experimental results showed that this algorithm is able to learn both stationary and nonstationary patterns of spike trains.

Table 6
Learning rates of feedback-based online local learning of weights (FOLLOW) and spike train kernel learning rule for recurrent spiking neural networks (R-STKLR) with different lengths of spike trains.

Algorithm	100 ms	200 ms	300 ms	400 ms	500 ms
FOLLOW	0.005	0.005	0.0005	0.0001	0.00005
R-STKLR	0.01	0.005	0.001	0.0005	0.0005

6.2.3. Other supervised learning algorithms

Memmesheimer, Rubin, Ölveczky, and Sompolinsky (2014) presented a simple, efficient, and biologically plausible supervised learning rule named *finite precision* that allows feed-forward and recurrent SNNs to learn multiple mappings between input and desired output spike trains. They proved that their algorithm converges after a finite number of updates. Gilra and Gerstner (2017) presented a supervised learning algorithm named *feedback-based online local learning of weights* (FOLLOW) for feed-forward and recurrent SNNs. In this learning rule, the error of the network is fed back through fixed random connections with a negative gain, causing the network to follow the desired dynamics. It is an online and local learning rule. They also proved that this algorithm converges to a stable solution. Lin and Shi (2018) extended their STKLR algorithm (Lin et al., 2016) to recurrent SNNs and proposed a new supervised multispike learning algorithm named *R-STKLR* that can implement complex spatiotemporal pattern learning of spike trains.

6.3. Performance comparison of spike train learning

In this subsection, the spike train learning performances of FOLLOW (Gilra & Gerstner, 2017) and R-STKLR (Lin & Shi, 2018) are compared. These two algorithms can achieve spike train learning. There are 50 input neurons, 30 hidden neurons, and one output neuron in the recurrent SNNs with internal feedbacks. The reference length of the spike trains is 200 ms and the reference firing rate of the spike trains is 40 Hz.

Fig. 15 shows the learning results for FOLLOW and R-STKLR with different lengths of spike trains. The length of the spike trains was increased from 100 ms to 500 ms in steps of 100 ms. The learning rates of FOLLOW and R-STKLR corresponding to the different lengths of spike trains are shown in Table 6. Fig. 15(a) shows the average learning accuracy C after 1000 learning epochs. It can be seen that the length of the spike trains has little effect on the learning performance of FOLLOW. When the length of the spike trains is short, the learning accuracy of R-STKLR is higher than that of FOLLOW. Fig. 15(b) shows the average number of learning epochs when the learning accuracy C reaches the maximum value. It can be seen that when the length of the spike trains is longer, the learning epoch of FOLLOW is greater than that of R-STKLR.

Fig. 16 shows the learning results for FOLLOW and R-STKLR with different firing rates of spike trains. The firing rate of spike trains was increased from 20 Hz to 100 Hz in steps of 20 Hz. The learning rates of FOLLOW and R-STKLR corresponding to the different firing rates of spike trains are shown in Table 7. Fig. 16(a) shows the average learning accuracy C after 1000 learning epochs. It can be seen that the learning accuracy of R-STKLR is higher than that of FOLLOW when the firing rate of spike trains is low. Fig. 16(b) shows the average number of learning epochs when the learning accuracy C reaches the maximum value. It can

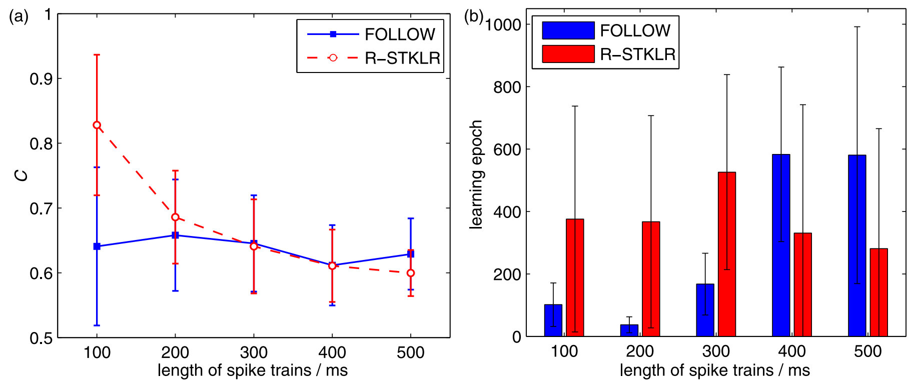


Fig. 15. Learning results with different lengths of spike trains for recurrent spiking neural networks after 1000 learning epochs. (a) The learning accuracy C. (b) The learning epochs when the learning accuracy C reaches the maximum value. FOLLOW, feedback-based online local learning of weights; R-STKLR, spike train kernel learning rule for recurrent spiking neural networks.

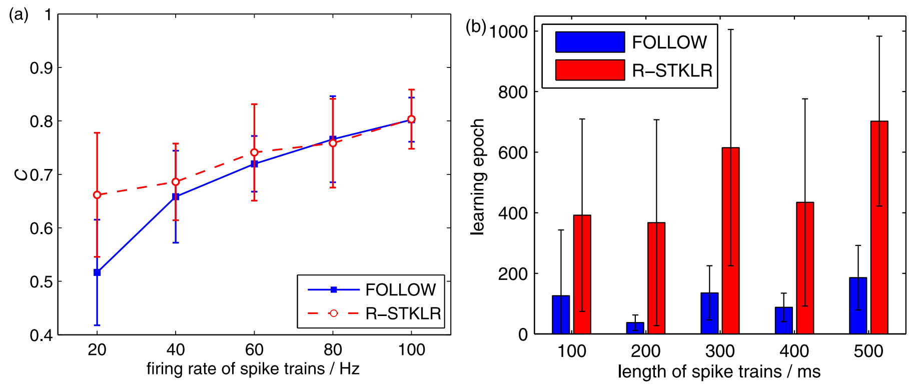


Fig. 16. Learning results with different firing rates of spike trains for recurrent spiking neural networks after 1000 learning epochs. (a) The learning accuracy C. (b) The learning epochs when the learning accuracy C reaches the maximum value. FOLLOW, feedback-based online local learning of weights; R-STKLR, spike train kernel learning rule for recurrent spiking neural networks.

Table 7
Learning rates of feedback-based online local learning of weights (FOLLOW) and spike train kernel learning rule for recurrent spiking neural networks (R-STKLR) with different firing rates of spike trains.

Algorithm	20 Hz	40 Hz	60 Hz	80 Hz	100 Hz
FOLLOW	0.005	0.005	0.001	0.001	0.0005
R-STKLR	0.005	0.005	0.0005	0.0005	0.0005

be seen that the learning epoch of FOLLOW is less than that of R-STKLR.

Fig. 17 shows the average running time of one trial of FOLLOW and R-STKLR under the reference conditions. The running times of FOLLOW and R-STKLR are not very different, but that of FOLLOW is relatively shorter.

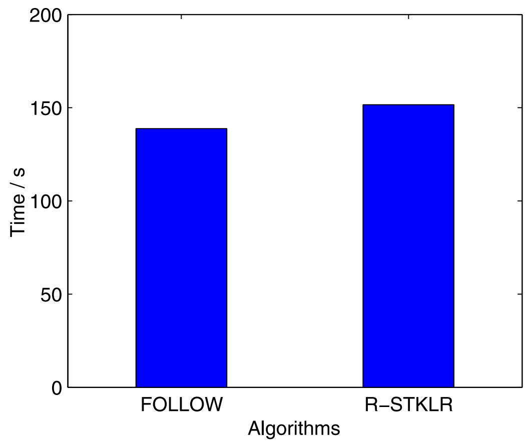


Fig. 17. Running time of one trial of feedback-based online local learning of weights (FOLLOW) and spike train kernel learning rule for recurrent spiking neural networks (R-STKLR).

7. Qualitative performance evaluation and taxonomy

7.1. Qualitative performance evaluation of supervised learning algorithms

The supervised learning algorithms for SNNs proposed in recent years exhibit different characteristics. To understand and compare different algorithms better, the performance of supervised learning algorithms for SNNs is evaluated qualitatively mainly from the following five aspects (Ponulak, 2006):

1. Learning ability of spike trains. Although the application of supervised learning algorithms is related to the spike encoding method chosen, from the performance of the supervised learning algorithm itself, some algorithms can achieve multiple-spike learning tasks, while some algorithms can achieve only single-spike learning tasks. Generally, the supervised learning algorithms based on spike train learning have greater learning ability and wider applicability but are more complex than the single-spike learning algorithms.

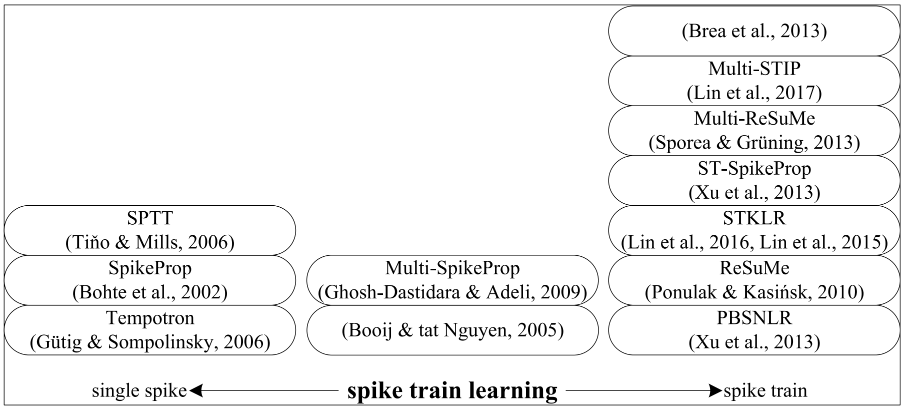

(Brea et al., 2013)
Multi-STIP (Lin et al., 2017)
Multi-ReSuMe (Sporea & Grüning, 2013)
ST-SpikeProp (Xu et al., 2013)
STKLR (Lin et al., 2016, Lin et al., 2015)
ReSuMe (Ponulak & Kasińsk, 2010)
PBSNLR (Xu et al., 2013)
SPTT (Tiňo & Mills, 2006)
SpikeProp (Bohte et al., 2002)
Tempotron (Gütig & Sompolinsky, 2006)
Multi-SpikeProp (Ghosh-Dastidara & Adeli, 2009)
(Booij & tat Nguyen, 2005)
single spike
spike train learning
spike train

Fig. 18. A taxonomy for supervised learning algorithms from the spike train learning dimension. PBSNLR, perceptron-based spiking neuron learning rule; SPTT, SpikeProp through time; STKLR, spike train kernel learning rule; ST-SpikeProp, spike train SpikeProp.

2. Offline and online processing performance. Supervised learning for SNNs generally has two running modes: offline learning and online learning (Wang, Belatreche, Maguire, & McGinnity, 2010). Offline learning methods calculate the adjustment of synaptic weights according to the entire actual and desired output spike trains after running of the network, while online learning methods adjust the synaptic weight of a neuron when the neuron fires a spike during the running process (Xu, Yang, & Zhong, 2017). Offline learning algorithms are suitable for static data processing. Spatiotemporal data are generally expressed as continuous spike train streams (Kasabov, 2014), which requires the learning algorithm to train the network in real time. Online learning algorithms are suitable and effective for dealing with such real-time tasks (Oniz & Kaynak, 2014). Online learning is a biologically plausible learning mode, so research on online learning algorithms has important significance.

   The synaptic weight is updated several times in a learning epoch for the online learning algorithm. The time-varying adjustment of the synaptic weight is usually expressed as $\Delta w(t)$. In contrast, the synaptic weight is updated only once in a learning epoch for the offline learning algorithm, in which the adjustment of the synaptic weight is usually expressed as Δw. Some algorithms can run not only in offline mode but also in online mode, while others can run only in either offline mode or online mode. For the algorithm that can run in two modes, the update of the synaptic weight for online learning and offline learning has the following relationship:

$$\Delta w = \int_{\Gamma} \Delta w(t) dt. \tag{30}$$

3. Locality property of learning rules. *Locality property* means that the learning rules are determined only by the presynaptic and postsynaptic activities of neurons, and the synaptic weight itself (Baldi & Sadowski, 2016); namely, the quantity that modifies the weight of a synapse must be available locally at the synapse (Gilra & Gerstner, 2017). It is a biologically plausible property and can be formally defined as

$$\Delta w_{ij} = F(v_i, v_j, w_{ij}), \tag{31}$$

   where v_i and v_j are activities of presynaptic and postsynaptic neurons, respectively, and $F(v_i, v_j, w_{ij})$ is a functional relationship between v_i, v_j, and w_{ij}. Supervised learning algorithms with locality property have a wider range of applications.
4. Stability of the optimal solution. Supervised learning for SNNs is a process of optimization of synaptic weights. It is expected that the global optimal solution can be obtained in every learning epoch, but not the local optimal solution. This requires the learning algorithm to possess stability of the optimal solution. Stability of the optimal solution is usually characterized by the convergence of synaptic weights. When a supervised learning algorithm for SNNs is convergent, the synaptic modifications are essentially driven by the difference $E(\boldsymbol{S}_a, \boldsymbol{S}_d)$ between the desired and the actual output spike trains. That means $\Delta \boldsymbol{W}=\boldsymbol{0}$ if and only if $E(\boldsymbol{S}_a, \boldsymbol{S}_d) = 0$. In other words, the synaptic weight matrix $\boldsymbol{W}$ reaches a fixed point if the actual output spike trains $\boldsymbol{S}_a$ equal the desired output spike trains $\boldsymbol{S}_d$. It can be shown that under certain conditions, this fixed point is a global, positive attractor in weight space.
5. Applicability of spiking neuron models. A spiking neuron is the basic computational unit of SNNs. There are many supervised learning algorithms for SNNs, especially the gradient-descent-based algorithms, which are restricted to work only with analytically tractable spiking neuron models, such as with the SRM. However, some supervised learning algorithms rely explicitly only on the spike times and do not refer to the particular properties of the spiking neuron models. It is expected that the algorithm should work correctly independently of the spiking neuron model used.

7.2. A taxonomy for supervised learning algorithms

In this subsection, we have tried to develop a principled taxonomy for supervised learning algorithms to make classification and comparison of different algorithms possible according to the following five performance evaluation criteria described in Section 7.1:

1. Spike train learning. Fig. 18 shows a taxonomy for supervised learning algorithms from the spike train learning dimension. It can be seen that some learning algorithms can achieve only single-spike learning; for example, tempotron (Gütig & Sompolinsky, 2006), SpikeProp (Bohte et al., 2002), and SPTT (Tiňo & Mills, 2006). Meanwhile, some algorithms can achieve spike train learning; for example, PBSNLR (Xu, Zeng, & Zhong, 2013), ReSuMe (Ponulak & Kasiński, 2010), STKLR (Lin, Ning, & Wang, 2015; Lin et al., 2016), ST-SpikeProp (Xu, Zeng, Han, & Yang, 2013),

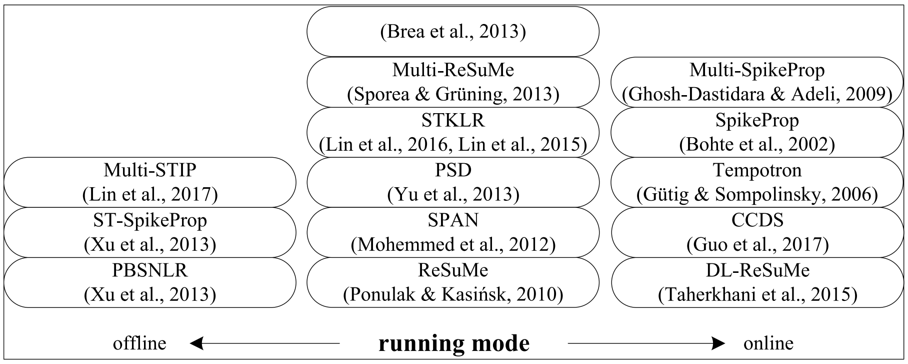

(Brea et al., 2013)
Multi-ReSuMe (Sporea & Grüning, 2013)
STKLR (Lin et al., 2016, Lin et al., 2015)
PSD (Yu et al., 2013)
SPAN (Mohemmed et al., 2012)
ReSuMe (Ponulak & Kasińsk, 2010)
Multi-STIP (Lin et al., 2017)
ST-SpikeProp (Xu et al., 2013)
PBSNLR (Xu et al., 2013)
Multi-SpikeProp (Ghosh-Dastidara & Adeli, 2009)
SpikeProp (Bohte et al., 2002)
Tempotron (Gütig & Sompolinsky, 2006)
CCDS (Guo et al., 2017)
DL-ReSuMe (Taherkhani et al., 2015)
offline
running mode
online

Fig. 19. A taxonomy for supervised learning algorithms from the running mode dimension. CCDS, cross-correlated delay shift; PBSNLR, perceptron-based spiking neuron learning rule; PSD, precise spike driven; SPAN, spike pattern association neuron; STKLR, spike train kernel learning rule; ST-SpikeProp, spike train SpikeProp.

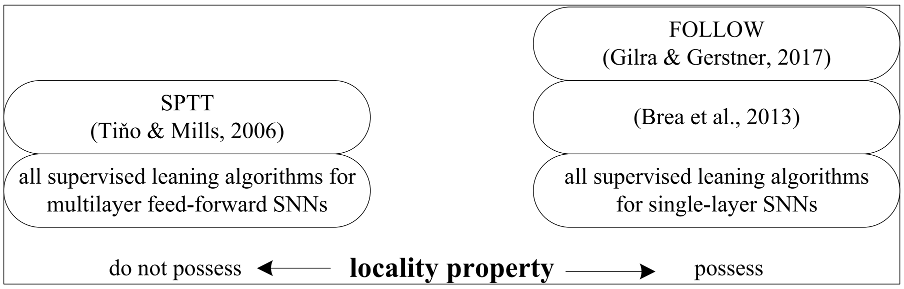

FOLLOW (Gilra & Gerstner, 2017)
SPTT (Tiňo & Mills, 2006)
(Brea et al., 2013)
all supervised leaning algorithms for multilayer feed-forward SNNs
all supervised leaning algorithms for single-layer SNNs
do not possess
locality property
possess

Fig. 20. A taxonomy for supervised learning algorithms from the locality property dimension. FOLLOW, feedback-based online local learning of weights; SNNs, spiking neural networks; SPTT, SpikeProp through time.

Multi-ReSuMe (Sporea & Grüning, 2013), Multi-STIP (Lin et al., 2017), and the algorithm presented by Brea et al. (2013). In addition, for some algorithms, the neurons in the input and hidden layers can fire multiple spikes, while neurons in the output layer can fire only a single spike; for example, the algorithm presented by Booij and tat Nguyen (2005) and Multi-SpikeProp (Ghosh-Dastidara & Adeli, 2009).

2. Running mode. Fig. 19 shows a taxonomy for supervised learning algorithms from the running mode dimension. It can be seen that some algorithms can run only in offline mode; for example, PBSNLR (Xu, Zeng, & Zhong, 2013), ST-SpikeProp (Xu, Zeng, Han, & Yang, 2013), and Multi-STIP (Lin et al., 2017). Meanwhile, some algorithms can run only in online mode; for example, DL-ReSuMe (Taherkhani et al., 2015a) and CCDS (Guo et al., 2017). Some algorithms that can achieve only single-spike learning (Bohte et al., 2002; Gütig & Sompolinsky, 2006) or output neurons that fire a single spike (Ghosh-Dastidara & Adeli, 2009) can be seen as an online running mode. In addition, some learning algorithms can run in both offline mode and online mode; for example, ReSuMe (Ponulak & Kasiński, 2010), SPAN (Mohemmed et al., 2012), PSD (Yu et al., 2013b), STKLR (Lin, Ning, & Wang, 2015; Lin et al., 2016), Multi-ReSuMe (Sporea & Grüning, 2013), and the algorithm presented by Brea et al. (2013).
3. Locality property. A supervised learning algorithm that is local means that the adjustment of the synaptic weight is available locally at the synapse. Therefore, all supervised learning algorithms for single-layer SNNs possess the locality property, while all supervised learning algorithms for multilayer feed-forward SNNs do not possess this property. Some supervised learning algorithms for recurrent SNNs possess this property, such as the algorithm presented by Brea et al. (2013) and FOLLOW (Gilra & Gerstner, 2017). However, there are still some supervised learning algorithms for recurrent SNNs that do not possess this property, such as SPTT (Tiňo & Mills, 2006). A taxonomy for supervised learning algorithms from the locality property dimension is shown in Fig. 20.
4. Stability of the optimal solution. Fig. 21 shows a taxonomy for supervised learning algorithms from the stability dimension. Some supervised learning algorithms have been proved to possess stability of the optimal solution; for example, ReSuMe (Ponulak & Kasiński, 2010), SpikeProp (Bohte et al., 2002), and FOLLOW (Gilra & Gerstner, 2017). Meanwhile, some supervised learning algorithms have been proved to not possess stability of the optimal solution; for example, the algorithm presented by Legenstein et al. (2005). However, there are still many supervised learning algorithms for which it cannot be determined whether they possess stability of the optimal solution; for example, PBSNLR (Xu, Zeng, & Zhong, 2013), STKLR (Lin, Ning, & Wang, 2015; Lin et al., 2016), ST-SpikeProp (Xu, Zeng, Han, & Yang, 2013), Multi-ReSuMe (Sporea & Grüning, 2013), Multi-STIP (Lin et al., 2017), and the algorithm presented by Brea et al. (2013).
5. Spiking neuron models. Fig. 22 shows a taxonomy for supervised learning algorithms from the spiking neuron model dimension. It can be seen that some supervised learning algorithms, especially some gradient-descent-based algorithms (Bohte et al., 2002; Booij & tat Nguyen, 2005; Ghosh-Dastidara & Adeli, 2009; Tiňo & Mills, 2006; Xu, Zeng, Han, & Yang, 2013), can use only the neurons in which the internal state can be described by an analytic expression, such as the SRM or the LIF model. Many other algorithms (Lin, Ning, & Wang, 2015; Lin et al., 2016, 2017; Mohemmed et al., 2012; Ponulak & Kasiński, 2010; Taherkhani et al., 2015a; Yu et al., 2013b) can be applied to various spiking neuron models in theory.

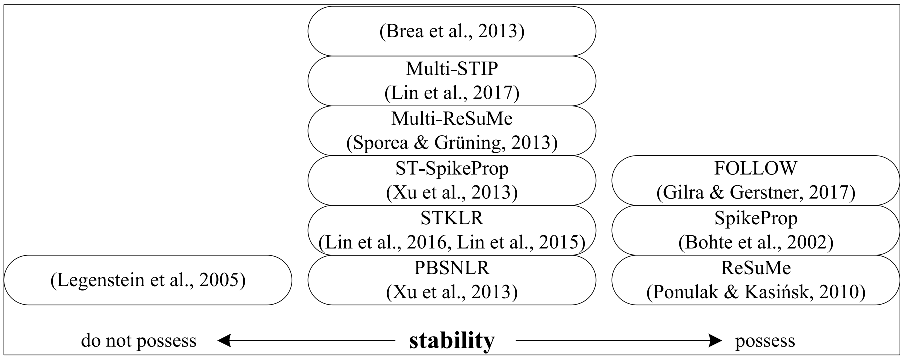

(Brea et al., 2013)
Multi-STIP (Lin et al., 2017)
Multi-ReSuMe (Sporea & Grüning, 2013)
ST-SpikeProp (Xu et al., 2013)
STKLR (Lin et al., 2016, Lin et al., 2015)
PBSNLR (Xu et al., 2013)
(Legenstein et al., 2005)
FOLLOW (Gilra & Gerstner, 2017)
SpikeProp (Bohte et al., 2002)
ReSuMe (Ponulak & Kasińsk, 2010)
do not possess
stability
possess

Fig. 21. A taxonomy for supervised learning algorithms from the stability dimension. FOLLOW, feedback-based online local learning of weights; PBSNLR, perceptron-based spiking neuron learning rule; STKLR, spike train kernel learning rule; ST-SpikeProp, spike train SpikeProp.

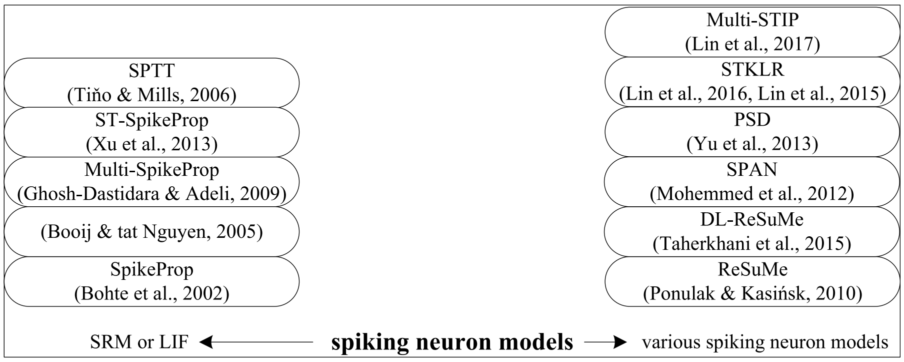

SPTT (Tiňo & Mills, 2006)
ST-SpikeProp (Xu et al., 2013)
Multi-SpikeProp (Ghosh-Dastidara & Adeli, 2009)
(Booij & tat Nguyen, 2005)
SpikeProp (Bohte et al., 2002)
Multi-STIP (Lin et al., 2017)
STKLR (Lin et al., 2016, Lin et al., 2015)
PSD (Yu et al., 2013)
SPAN (Mohemmed et al., 2012)
DL-ReSuMe (Taherkhani et al., 2015)
ReSuMe (Ponulak & Kasińsk, 2010)
SRM or LIF
spiking neuron models
various spiking neuron models

Fig. 22. A taxonomy for supervised learning algorithms from the spiking neuron model dimension. LIF, leaky integrate and fire; PSD, precise spike driven; SPAN, spike pattern association neuron; SPTT, SpikeProp through time; SRM, spike response mode; STKLR, spike train kernel learning rule; ST-SpikeProp, spike train SpikeProp.

8. Future directions

In general, supervised learning for SNNs is a significant research field. Researchers have conducted much research and achieved a series of fruitful results. However, because of the inherent complexity of SNNs, it is difficult to construct supervised learning algorithms with extensive applicability. There are many difficult problems that need to be solved, and new learning mechanisms and algorithms need to be explored. By analyzing and summarizing the current supervised learning algorithms for SNNs, we can predict the major problems that will need to be solved in the future.

8.1. Online supervised learning for real-time problems

Supervised learning algorithms for multilayer feed-forward SNNs, especially some gradient-descent-based algorithms, have achieved more research results with intensive research. However, most of these algorithms use offline processing. The spatiotemporal data acquired in the real world exhibit spatial and temporal characteristics (Sengupta, McNabb, Kasabov, & Russell, 2018). Therefore, it is necessary to develop online supervised learning algorithms in SNNs for real-time problems. At present, there are few online supervised learning algorithms for SNNs, especially gradient-descent-based supervised learning algorithms (Bohte et al., 2002; Xu et al., 2017). By defining the real-time error function of spike trains and setting the reasonable adjustment of synaptic weights, one can realize the real-time adjustment of synaptic weight in SNNs.

8.2. Supervised learning with various synaptic plasticity

Research suggests that the timing of presynaptic and postsynaptic spike trains can cause long-term potentiation or long-term depression of synapses. Induction of synaptic potentiation and depression is also accompanied by the bidirectional modification (Bear, 2003; Li, Lu, Wu, Duan, & Poo, 2004) of the overall excitability of presynaptic neurons. Synaptic potentiation or depression can be reversed and rapidly transmitted to synapses on dendrites of presynaptic neurons. This specific and fast synaptic plasticity back-propagation of presynaptic neurons is similar to the mechanism in the back-propagation algorithm. It can exist in biological neural networks and play a certain role (Lillicrap & Santoro, 2019; Whittington & Rafal, 2019). Therefore, it is of great importance to construct supervised learning algorithms using various biological synaptic plasticity mechanisms for complex SNN architectures (Kuśmierz, Isomura, & Toyoizumi, 2017; Schiess, Urbanczik, & Senn, 2016; Tavanaei & Maida, 2019; Urbanczik & Senn, 2014; Whittington & Bogacz, 2017).

8.3. Supervised learning for hybrid SNNs

Feed-forward SNNs have been extensively studied for the processing of nontemporal problems. However, feed-forward SNNs cannot solve real-world temporal tasks very well. Recurrent SNNs can represent more complex time-varying systems. It is important to construct efficient hybrid SNNs in combination with the advantages of feed-forward SNNs and recurrent SNNs. In most cases, recurrent architecture in hybrid SNNs is static. Supervised learning algorithms for hybrid SNNs with dynamic recurrent architecture are not a lot (Lin, Li, & Li, 2018). It is necessary to study the dynamic characteristics of recurrent architecture in hybrid SNNs and construct efficient supervised learning algorithms for large-scale hybrid SNNs in combination with the bidirectional synaptic plasticity mechanism (Bear, 2003; Li et al., 2004).

8.4. Supervised learning for deep SNNs

In recent years, deep learning (LeCun, Bengio, & Hinton, 2015; Schmidhuber, 2015) has shown remarkable performance in many areas of pattern recognition. Most deep neural network models use the second generation of ANNs. Some research recently attempted to take advantages of both deep learning and SNNs to develop spiking deep neural networks to achieve high performance (Illing, Gerstner, & Brea, 2019; Kheradpisheh, Ganjtabesh, Thorpe, & Masquelier, 2018; Pfeiffer & Pfeil, 2018; Tavanaei, Ghodrati, Kheradpisheh, Masquelier, & Maida, 2019). In the future, it will be necessary to construct efficient computational models combining deep learning in traditional ANNs and SNNs, such as spiking deep belief networks and spiking convolutional neural networks. Furthermore, efficient supervised learning algorithms for large-scale spiking deep neural networks should be constructed.

8.5. Hardware implementation of supervised learning

The hardware implementation of supervised learning is an important and challenging field in SNNs (Bouvier et al., 2019). There are two popular hardware architectures for neuromorphic computing (Zheng & Mazumder, 2018a, 2018b). One architecture is called a *centralized memory architecture*, which is closely related to the conventional von Neumann architecture. Synaptic weights are stored in a memory array, and they can be accessed through buses. Another architecture, the distributed memory architecture, is more related to biological neural networks. This architecture has been very popular in recent years because of many emerging memory technologies, such as memristors and phase-change memory. It is of great importance to study the hardware implementation of supervised learning for SNNs using neuromorphic devices, especially for large-scale SNNs.

8.6. Large-scale SNNs for pattern recognition

An SNN has the outstanding characteristics of high self-adaptability and fault tolerance because of its precise spike timing encoding and spike-driven nonlinear dynamics. Supervised learning in SNNs has been extensively studied for algorithm design and pattern recognition. The implementation of most of the supervised algorithms is not limited by the network scale. However, because of the complexity of temporal or spatiotemporal data in the real world, the application of large-scale SNNs for pattern recognition is not wide enough and the solution of complex problems is not accurate enough. Kasabov et al. have done much work on pattern recognition based on large-scale SNNs (Kasabov, 2018). In particular, they have developed a unifying computational framework, NeuCube (Kasabov, 2014), for dealing with spatiotemporal and spectrotemporal brain data. According to the spike-driven information processing mechanism in SNNs, it is necessary to convert the formal datasets to event-driven datasets. In the future, much effort needs to put into the development of large-scale SNNs for pattern recognition to achieve better performance in terms of both accuracy and computation cost.

9. Conclusion

Aiming at the characteristics of precise spike timing encoding and neural information processing, we have provided a comprehensive review of supervised learning algorithms for SNNs. We first provided a comparison between SNNs and traditional ANNs. The basic theories of supervised learning for SNNs were also introduced, including the general framework and some related theories of supervised learning. We then surveyed the state-of-the-art supervised learning algorithms from the perspectives of the applicability to SNN architecture and the inherent mechanisms of supervised learning algorithms. A performance comparison of spike train learning of some representative algorithms was also made. Furthermore, a qualitative performance evaluation and a taxonomy of supervised learning algorithms were provided. Finally, some future research directions in this research field were outlined.

Similarly to traditional ANNs, there are three kinds of common topological architecture for SNNs: feed-forward SNNs, recurrent SNNs, and hybrid SNNs. A single-layer SNN is a special feed-forward SNN. Because of the simplicity of its structure, a single-layer SNN is widely used in the pattern recognition field. In most cases, recurrent architecture in hybrid SNNs is static. Therefore, from the perspective of network topological architecture, we discussed the supervised learning algorithms for single-layer SNNs, multilayer feed-forward SNNs, and recurrent SNNs, respectively.

Although there are various supervised learning algorithms for SNNs, some of them have common characteristics. Some biological mechanisms and mathematical methods are commonly used to design supervised learning algorithms for SNNs. In this review, we discussed mainly supervised learning algorithms for SNNs based on the mechanisms of the perceptron, gradient descent, synaptic plasticity, spike train convolution, and Kullback–Leibler divergence. There are also some supervised learning algorithms based on other mechanisms, just like some optimization methods.

It is difficult to evaluate a supervised learning algorithm. In this review, we provided five qualitative performance evaluation criteria for supervised learning algorithms for SNNs, including the learning ability of spike trains, the offline and online processing performance, the locality property of learning rules, the stability of the optimal solution, and the applicability of spiking neuron models. Furthermore, a principled taxonomy for supervised learning algorithms was presented according to these five performance evaluation criteria. It shows that some algorithms can learn only a single spike, while some algorithms can learn spike trains; some algorithms can run with both offline and online, while some algorithms can run either only offline or only online; all algorithms for single-layer SNNs and some algorithms for recurrent SNNs possess the locality property, while all algorithms for multilayer feed-forward SNNs and some algorithms for recurrent SNNs do not possess this property; some algorithms possess stability of the optimal solution, while some algorithms do not possess this property; some algorithms (e.g., some gradient-descent-based algorithms) are restricted to working only with the SRM or the LIF model, while some algorithms are independent of the spiking neuron model.

An SNN plays a significant role in brain-inspired artificial intelligence, while supervised learning is a core research topic in SNNs. Developing high-performance supervised learning algorithms with wide applicability will help accelerate the utility process of large-scale SNNs. Many research results have been achieved, but there are still many difficult problems that need to be solved. This requires continuous and highly effective studies by researchers.

Acknowledgments

The authors would like to thank the Co-Editor in Chief, the Action Editor and the anonymous reviewers very much for their careful work and valuable constructive suggestions and comments that have helped improve the quality of this paper greatly. This work was supported by the National Natural Science Foundation of China under Grant No. 61762080, the Lanzhou Municipal Science and Technology Project, China under Grant No. 2019-1-34, and the Program for Innovative Research Team in Northwest Normal University, China under Grant No. 6008-01602.

Appendix A. Spike response model

The SRM (Gerstner & Kistler, 2002) is used in the spike train learning. Assuming that a neuron has N_I input synapses, the ith synapse transmits a total of F_i spikes, and the fth spike ($f \in [1, F_i]$) is fired at time t_i^f. The internal state $u(t)$ of the neuron at time t is given by

$$u(t) = \sum_{i=1}^{N_I} \sum_{f=1}^{F_i} w_i \varepsilon(t - t_i^f) + \eta(t - t_o^l), \tag{32}$$

where w_i is the synaptic weight for the ith synapse. The spike response function $\varepsilon(t - t_i^f)$ describes the effect of the presynaptic spike on the internal state of the postsynaptic neuron, which is expressed as

$$\varepsilon(t - t_i^f) = \begin{cases} \frac{t-t_i^f}{\tau} \exp(1 - \frac{t-t_i^f}{\tau}), & t - t_i^f > 0, \\ 0, & t - t_i^f \le 0, \end{cases} \tag{33}$$

where τ is the time decay constant of postsynaptic potentials. $\eta(t - t_o^l)$ is the refractoriness function, which is reflected mainly in the effect that only the last output spike t_o^l contributes to the refractoriness:

$$\eta(t - t_o^l) = \begin{cases} -\theta \exp(-\frac{t-t_o^l}{\tau_R}), & t - t_o^l > 0, \\ 0, & t - t_o^l \le 0, \end{cases} \tag{34}$$

where θ is the spike firing threshold and τ_R is the time constant.

The parameter values of the SRM used in the spike train learning are the time constant of the postsynaptic potential $\tau = 2$ ms, the time constant of the refractory period $\tau_R = 50$ ms, the spike firing threshold $\theta = 1$, and the length of the absolute refractory period $t_{ref} = 1$ ms.

Appendix B. Spike train learning task

The clock-driven simulation strategy with time step $dt = 0.1$ ms is used to implement the spike train learning tasks. All simulations run on NetBeans IDE 8.2 with Java 1.8 on a six-core system with 16-GB RAM in a Windows 10 environment. Initially, the synaptic weights are generated randomly as the uniform distribution in the interval $[0, 0.2]$. Every input spike train and desired output spike train is generated randomly by a homogeneous Poisson process within the time interval of Γ with the same firing rate. There is only one weighted connection between presynaptic and postsynaptic neurons. All neurons are assumed to be excitatory. The parameters of the learning algorithms themselves are consistent with those in the corresponding references. All simulation results are averaged over 20 trials, and on each testing trial, the learning algorithm is applied for a maximum of 1000 learning epochs or until the network error $E = 0$.

To quantitatively evaluate the learning performance, the spike train kernel is used to define a correlation-based metric (Schreiber, Fellous, Whitmer, Tiesinga, & Sejnowski, 2003) C to express the distance between the desired output spike train $s_d^o(t)$ and the actual output spike train $s_a^o(t)$. The metric C is calculated after each learning epoch according to

$$C = \frac{\langle f_{s_d^o}(t), f_{s_a^o}(t) \rangle}{\| f_{s_d^o}(t) \| \, \| f_{s_a^o}(t) \|}, \tag{35}$$

where $f_{s_d^o}(t)$ and $f_{s_a^o}(t)$ are continuous functions convolved by a Gaussian filter function corresponding to spike trains $s_d^o(t)$ and $s_a^o(t)$, respectively. $\langle f_{s_d^o}(t), f_{s_a^o}(t) \rangle$ is the inner product of $f_{s_d^o}(t)$ and $f_{s_a^o}(t)$. $\| f_{s_d^o}(t) \| = \sqrt{\langle f_{s_d^o}(t), f_{s_d^o}(t) \rangle}$ and $\| f_{s_a^o}(t) \| = \sqrt{\langle f_{s_a^o}(t), f_{s_a^o}(t) \rangle}$ are the Euclidean norms of $f_{s_d^o}(t)$ and $f_{s_a^o}(t)$, respectively. $C = 1$ for identical spike trains and decreases toward 0 for loosely correlated spike trains.

References

Abusnaina, A. A., Abdullah, R., & Kattan, A. (2019). Supervised training of spiking neural network by adapting the E-MWO algorithm for pattern classification. *Neural Processing Letters*, *49*(2), 661–682.

Almási, A. D., Woźniak, S., Cristea, V., Leblebici, Y., & Engbersen, T. (2016). Review of advances in neural networks: Neural design technology stack. *Neurocomputing*, *174*, 31–41.

Amit, D. J., & Brunel, N. (1997). Dynamics of a recurrent network of spiking neurons before and following learning. *Network. Computation in Neural Systems*, *8*(4), 373–404.

Baldi, P., & Sadowski, P. (2016). A theory of local learning, the learning channel, and the optimality of backpropagation. *Neural Networks*, *83*, 51–74.

Banerjee, A. (2016). Learning precise spike train-to-spike train transformations in multilayer feedforward neuronal networks. *Neural Computation*, *28*(5), 826–848.

Bear, M. F. (2003). Bidirectional synaptic plasticity: From theory to reality. *Philosophical Transactions of the Royal Society, Series B (Biological Sciences)*, *358*(1432), 649–655.

Belatreche, A., Maguire, L. P., & McGinnity, M. (2007). Advances in design and application of spiking neural networks. *Soft Computing*, *11*(3), 239–248.

Belatreche, A., Maguire, L. P., McGinnity, M., & Wu, Q. (2003). An evolutionary strategy for supervised training of biologically plausible neural networks. In *International conference on computational intelligence and natural computing* (pp. 1524–1527).

Belatreche, A., Maguire, L. P., McGinnity, M., & Wu, Q. (2006). Evolutionary design of spiking neural networks. *New Mathematics and Natural Computation*, *2*(3), 237–253.

Beyeler, M., Dutt, N. D., & Krichmar, J. L. (2013). Categorization and decision-making in a neurobiologically plausible spiking network using a STDP-like learning rule. *Neural Networks*, *48*, 109–124.

Bienenstock, E. L., Cooperk, L. N., & Munro, P. W. (1982). Theory for the development of neuron selectivity: Orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience*, *2*(1), 32–48.

Bing, Z., Baumann, I., Jiang, Z., Huang, K., Cai, C., & Knoll, A. (2019). Supervised learning in SNN via reward-modulated spike-timing-dependent plasticity for a target reaching vehicle. *Frontiers in Neurorobotics*, *13*, 18.

Bohte, S. M. (2004). The evidence for neural information processing with precise spike-times: A survey. *Natural Computing*, *3*(2), 195–206.

Bohte, S. M., Kok, J. N., & Poutré, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, *48*(1), 17–37.

Booij, O., & tat Nguyen, H. (2005). A gradient descent rule for spiking neurons emitting multiple spikes. *Information Processing Letters*, *95*(6), 552–558.

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

Bouvier, M., Valentian, A., Mesquida, T., Rummens, F., Reyboz, M., Vianello, E., et al. (2019). Spiking neural networks hardware implementations and challenges: A survey. *ACM Journal on Emerging Technologies in Computing Systems*, *15*(2), 22.

Brea, J., Senn, W., & Pfister, J. P. (2011). Sequence learning with hidden units in spiking neural networks. In *Advances in neural information processing systems* (pp. 1422–1430). Neural Information Processing Systems Foundation.

Brea, J., Senn, W., & Pfister, J. P. (2013). Matching recall and storage in sequence learning with spiking neural networks. *Journal of Neuroscience*, *33*(23), 9565–9575.

Brette, R., Rudolph, M., Carnevale, T., et al. (2007). Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience*, *23*(3), 349–398.

Brockmeier, A. J., Choi, J. S., Kriminger, E. G., Francis, J. T., & Principe, J. C. (2014). Neural decoding with kernel-based metric learning. *Neural Computation*, *26*(6), 1080–1107.

Buesing, L., Bill, J., Nessler, B., & Maass, W. (2011). Neural dynamics as sampling: A model for stochastic computation in recurrent networks of spiking neurons. *PLoS Computational Biology*, *7*(11), e1002211.
Burkitt, A. N. (2006a). A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input. *Biological Cybernetics*, *95*(1), 1–19.
Burkitt, A. N. (2006b). A review of the integrate-and-fire neuron model: II. Inhomogeneous synaptic input and network properties. *Biological Cybernetics*, *95*(2), 97–112.
Caporale, N., & Dan, Y. (2008). Spike timing-dependent plasticity: A Hebbian learning rule. *Annual Review of Neuroscience*, *31*, 25–46.
Carnell, A., & Richardson, D. (2005). Linear algebra for times series of spikes. In *European symposium on artificial neural networks* (pp. 363–368).
Chauvin, Y., & Rumelhart, D. E. (1995). *Backpropagation: Theory, architectures, and applications*. Psychology Press.
Chen, Z. (2013). An overview of bayesian methods for neural spike train analysis. *Computational Intelligence and Neuroscience*, *2013*, 251905.
Clopath, C., Büsing, L., Vasilaki, E., & Gerstner, W. (2010). Connectivity reflects coding: A model of voltage-based STDP with homeostasis. *Nature Neuroscience*, *13*(3), 344–352.
Dauwels, J., Vialatte, F., Weber, T., & Cichocki, A. (2008). On similarity measures for spike trains. In *International conference on neural information processing* (pp. 177–185). Springer.
Denéve, S., Alemi, A., & Bourdoukan, R. (2017). The brain as an efficient and robust adaptive learner. *Neuron*, *94*(5), 969–977.
D'Haene, M., Hermans, M., & Schrauwen, B. (2014). Toward unified hybrid simulation techniques for spiking neural networks. *Neural Computation*, *26*(6), 1055–1079.
Dhoble, K., Nuntalid, N., Indiveri, G., & Kasabov, N. (2012). Online spatio-temporal pattern recognition with evolving spiking neural networks utilising address event representation, rank order, and temporal spike learning. In *International joint conference on neural networks* (pp. 1–7). IEEE.
Diehl, P. U., Zarrella, G., Cassidy, A., Pedroni, B. U., & Neftci, E. (2016). Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. In *IEEE international conference on rebooting computing* (pp. 1–8). IEEE.
Dora, S., Sundaram, S., & Sundararajan, N. (2015). A two stage learning algorithm for a growing-pruning spiking neural network for pattern classification problems. In *International joint conference on neural networks* (pp. 1–7). IEEE.
Dora, S., Suresh, S., & Sundararajan, N. (2014). A sequential learning algorithm for a minimal spiking neural network (MSNN) classifier. In *International joint conference on neural networks* (pp. 2415–2421). IEEE.
Dora, S., Suresh, S., & Sundararajan, N. (2015). A sequential learning algorithm for a spiking neural classifier. *Applied Soft Computing*, *36*, 255–268.
Dorogyy, Y., & Kolisnichenko, V. (2016). Designing spiking neural networks. In *International conference on modern problems of radio engineering, telecommunications and computer science* (pp. 124–127). IEEE.
Du, Z., Rubin, D. D. B.-D., Chen, Y., He, L., Chen, T., Zhang, L., et al. (2015). Neuromorphic accelerators: A comparison between neuroscience and machine-learning approaches. In *International symposium on microarchitecture* (pp. 494–507). IEEE.
Fang, H., & Wang, Y. (2008). Spiking neural networks with neurons firing multiple spikes. *Journal of Applied Sciences*, *26*(6), 638–644.
Farsa, E. Z., Ahmadi, A., Maleki, M. A., Gholami, M., & Rad, H. N. (2019). A low-cost high-speed neuromorphic hardware based on spiking neural network. *IEEE Transactions on Circuits and Systems II: Express Briefs*, *66*(9), 1582–1586.
Feldman, D. E. (2012). The spike-timing dependence of plasticity. *Neuron*, *75*(4), 556–571.
Franosch, J.-M. P., Urban, S., & van Hemmen, J. L. (2013). Supervised spike-timing-dependent plasticity: A spatiotemporal neuronal learning rule for function approximation and decisions. *Neural Computation*, *25*(12), 3113–3130.
Frémaux, N., Sprekeler, H., & Gerstner, W. (2010). Functional requirements for reward-modulated spike-timing-dependent plasticity. *Journal of Neuroscience*, *30*(40), 13326–13337.
Fusi, S., Annunziato, M., Badoni, D., Salamon, A., & Amit, D. J. (2000). Spike-driven synaptic plasticity: Theory, simulation, VLSI implementation. *Neural Computation*, *12*(10), 2227–2258.
Gardner, B., & Grüning, A. (2013). Learning temporally precise spiking patterns through reward modulated spike-timing-dependent plasticity. In *International conference on artificial neural networks* (pp. 256–263). Springer.
Gardner, B., & Grüning, A. (2016). Supervised learning in spiking neural networks for precise temporal encoding. *PLoS One*, *11*(8), e0161335.
Gardner, B., Sporea, I., & Grüning, A. (2015). Learning spatiotemporally encoded pattern transformations in structured spiking neural networks. *Neural Computation*, *27*(12), 2548–2586.
Georgopoulos, A. P., Schwartz, A. B., & Kettner, R. E. (1986). Neuronal population coding of movement direction. *Science*, *233*(4771), 1416–1419.
Gerstner, W. (2001). A framework for spiking neuron models: The spike response model. In *Handbook of biological physics (vol. 4)* (pp. 469–516). Elsevier.
Gerstner, W., & Kistler, W. M. (2002). *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge University Press.
Ghosh-Dastidar, S., & Adeli, H. (2009). Spiking neural networks. *International Journal of Neural Systems*, *19*(4), 295–308.
Ghosh-Dastidara, S., & Adeli, H. (2009). A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection. *Neural Networks*, *22*(10), 1419–1431.
Gilra, A., & Gerstner, W. (2017). Predicting non-linear dynamics by stable local learning in a recurrent spiking neural network. *eLife*, *6*, e28295.
Glackin, C., Maguire, L., McDaid, L., & Sayers, H. (2011). Receptive field optimisation and supervision of a fuzzy spiking neural network. *Neural Networks*, *24*(3), 247–256.
Glasera, J. I., Benjamina, A. S., Farhoodia, R., & Kordinga, K. P. (2019). The roles of supervised machine learning in systems neuroscience. *Progress in Neurobiology*, *175*, 126–137.
Gong, Z. (2013). *Multi-spike timing error backpropagation algorithm in spiking neural networks* (Master's thesis), Northwest Normal University.
Guo, L., Wang, Z., Cabrerizo, M., & Adjouadi, M. (2017). A cross-correlated delay shift supervised learning method for spiking neurons with application to interictal spike detection in epilepsy. *International Journal of Neural Systems*, *27*(3), 1750002.
Gütig, R. (2014). To spike, or when to spike? *Current Opinion in Neurobiology*, *25*, 134–139.
Gütig, R., & Sompolinsky, H. (2006). The tempotron: A neuron that learns spike timing based decisions. *Nature Neuroscience*, *9*(3), 420–428.
Haykin, S. S. (2009). *Neural networks and learning machines*. Pearson Education Upper Saddle River.
Hebb, D. O. (2005). *The organization of behavior: A neuropsychological theory*. Psychology Press.
Henker, S., Partzsch, J., & Schüffny, R. (2012). Accuracy evaluation of numerical methods used in state-of-the-art simulators for spiking neural networks. *Journal of Computational Neuroscience*, *32*(2), 309–326.
Hodgkin, A. L., & Huxley, A. F. (1952a). Currents carried by sodium and potassium ions through the membrane of the giant axon of Loligo. *The Journal of Physiology*, *116*(4), 449–472.
Hodgkin, A. L., & Huxley, A. F. (1952b). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, *117*(4), 500–544.
Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, *79*(8), 2554–2558.
Hu, J., Tang, H., Tan, K. C., Li, H., & Shi, L. (2013). A spike-timing-based integrated model for pattern recognition. *Neural Computation*, *25*(2), 450–472.
Huh, D., & Sejnowski, T. J. (2018). Gradient descent for spiking neural networks. In *Advances in neural information processing systems* (pp. 1438–1448). Neural Information Processing Systems Foundation.
Illing, B., Gerstner, W., & Brea, J. (2019). Biologically plausible deep learning but how far can we go with shallow networks? *Neural Networks*, *118*, 90–101.
Izhikevich, E. M. (2004). Which model to use for cortical spiking neurons. *IEEE Transactions on Neural Networks*, *15*(5), 1063–1070.
Izhikevich, E. M., & Desai, N. S. (2003). Relating STDP to BCM. *Neural Computation*, *15*(7), 1511–1523.
Jeyasothy, A., Sundaram, S., & Sundararajan, N. (2019). SEFRON: A new spiking neuron model with time-varying synaptic efficacy function for pattern classification. *IEEE Transactions on Neural Networks and Learning Systems*, *30*(4), 1231–1240.
Kaabi, M. G., Tonnelier, A., & Martinez, D. (2011). On the performance of voltage stepping for the simulation of adaptive, nonlinear integrate-and-fire neuronal networks. *Neural Computation*, *23*(5), 1187–1204.
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.
Kasabov, N. (2018). *Time-space, spiking neural networks and brain-inspired artificial intelligence*. Springer.
Kasabov, N., Dhoble, K., Nuntalid, N., & Indiveri, G. (2013). Dynamic evolving spiking neural networks for on-line spatio-and spectro-temporal pattern recognition. *Neural Networks*, *41*, 188–201.
Kasiński, A., & Ponulak, F. (2006). Comparison of supervised learning methods for spike time coding in spiking neural networks. *International Journal of Applied Mathematics and Computer Science*, *16*(1), 101–113.
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.
Kiselev, M. (2017). A synaptic plasticity rule providing a unified approach to supervised and unsupervised learning. In *International joint conference on neural networks* (pp. 3806–3813). IEEE.
Knudsen, E. I. (1994). Supervised learning in the brain. *Journal of Neuroscience*, *14*(7), 3985–3997.
Kreuz, T., Haas, J. S., Morelli, A., Abarbanel, H. D. I., & Politi, A. (2007). Measuring spike train synchrony. *Journal of Neuroscience Methods*, *165*(1), 151–161.
Kulkarni, S. R., & Rajendran, B. (2018). Spiking neural networks for handwritten digit recognition - supervised learning and network optimization. *Neural Networks*, *103*, 118–127.

Kuriscak, E., Marsalek, P., Stroffek, J., & Toth, P. G. (2015). Biological context of Hebb learning in artificial neural networks, a review. *Neurocomputing*, *152*, 27–35.

Kuśmierz, L., Isomura, T., & Toyoizumi, T. (2017). Learning with three factors: Modulating hebbian plasticity with errors. *Current Opinion in Neurobiology*, *46*, 170–177.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.

Lee, W. W., Kukreja, S. L., & Thakor, N. V. (2017). CONE: Convex-optimized-synaptic efficacies for temporally precise spike mapping. *IEEE Transactions on Neural Networks and Learning Systems*, *28*(4), 849–861.

Legenstein, R., Naeger, C., & Maass, W. (2005). What can a neuron learn with spike-timing-dependent plasticity. *Neural Computation*, *17*(11), 2337–2382.

Legenstein, R., Pecevski, D., & Maass, W. (2008a). A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Computational Biology*, *4*(10), e1000180.

Legenstein, R., Pecevski, D., & Maass, W. (2008b). Theoretical analysis of learning with reward-modulated spike-timing-dependent plasticity. In *Advances in neural information processing systems* (pp. 881–888). Neural Information Processing Systems Foundation.

Leon, S. J., Björck, Å., & Gander, W. (2013). Gram–Schmidt orthogonalization: 100 years and more. *Numerical Linear Algebra with Applications*, *20*(3), 492–532.

Li, C., Lu, J., Wu, C., Duan, S., & Poo, M. (2004). Bidirectional modification of presynaptic neuronal excitability accompanying spike timing-dependent synaptic plasticity. *Neuron*, *41*(2), 257–268.

Lillicrap, T. P., & Santoro, A. (2019). Backpropagation through time and the brain. *Current Opinion in Neurobiology*, *55*, 82–89.

Lin, P., Chang, S., Wang, H., Huang, Q., & He, J. (2018). SpikeCD: A parameter-insensitive spiking neural network with clustering degeneracy strategy. *Neural Computing and Applications*, 1–13.

Lin, X., Chen, G., Wang, X., & Ma, H. (2016). An improved supervised learning algorithm using triplet-based spike-timing-dependent plasticity. In *International conference on intelligent computing* (pp. 44–53). Springer.

Lin, X., & Gong, Z. (2011). Survey on modeling methods for single compartmental spiking neuron. *Computer Engineering and Applications*, *47*(35), 41–44.

Lin, X., Li, Q., & Li, D. (2018). Supervised learning algorithm for multi-spike liquid state machines. In *International conference on intelligent computing* (pp. 243–253). Springer.

Lin, Z., Ma, D., Meng, J., & Chen, L. (2018). Relative ordering learning in spiking neural network for pattern recognition. *Neurocomputing*, *275*, 94–106.

Lin, X., Ning, Z., & Wang, X. (2015). An online supervised learning algorithm based on nonlinear spike train kernels. In *International conference on intelligent computing* (pp. 106–115). Springer.

Lin, X., & Shi, G. (2018). A supervised multi-spike learning algorithm for recurrent spiking neural networks. In *International conference on artificial neural networks* (pp. 222–234). Springer.

Lin, X., Wang, X., & Dang, X. (2016). A new supervised learning algorithm for spiking neurons based on spike train kernels. *Acta Electronica Sinica*, *44*(12), 2877–2886.

Lin, X., Wang, X., & Hao, Z. (2017). Supervised learning in multilayer spiking neural networks with inner products of spike trains. *Neurocomputing*, *237*, 59–70.

Lin, X., Wang, X., Zhang, N., & Ma, H. (2015). Supervised learning algorithms for spiking neural networks: A review. *Acta Electronica Sinica*, *43*(3), 577–586.

Lin, X., & Zhang, T. (2009). Dynamical properties of piecewise linear spiking neuron model. *Acta Electronica Sinica*, *37*(6), 1270–1276.

Lobo, J. L., Del Ser, J., Bifet, A., & Kasabov, N. (2020). Spiking neural networks and online learning: An overview and perspectives. *Neural Networks*, *121*, 88–100.

Luo, Y., Fu, Q., Liu, J., Harkin, J., McDaid, L., & Cao, Y. (2017). An extended algorithm using adaptation of momentum and learning rate for spiking neurons emitting multiple spikes. In *International work-conference on artificial neural networks* (pp. 569–579). Springer.

Luo, Y., Fu, Q., Liu, J., Huang, Y., Ding, X., & Cao, Y. (2018). Improving the stability for spiking neural networks using anti-noise learning rule. In *Pacific rim international conference on artificial intelligence* (pp. 29–37). Springer.

Luo, X., Qu, H., Zhang, Y., & Chen, Y. (2019). First error-based supervised learning algorithm for spiking neural networks. *Frontiers in Neuroscience*, *13*, 559.

Ma, Q., Lin, X., & Wang, X. (2018). Supervised learning of single-layer spiking neural networks for image classification. In *International conference on artificial intelligence applications and technologies (vol. 435)* (p. 012049). IOP Publishing.

Maass, W. (1996). Lower bounds for the computational power of networks of spiking neurons. *Neural Computation*, *8*(1), 1–40.

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

Maass, W. (1997b). Fast sigmoidal networks via spiking neurons. *Neural Computation*, *9*(2), 279–304.

Maass, W., & Natschläger, T. (1997). Networks of spiking neurons can emulate arbitrary hopfield nets in temporal coding. *Network. Computation in Neural Systems*, *8*(4), 355–371.

Matsuda, S. (2016a). BPSpike: A backpropagation learning for all parameters in spiking neural networks with multiple layers and multiple spikes. In *International joint conference on neural networks* (pp. 293–298). IEEE.

Matsuda, S. (2016b). BPSpike II: A new backpropagation learning algorithm for spiking neural networks. In *International conference on neural information processing* (pp. 56–65). Springer.

Mattia, M., & Del Giudice, P. (2000). Efficient event-driven simulation of large networks of spiking neurons and dynamical synapses. *Neural Computation*, *12*(10), 2305–2329.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*(4), 115–133.

McKennoch, S., Liu, D., & Bushnell, L. G. (2006). Fast modifications of the SpikeProp algorithm. In *International joint conference on neural networks* (pp. 3970–3977). IEEE.

McKennoch, S., Voegtlin, T., & Bushnell, L. (2009). Spike-timing error backpropagation in theta neuron networks. *Neural Computation*, *21*(1), 9–45.

Memmesheimer, R. M., Rubin, R., Ölveczky, B. P., & Sompolinsky, H. (2014). Learning precisely timed spikes. *Neuron*, *82*(4), 925–938.

Minsky, M., & Papert, S. (1969). *Perceptrons*. MIT Press.

Mohemmed, A., & Kasabov, N. (2012). Incremental learning algorithm for spatio-temporal spike pattern classification. In *International joint conference on neural networks* (pp. 1–6). IEEE.

Mohemmed, A., Matsuda, S., Schliebs, S., Dhoble, K., & Kasabov, N. (2011). Optimization of spiking neural networks with dynamic synapses for spike sequence generation using PSO. In *International joint conference on neural networks* (pp. 2969–2974). IEEE.

Mohemmed, A., Schliebs, S., Matsuda, S., & Kasabov, N. (2012). SPAN: Spike pattern association neuron for learning spatio-temporal spike patterns. *International Journal of Neural Systems*, *22*(4), 1250012.

Mohemmed, A., Schliebs, S., Matsuda, S., & Kasabov, N. (2013). Training spiking neural networks to associate spatio-temporal input output spike patterns. *Neurocomputing*, *107*, 3–10.

Mostafa, H. (2018). Supervised learning based on temporal coding in spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(7), 3227–3235.

Nadasdy, Z. (2009). Information encoding and reconstruction from the phase of action potentials. *Frontiers in Systems Neuroscience*, *3*, 6.

Naud, R., Gerhard, F., Mensi, S., & Gerstner, W. (2011). Improved similarity measures for small sets of spike trains. *Neural Computation*, *23*(12), 3016–3069.

Oniz, Y., & Kaynak, O. (2014). Variable-structure-systems based approach for online learning of spiking neural networks and its experimental evaluation. *Journal of the Franklin Institute*, *351*(6), 3269–3285.

Paiva, A. R. C., Park, I., & Príncipe, J. C. (2009). A reproducing kernel Hilbert space framework for spike train signal processing. *Neural Computation*, *21*(2), 424–449.

Park, I. M., Seth, S., Paiva, A. R. C., Li, L., & Principe, J. C. (2013). Kernel methods on spike train space for neuroscience: A tutorial. *IEEE Signal Processing Magazine*, *30*(4), 149–160.

Park, I. M., Seth, S., Rao, M., & Príncipe, J. C. (2012). Strictly positive-definite spike train kernels for point-process divergences. *Neural Computation*, *24*(8), 2223–2250.

Pavlidis, N. G., Tasoulis, O. K., Plagianakos, V. P., Nikiforidis, G., & Vrahatis, M. N. (2005). Spiking neural network training using evolutionary algorithms. In *IEEE international jiont conference on neural networks (vol. 4)* (pp. 2190–2194). IEEE.

Peng, X., Wang, Z., Han, F., Song, G., & Ding, S. (2018). A novel time-event-driven algorithm for simulating spiking neural networks based on circular array. *Neurocomputing*, *292*, 121–129.

Petro, B., Kasabov, N., & Kiss, R. M. (2020). Selection and optimization of temporal spike encoding methods for spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems 31*(2), 358–370.

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

Pfister, J. P., & Gerstner, W. (2006). Triplets of spikes in a model of spike timing-dependent plasticity. *Journal of Neuroscience*, *26*(38), 9673–9682.

Pfister, J. P., Toyoizumi, T., Barber, D., & Gerstner, W. (2006). Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning. *Neural Computation*, *18*(6), 1318–1348.

Ponulak, F. (2006). *Supervised learning in spiking neural networks with ReSuMe method* (Ph.D. thesis), Poznan University of Technology.

Ponulak, F., & Kasiński, A. (2008). Analysis of the ReSuMe learning process for spiking neural networks. *International Journal of Applied Mathematics and Computer Science*, *18*(2), 117–127.

Ponulak, F., & Kasiński, A. (2010). Supervised learning in spiking neural networks with ReSuMe: Sequence learning, classification, and spike shifting. *Neural Computation*, *22*(2), 467–510.

Prieto, A., Prieto, B., Ortigosa, E. M., Ros, E., Pelayo, F., Ortega, J., et al. (2016). Neural networks: An overview of early research, current frameworks and new challenges. *Neurocomputing*, *214*, 242–268.

Pyle, R., & Rosenbaum, R. (2017). Spatiotemporal dynamics and reliable computations in recurrent spiking neural networks. *Physical Review Letters*, *118*(1), 018103.

Qu, H., Xie, X., Liu, Y., Zhang, M., & Lu, L. (2015). Improved perception-based spiking neuron learning rule for real-time user authentication. *Neurocomputing*, *151*, 310–318.

Quiroga, R. Q., & Panzeri, S. (2013). *Principles of neural coding*. CRC Press.

Rezende, D. J., & Gerstner, W. (2014). Stochastic variational learning in recurrent spiking networks. *Frontiers in Computational Neuroscience*, *8*, 38.

Rezende, D. J., Wierstra, D., & Gerstner, W. (2011). Variational learning for recurrent spiking networks. In *Advances in neural information processing systems* (pp. 136–144). Neural Information Processing Systems Foundation.

Ros, E., Carrillo, R., Ortigosa, E. M., Barbour, B., & Agís, R. (2006). Event-driven simulation scheme for spiking neural networks using lookup tables to characterize neuronal dynamics. *Neural Computation*, *18*(12), 2959–2993.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386–408.

Roy, S., San, P. P., Hussain, S., Wei, L. W., & Basu, A. (2016). Learning spike time codes through morphological learning with binary synapses. *IEEE Transactions on Neural Networks and Learning Systems*, *27*(7), 1572–1577.

Ruf, B., & Schmitt, M. (1997). Learning temporally encoded patterns in networks of spiking neurons. *Neural Processing Letters*, *5*(1), 9–18.

Rummelhart, D. E. (1986). Learning representations by back-propagating errors. *Nature*, *323*(9), 533–536.

Schæfer, M., Schœnauer, T., Wolff, C., Hartmann, G., Klar, H., & Rückert, U. (2002). Simulation of spiking neural networks - architectures and implementations. *Neurocomputing*, *48*(1), 647–679.

Schiess, M., Urbanczik, R., & Senn, W. (2016). Somato-dendritic synaptic plasticity and error-backpropagation in active dendrites. *PLoS Computational Biology*, *12*(2), e1004638.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85–117.

Schrauwen, B., & Van Campenhout, J. (2003). BSA, a fast and accurate spike train encoding scheme. In *International joint conference on neural networks (vol. 4)* (pp. 2825–2830). IEEE.

Schrauwen, B., & Van Campenhout, J. (2004a). Extending SpikeProp. In *International joint conference on neural networks (vol. 1)* (pp. 471–475). IEEE.

Schrauwen, B., & Van Campenhout, J. (2004b). Improving SpikeProp: Enhancements to an error-backpropagation rule for spiking neural networks. In *Proceedings of the 15th ProRISC workshop (vol. 11)* (pp. 301–305).

Schrauwen, B., & Van Campenhout, J. (2006). Backpropagation for population-temporal coded spiking neural networks. In *International joint conference on neural networks* (pp. 1797–1804). IEEE.

Schreiber, S., Fellous, J. M., Whitmer, D., Tiesinga, P., & Sejnowski, T. J. (2003). A new correlation-based measure of spike timing reliability. *Neurocomputing*, *52*, 925–931.

Sengupta, N., McNabb, C. B., Kasabov, N., & Russell, B. R. (2018). Integrating space, time, and orientation in spiking neural networks: A case study on multimodal brain data modeling. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(11), 5249–5263.

Shrestha, S. B., & Song, Q. (2015). Adaptive learning rate of SpikeProp based on weight convergence analysis. *Neural Networks*, *63*, 185–198.

Shrestha, S. B., & Song, Q. (2016). Adaptive delay learning in SpikeProp based on delay convergence analysis. In *International joint conference on neural networks* (pp. 277–284). IEEE.

Shrestha, S. B., & Song, Q. (2017a). Robust spike-train learning in spike-event based weight update. *Neural Networks*, *86*, 54–68.

Shrestha, S. B., & Song, Q. (2017b). Robust learning in SpikeProp. *Neural Networks*, *96*, 33–46.

Shrestha, S. B., & Song, Q. (2018). Robustness to training disturbances in SpikeProp learning. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(7), 3126–3139.

Soula, H., Beslon, G., & Mazet, O. (2006). Spontaneous dynamics of asymmetric random recurrent spiking neural networks. *Neural Computation*, *18*(1), 60–79.

Sporea, I., & Grüning, A. (2013). Supervised learning in multilayer spiking neural networks. *Neural Computation*, *25*(2), 473–509.

Strain, T. J., McDaid, L. J., McGinnity, T. M., Maguire, L. P., & Sayers, H. M. (2010). An STDP training algorithm for a spiking neural network with dynamic threshold neurons. *International Journal of Neural Systems*, *20*(6), 463–480.

Taherkhani, A., Belatreche, A., Li, Y., & Maguire, L. P. (2015a). DL-ReSuMe: A delay learning-based remote supervised method for spiking neurons. *IEEE Transactions on Neural Networks and Learning Systems*, *26*(12), 3137–3149.

Taherkhani, A., Belatreche, A., Li, Y., & Maguire, L. P. (2015b). Multi-DL-ReSuMe: Multiple neurons delay learning remote supervised method. In *International joint conference on neural networks* (pp. 1–7). IEEE.

Taherkhani, A., Belatreche, A., Li, Y., & Maguire, L. P. (2018). A supervised learning algorithm for learning precise timing of multiple spikes in multilayer spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(11), 5394–5407.

Taherkhani, A., Cosma, G., & McGinnity, T. M. (2019). Optimization of output spike train encoding for a spiking neuron based on its spatiotemporal input pattern. *IEEE Transactions on Cognitive and Developmental Systems*

Takuya, T., Haruhiko, T., Hiroharu, K., & Shinji, T. (2016). A training algorithm for spike sequence in spiking neural networks — a discussion on growing network for stable training performance. In *International conference on natural computation, fuzzy systems and knowledge discovery* (pp. 1773–1777). IEEE.

Tapson, J. C., Cohen, G. K., Afshar, S., Stiefel, K. M., Buskila, Y., Wang, R. M., et al. (2013). Synthesis of neural networks for spatio-temporal spike pattern recognition and processing. *Frontiers in Neuroscience*, *7*, 153.

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., & Maida, A. (2019). Deep learning in spiking neural networks. *Neural Networks*, *111*, 47–63.

Tavanaei, A., & Maida, A. S. (2019). BP-STDP: Approximating backpropagation using spike timing dependent plasticity. *Neurocomputing*, *330*, 39–47.

Thiruvarudchelvan, V., Moore, W., & Antolovich, M. (2013). Improving the efficiency of spiking network learning. In *International conference on neural information processing* (pp. 172–179). Springer.

Tiňo, P., & Mills, A. J. S. (2006). Learning beyond finite memory in recurrent networks of spiking neurons. *Neural Computation*, *18*(3), 591–613.

Tuckwell, H. C., & Wan, F. Y. M. (2005). Time to first spike in stochastic Hodgkin–Huxley systems. *Physica A. Statistical Mechanics and its Applications*, *351*(2), 427–438.

Urbanczik, R., & Senn, W. (2009). A gradient learning rule for the tempotron. *Neural Computation*, *21*(2), 340–352.

Urbanczik, R., & Senn, W. (2014). Learning by the dendritic prediction of somatic spiking. *Neuron*, *81*(3), 521–528.

Vázquez, R. A., & Garro, B. A. (2011). Training spiking neurons by means of particle swarm optimization. In *International conference in swarm intelligence* (pp. 242–249). Springer.

Vázquez, R. A., & Garro, B. A. (2015). Training spiking neural models using artificial bee colony. *Computational Intelligence and Neuroscience*, *2015*, 18.

Wade, J. J., McDaid, L. J., Santos, J. A., & Sayers, H. M. (2010). SWAT: A spiking neural network training algorithm for classification problems. *IEEE Transactions on Neural Networks*, *21*(11), 1817–1830.

Walter, F., Röhrbein, F., & Knoll, A. (2016). Computation by time. *Neural Processing Letters*, *44*(1), 103–124.

Wang, J., Belatreche, A., Maguire, L., & McGinnity, M. (2010). Online versus offline learning for spiking neural networks: A review and new strategies. In *International conference on cybernetic intelligent systems* (pp. 1–6). IEEE.

Wang, J., Belatreche, A., Maguire, L., & McGinnity, T. M. (2014). An online supervised learning method for spiking neural networks with adaptive structure. *Neurocomputing*, *144*, 526–536.

Wang, J., Belatreche, A., Maguire, L. P., & McGinnity, T. M. (2017). SpikeTemp: An enhanced rank-order-based learning approach for spiking neural networks with adaptive structure. *IEEE Transactions on Neural Networks and Learning Systems*, *28*(1), 30–43.

Wang, X., Lin, X., & Dang, X. (2019). A delay learning algorithm based on spike train kernels for spiking neurons. *Frontiers in Neuroscience*, *13*, 252.

Wang, X., Lin, X., Zhao, J., & Ma, H. (2016). Supervised learning algorithm for spiking neurons based on nonlinear inner products of spike trains. In *International conference on intelligent computing* (pp. 95–104). Springer.

Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560.

Whalley, K. (2013). Neural coding: Timing is key in the olfactory system. *Nature Reviews Neuroscience*, *14*(7), 458.

Whittington, J. C. R., & Bogacz, R. (2017). An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. *Neural Computation*, *29*(5), 1229–1262.

Whittington, J. C. R., & Rafal, B. (2019). Theories of error back-propagation in the brain. *Trends in Cognitive Sciences*, *23*(3), 235–250.

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, Q. X., McGinnity, T. M., Maguire, L. P., Glackin, B., & Belatreche, A. (2006). Learning under weight constraints in networks of temporal encoding spiking neurons. *Neurocomputing*, *69*(16), 1912–1922.

Wysoski, S. G., Benuskova, L., & Kasabov, N. (2006). On-line learning with structural adaptation in a network of spiking neurons for visual pattern recognition. In *International conference on artificial neural networks* (pp. 61–70). Springer.

Xie, X., Qu, H., Liu, G., & Zhang, M. (2017). Efficient training of supervised spiking neural networks via the normalized perceptron based learning rule. *Neurocomputing*, *37*, 152–163.

Xie, X., Qu, H., Liu, G., Zhang, M., & Kurths, J. (2016). An efficient supervised training algorithm for multilayer spiking neural networks. *PLoS One*, *11*(4), e0150329.

Xie, X., Qu, H., Yi, Z., & Kurths, J. (2017). Efficient training of supervised spiking neural network via accurate synaptic-efficiency adjustment method. *IEEE Transactions on Neural Networks and Learning Systems*, *28*(6), 1411–1424.

Xin, J., & Embrechts, M. J. (2001). Supervised learning with spiking neural networks. In *International joint conference on neural networks (vol. 3)* (pp. 1772–1777). IEEE.

Xu, Y., Yang, J., & Zeng, X. (2019). An optimal time interval of input spikes involved in synaptic adjustment of spike sequence learning. *Neural Networks*, *116*, 11–24.

Xu, Y., Yang, J., & Zhong, S. (2017). An online supervised learning method based on gradient descent for spiking neurons. *Neural Networks*, *93*, 7–20.

Xu, Y., Zeng, X., Han, L., & Yang, J. (2013). A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks. *Neural Networks*, *43*, 99–113.

Xu, Y., Zeng, X., & Zhong, S. (2013). A new supervised learning algorithm for spiking neurons. *Neural Computation*, *25*(6), 1472–1511.

Yang, J., Yang, W., & Wu, W. (2012a). A remark on the error-backpropagation learning algorithm for spiking neural networks. *Applied Mathematics Letters*, *25*(8), 1118–1120.

Yang, W., Yang, J., & Wu, W. (2012b). Modified spiking neuron that involves derivative of the state function at firing time. *Neural Processing Letters*, *36*(2), 135–144.

Yu, Q., Li, H., & Tan, K. C. (2019). Spike timing or rate? Neurons learn to make decisions for both through threshold-driven plasticity. *IEEE Transactions on Cybernetics*, *49*(6), 2178–2189.

Yu, Q., Tang, H., Hu, J., & Tan, K. C. (2017). Temporal learning in multilayer spiking neural networks through construction of causal connections. In *Neuromorphic cognitive systems* (pp. 115–129). Springer.

Yu, Q., Tang, H., Tan, K. C., & Li, H. (2013a). Rapid feedforward computation by temporal encoding and learning with spiking neurons. *IEEE Transactions on Neural Networks and Learning Systems*, *24*(10), 1539–1552.

Yu, Q., Tang, H., Tan, K. C., & Li, H. (2013b). Precise-spike-driven synaptic plasticity: Learning hetero-association of spatiotemporal spike patterns. *PLoS One*, *8*(11), e78318.

Yu, Q., Tang, H., Tan, K. C., & Yu, H. (2014). A brain-inspired spiking neural network model with temporal encoding and learning. *Neurocomputing*, *138*, 3–13.

Yu, Q., Yan, R., Tang, H., Tan, K. C., & Li, H. (2016). A spiking neural network system for robust sequence recognition. *IEEE Transactions on Neural Networks and Learning Systems*, *27*(3), 621–635.

Zenke, F., & Ganguli, S. (2018). SuperSpike: Supervised learning in multi-layer spiking neural networks. *Neural Computation*, *30*(6), 1514–1541.

Zhang, Y., Geng, T., Zhang, M., Wu, X., Zhou, J., & Qu, H. (2018). Efficient and robust supervised learning algorithm for spiking neural networks. *Sensing and Imaging*, *19*(1), 8.

Zhang, Y., & Lin, X. (2015). Supervised learning based on convolution calculation for multilayer spiking neural networks. *Computer Engineering and Science*, *37*(2), 348–353.

Zhang, M., Qu, H., Belatreche, A., Chen, Y., & Yi, Z. (2019). A highly effective and robust membrane potential-driven supervised learning method for spiking neurons. *IEEE Transactions on Neural Networks and Learning Systems*, *30*(1), 123–137.

Zhang, M., Qu, H., Belatreche, A., & Xie, X. (2018). EMPD: An efficient membrane potential driven supervised learning algorithm for spiking neurons. *IEEE Transactions on Cognitive and Developmental Systems*, *10*(2), 151–162.

Zhang, M., Qu, H., Xie, X., & Kurths, J. (2017). Supervised learning in spiking neural networks with noise-threshold. *Neurocomputing*, *219*, 333–349.

Zhao, B., Yu, Q., Ding, R., Chen, S., & Tang, H. (2014). Event-driven simulation of the tempotron spiking neuron. In *Biomedical circuits and systems conference* (pp. 667–670). IEEE.

Zhao, J., Zurada, J. M., Yang, J., & Wu, W. (2018). The convergence analysis of spikeprop algorithm with smoothing $L_{1/2}$ regularization. *Neural Networks*, *103*, 19–28.

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.

Zheng, N., & Mazumder, P. (2018b). Learning in memristor crossbar-based spiking neural networks through modulation of weight-dependent spike-timing-dependent plasticity. *IEEE Transactions on Nanotechnology*, *17*(3), 520–532.

Zheng, G., Tonnelier, A., & Martinez, D. (2009). Voltage-stepping schemes for the simulation of spiking neural networks. *Journal of Computational Neuroscience*, *26*(3), 409–423.