Neurocomputing 237 (2017) 59–70

Contents lists available at ScienceDirect

Neurocomputing

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

Supervised learning in multilayer spiking neural networks with inner products of spike trains

Xianghong Lin, Xiangwen Wang *, Zhanjun Hao

School of Computer Science and Engineering, Northwest Normal University, Lanzhou 730070, PR China

ARTICLE INFO

Article history:
Received 23 August 2015
Received in revised form
6 April 2016
Accepted 28 August 2016
Communicated by Dr. A. Belatreche
Available online 31 August 2016

Keywords:
Spiking neural networks
Supervised learning
Inner products of spike trains
Multilayer feedforward network
Backpropagation algorithm

ABSTRACT

Recent advances in neurosciences have revealed that neural information in the brain is encoded through precisely timed spike trains, not only through the neural firing rate. This paper presents a new supervised, multi-spike learning algorithm for multilayer spiking neural networks, which can implement the complex spatio-temporal pattern learning of spike trains. The proposed algorithm firstly defines inner product operators to mathematically describe and manipulate spike trains, and then solves the problems of error function construction and backpropagation among multiple output spikes during learning. The algorithm is successfully applied to different temporal tasks, such as learning sequences of spikes and nonlinear pattern classification problems. The experimental results show that the proposed algorithm has higher learning accuracy and efficiency than the Multi-ReSuMe learning algorithm. It is effective for solving complex spatio-temporal pattern learning problems.

1. Introduction

Traditional artificial neural networks (ANNs) encode information by the firing rate of the biological neurons, and the outputs of neurons are generally expressed as analog variables in the given interval. Their learning algorithms are to minimize a selected cost or error function (a measure of the difference between the network outputs and the desired outputs) by adjusting the measures of synaptic strength. They mainly depend on the real values of neuronal outputs [1], such as the widely-used backpropagation (BP) training algorithm [2,3]. However, experimental evidence from the field of neuroscience suggests that neural systems encode information through the precise timing of spikes, not only through the neural firing rate [4,5]. Using a biologically plausible spiking neuron model [6,7] as the basic unit for constructing spiking neural networks (SNNs), they encode and process neural information through the precisely timed spike trains. SNNs are often referred to as the new generation of neural networks [8,9]. They have more powerful computing capacity to simulate a variety of neuronal signals and approximate any continuous function [10,11], and have been shown to be suitable tools for the processing of spatio-temporal information.

Supervised learning in ANNs involves a mechanism of providing the desired outputs with the corresponding inputs [12]. The network then processes the inputs and compares its resulting outputs against the desired outputs. Errors are calculated to control the synaptic weight adjustment. This process occurs over and over until the synaptic weights converge to certain values. The set of data that enables the training is called the training set. When the sample conditions changed, synaptic weights can be modified through supervised learning to adapt to the new environment. Experimental studies have shown that supervised learning exists in the biological nervous system, especially in the sensorimotor networks and sensory system [13–15], but there is no clear conclusion to explain how biological neurons realize this process. The purpose of supervised learning with temporal encoding for spiking neurons is to make the neurons emit arbitrary spike trains in response to given synaptic inputs. At present, researchers have conducted many studies on the supervised learning in SNNs [16,17], and achieved some results, but many problems remain unsolved. The supervised learning algorithms for SNNs proposed in recent years can be roughly divided into three categories: (1) supervised learning algorithms based on gradient descent, (2) supervised learning algorithms based on a synaptic plasticity mechanism, and (3) supervised learning algorithms based on the convolution of spike trains.

Supervised learning algorithms based on gradient descent use gradient computation and error backpropagation for adjusting the synaptic weights, and ultimately minimize the error function that indicates the deviation between the actual and the desired output spikes. Bohte et al. [18] first proposed a backpropagation training

* Corresponding author.
E-mail address: wangxiangwen2@163.com (X. Wang).

http://dx.doi.org/10.1016/j.neucom.2016.08.087
0925-2312/

algorithm for feedforward SNNs, called SpikeProp, similar in concept to the BP algorithm developed for traditional ANNs [2]. The spike response model (SRM) [19] is used in this algorithm. In SRM, the neuronal potential is represented by analytical expression. To overcome the discontinuity of the internal state variable caused by spike firing, all neurons in the network can fire only one single spike. Xin and Embrechts [20] presented a method with simple momentum that accelerates the convergence speed of the SpikeProp algorithm. In addition, McKennoch et al. proposed RProp and QuickProp algorithms [21] with faster convergence, and further extended SpikeProp to a class of nonlinear neuron models and constructed a BP algorithm in Theta neuron networks [22]. Fang et al. [23] proposed a learning rate adaptive method in which the learning rate dynamically changes in the learning process. However, all of the above algorithms encode information with a single spike, a limitation that means they cannot be effective for solving complex problems. A more important extension of SpikeProp was presented by Booij and Nguyen [24]. 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; this minimizes the time difference between the actual output spike and the target spike. Similarly, Ghosh-Dastidar and Adeli [25] put forward a BP learning algorithm named Multi-SpikeProp, with derivations of the learning rule based on the chain rule for a multi-spiking network model. Multi-SpikeProp was applied to the standard XOR problem and the Fisher Iris and EEG classification problems, and experimental results show that the algorithm has higher classification accuracy than the SpikeProp algorithm. Recently, Xu et al. [26] have extended the Multi-SpikeProp algorithm to allow neurons to fire multiple spikes in all layers. That is, the algorithm can implement the complex spatio-temporal pattern learning of spike trains. The experimental results show that this algorithm has higher learning accuracy for a large number of output spikes. Florian [27] introduced two supervised learning rules for spiking neurons with temporal coding of information (Chronotrons), and an E-learning rule based on gradient descent provides high memory capacity. But the E-learning rule is only suitable for a single neuron or single layer network. Supervised learning algorithms based on gradient descent have been developed mostly for simple neuron models that require the analytical expression of state variables. These algorithms cannot be applied to various neuron models.

Supervised learning algorithms based on a synaptic plasticity mechanism, in contrast, aim at modeling the learning rule from the correlation of spike firing times of the presynaptic neuron and postsynaptic neuron, which are more biologically plausible learning algorithms. In fact, the spike train can not only cause persistent changes of synaptic strength, but it also satisfies the spike-timing-dependent plasticity (STDP) mechanism [28,29]. Based on the STDP learning rule, many researchers have proposed various supervised learning algorithms suitable for SNNs. Legenstein et al. [30] presented a supervised Hebbian learning algorithm for spiking neurons, based on injecting external input current to make the learning neurons fire in a specific target spike train. Combining the Bienenstock-Cooper-Munro (BCM) learning rule with the STDP mechanism, a synaptic weight association training (SWAT) algorithm for SNNs is proposed [31], which yields a unimodal synaptic weight distribution where weight stabilization is achieved using the sliding threshold associated with the BCM model after a period of training. The chronotron I-Learning rule [27] changes the synaptic weights depending on the synaptic currents at the timings of actual and target output spikes. The algorithm has a high biological plausibility, but it can only be applied to single layer networks. Ponulak et al. [32,33] proposed the ReSuMe algorithm, which adjusts the synaptic weights according to STDP and anti-STDP processes and is suitable for various types of spiking neuron models. However, the algorithm can only be applied to single layer networks or train readouts for reservoir networks. Recently, Sporea and Grüning [34] extended the ReSuMe algorithm to multilayer feedforward SNNs using backpropagation of the network error. The weights are updated according to STDP and anti-STDP rules, and the neurons in every layer can fire multiple spikes. This algorithm is named Multi-ReSuMe. Simulation experiments show that the algorithm can be successfully applied to various complex classification problems and permits precise spike train encoding.

The main idea of the last class of supervised learning algorithms is the definition of convolution of the spike trains to design the proper learning rule. A spike train contains an abstraction of neurophysiological recordings [35], which is a simply sequence of ordered spike times. For convenience of analysis and calculation, we can select a specific convolution kernel to transform spike trains into continuous functions so that common mathematical operators can be performed on them. Through the convolution calculation of a spike train based on kernel function, the spike train can be interpreted as a specific neural physiological signal, such as neuronal postsynaptic potential or spike firing intensity function [36]. Evaluating the relationship between spike trains by the definition of convolution kernel, a supervised learning algorithm for SNNs can be constructed based on the difference of the kernelized spike trains. Carnell and Richardson [37] expanded the set of spike trains into a vector space, then applied linear algebra methods to implement the spatio-temporal pattern learning of spike trains. Mohemmed et al. [38,39] proposed a SPAN algorithm based on a Hebbian interpretation of the Widrow-Hoff rule and kernel function convolution. Inspired by the SPAN algorithm, Yu et al. [40,41] proposed a PSD supervised learning rule that can be used to train neurons to associate an input spatio-temporal spike pattern with a desired spike train. Unlike the SPAN method that requires spike convolution on all the spike trains of the input, the desired output and the actual output, the PSD learning rule only convolves the input spike trains. However, none of these three algorithms implements the error backpropagation mechanism, and all can be applied only to a single neuron or single layer SNNs.

For SNNs, input and output information is encoded through precisely timed spike trains, not only through the neural firing rate. In addition, the internal state variables of spiking neurons and error function do not satisfy the continuous differentiability. So, traditional learning algorithms of ANNs, especially the BP algorithm, cannot be used directly, and the formulation of efficient supervised learning algorithms for SNNs is a very challenging problem. In this paper, we present a new supervised learning algorithm for feedforward SNNs with multiple layers. We refer to this algorithm as Multi-STIP for multilayer SNNs learning with spike train inner products. The Multi-STIP learning algorithm combines the mechanism of error backpropagation, constructing a novel error function of spike trains and spanning to multiple layers, with a synaptic weight learning rule based on the difference of inner products of spike trains, which can be applied to neurons firing multiple spikes in all layers.

The rest of this paper is organized as follows. In Section 2 we analyze and define the inner products of spike trains. In Section 3 we construct the error function and derive the learning rule based on the inner products of spike trains for multilayer feedforward SNNs. In Section 4 the flexibility and power of feedforward SNNs trained with our algorithm are showcased by a spike sequence learning problem and a nonlinear pattern classification task. The discussion and conclusion are presented in Section 5.

2. Inner products of spike trains

The spike train $s = \{t_i \in \Gamma: i = 1, \ldots, N\}$ represents the ordered sequence of spike times fired by the spiking neuron in the interval

$\Gamma = [0, T]$, and can be expressed formally as:

$$s(t) = \sum_{i=1}^{N} \delta(t - t_i) \tag{1}$$

where N is the number of spikes, and $\delta(\cdot)$ represents the Dirac delta function, $\delta(x) = 1$ if $x=0$ and $\delta(x) = 0$ otherwise.

Before defining the inner products of spike trains, we should first define the inner products of spike times, because the inner products of spike trains can be combined with the inner products of spike times. Generally, the inner product of spike times is set by choosing a symmetric and positive definite kernel function [37,42]. For two given spikes corresponding to the fire times t_m and t_n, the inner product between two spikes can be defined in the form of kernel function:

$$\langle \delta(t - t_m), \delta(t - t_n) \rangle = \kappa(t_m, t_n), \quad \forall\, t_m, t_n \in \Gamma \tag{2}$$

The kernel function κ must be symmetric, shift-invariant and positive definite. In this paper, a Gaussian kernel is used in supervised learning with parameter $\sigma = 2.0$ and expressed as $\kappa(x, y) = \exp(-|x - y|^2/2\sigma^2)$.

In order to facilitate the analysis and calculation, we can choose a specific smoothing function h, using the convolution to convert the discrete spike train to a unique continuous function:

$$f_s(t) = s*h = \sum_{i=1}^{N} h(t - t_i) \tag{3}$$

Due to the limited time interval of the corresponding spike train and boundedness of the function $f_s(t)$, we can get:

$$\int_{\Gamma} f_s^2(t)dt < \infty \tag{4}$$

In other words, the function $f_s(t)$ is an element of $L_2(\Gamma)$ space.

For any two given spike trains $s_i, s_j \in S(\Gamma)$, we can define the inner product of the corresponding functions $f_{s_i}(t)$ and $f_{s_j}(t)$ on the $L_2(\Gamma)$ space as follows:

$$F(s_i, s_j) = \left\langle f_{s_i}(t), f_{s_j}(t) \right\rangle_{L_2(\Gamma)} = \int_{\Gamma} f_{s_i}(t) f_{s_j}(t)dt \tag{5}$$

Using the inner product representation of spike times, Eq. (5) can be further rewritten as the accumulation form of spike time pairs with order $O(N_i N_j)$:

$$F(s_i, s_j) = \sum_{m=1}^{N_i}\sum_{n=1}^{N_j} \int_{\Gamma} h\left(t - t_m^i\right) h\left(t - t_n^j\right)dt = \sum_{m=1}^{N_i}\sum_{n=1}^{N_j} \kappa\left(t_m^i, t_n^j\right) \tag{6}$$

where kernel κ is the autocorrelation of the smoothing function h, $\kappa(t_m, t_n) = \int_{\Gamma} h(t - t_m)h(t - t_n)dt$.

Because $F(s_i, s_j)$ has symmetry and positive properties, according to the Moore-Aronszajn theorem, there exists a reproducing kernel Hilbert space (RKHS) H_F induced by the spike train's inner products F [43]. That is, any spike train $s_i \in S(\Gamma)$ can be mapped to an element in the RKHS H_F. The analysis of inner product representation of spike trains and characteristics of RKHS H_F yields two advantages: (1) Converting the discrete spike train to a continuous function by constructing a special kernel function means the spike train can be interpreted as neural physiological signals, such as neuronal postsynaptic potential or spike firing intensity function. (2) Using the inner products of spike trains, we can obtain a formal definition of the spike trains similarity measure, which is the basis of an error function for supervised learning in multilayer SNNs.

3. Learning algorithm: Multi-STIP

The multilayer SNNs used in this study are fully connected feedforward networks. All neurons in one layer are connected to all neurons in the subsequent layer. In order to simplify the learning rule for simpler description, the network only contains one hidden layer. The feedforward SNNs contain three layers, including the input layer, hidden layer and output layer, and the number of neurons in each layer is N_I, N_H and N_O respectively. For multiple hidden layers of SNNs, the derivation of the learning rule is analogous.

The input and output signals of spike neurons are expressed in the form of spike trains; that is, spike trains encode neural information or external stimuli. The computation performed by single spiking neurons can be defined as a mapping from the input spike trains to the appropriate output spike train. For a given spiking neuron, we assume that the input spike trains are $s_i \in S(\Gamma)$, $i = 1, \ldots, N$, and the output spike train is $s_o \in S(\Gamma)$. In order to analyze the relationship between the input and output spike trains, we use the linear Poisson neuron model [44]. This neuron model outputs a spike train, which is a realization of a Poisson process with the underlying intensity function estimation. The spiking activity of the postsynaptic neuron is defined by the estimated intensity functions of the presynaptic neurons. The contributions of all input spike trains are summed up linearly:

$$f_{s_o}(t) = \sum_{i=1}^{N} w_{oi} f_{s_i}(t) \tag{7}$$

where the weights w_{oi} represent the strength of the connection between the presynaptic neuron i and the postsynaptic neuron o. There are two reasons for this simplification. (1) Although dendritic trees of neurons have complex structures for information processing, the linear summation of inputs has been observed both in hippocampal pyramidal neurons [45] and cerebellar Purkinje cells [46]. (2) The linear summation of smoothed spike trains will be used for the derivation of the corresponding learning rule, in accordance with the preliminary results reported by Carnell and Richardson [37].

3.1. The error function of multilayer SNNs

The goal of supervised learning for SNNs is that for a given input spike train pattern, the output neurons eventually fire the desired spike trains by adjusting the synaptic weights. Therefore, the key of the supervised learning algorithm for multilayer feedforward SNNs is to define the spike train error function and the learning rule of synaptic weights.

In order to compute the network error, we first convert the actual spike train $s_o^a \in S(\Gamma)$ and the desired spike train $s_o^d \in S(\Gamma)$ of the output neuron to continuous functions using Eq. (3). The instantaneous network error is formally defined in terms of the square difference between the corresponding smoothed functions $f_{s_o^a}(t)$ and $f_{s_o^d}(t)$ at time t for all output neurons. It can be represented as:

$$E(t) = \frac{1}{2}\sum_{o=1}^{N_O}\left[f_{s_o^a}(t) - f_{s_o^d}(t)\right]^2 \tag{8}$$

From Eqs. (5) and (8), the total error of the network in the time interval Γ is:

$$\begin{aligned} E &= \int_{\Gamma} E(t)dt = \frac{1}{2}\sum_{o=1}^{N_O}\int_{\Gamma}\left[f_{s_o^a}(t) - f_{s_o^d}(t)\right]^2 dt \\ &= \frac{1}{2}\sum_{o=1}^{N_O}\left\langle f_{s_o^a}(t) - f_{s_o^d}(t), f_{s_o^a}(t) - f_{s_o^d}(t)\right\rangle \\ &= \frac{1}{2}\sum_{o=1}^{N_O}\left[F(s_o^a, s_o^a) - 2F(s_o^a, s_o^d) + F(s_o^d, s_o^d)\right] \end{aligned} \tag{9}$$

Thus, the spike train error function in the network can be computed by the spike train inner products.

3.2. The learning rule of multilayer SNNs

Using error backpropagation for adjusting synaptic weights, the generalized delta update rule is employed to backpropagate the network error and modify the synaptic weights. The synaptic weight w from the presynaptic neuron to the postsynaptic neuron is computed as follows:

$$\Delta w = -\eta \nabla E \tag{10}$$

where η is the learning rate and ∇E is the gradient of the spike train error function E for the synaptic weight w. The gradient can be expressed as the integration of derivative of the instantaneous network error $E(t)$ with respect to synaptic weight w in the time interval Γ:

$$\nabla E = \int_{\Gamma} \frac{\partial E(t)}{\partial w} dt \tag{11}$$

According to the synapses in different layers, the computation of the gradient is different for the output layer and the hidden layer.

For an output neuron o in the SNN, using the chain rule, the derivative of the error function $E(t)$ at time t can be represented as the product of two partial derivative terms:

$$\frac{\partial E(t)}{\partial w_{oh}} = \frac{\partial E(t)}{\partial f_{s_o^a}(t)} \frac{\partial f_{s_o^a}(t)}{\partial w_{oh}} \tag{12}$$

where w_{oh} represents the synaptic weight between the output neuron o and hidden neuron h. The first partial derivative term of the right-hand part of Eq. (12) is computed as:

$$\frac{\partial E(t)}{\partial f_{s_o^a}(t)} = \frac{\partial \left[\frac{1}{2} \sum_{o=1}^{N_O} \left[f_{s_o^a}(t) - f_{s_o^d}(t) \right]^2 \right]}{\partial f_{s_o^a}(t)} = f_{s_o^a}(t) - f_{s_o^d}(t) \tag{13}$$

Using the linear summation of input spike trains for the linear Poisson neuron model, the second partial derivative term of the right-hand part of Eq. (12) is computed as:

$$\frac{\partial f_{s_o^a}(t)}{\partial w_{oh}} = \frac{\partial \left[\sum_{h=1}^{N_H} w_{oh} f_{s_h}(t) \right]}{\partial w_{oh}} = f_{s_h}(t) \tag{14}$$

where $s_h \in S(\Gamma)$ is the spike train fired by the neuron h in the hidden layer. By combining Eqs. (12)–(14), the derivative of the error function $E(t)$ at time t to the output neurons becomes:

$$\frac{\partial E(t)}{\partial w_{oh}} = \left[f_{s_o^a}(t) - f_{s_o^d}(t) \right] f_{s_h}(t) \tag{15}$$

Substituting Eq. (15) in Eq. (11), the error gradient for adjusting the synaptic weights between the output and hidden neurons, ∇E_{oh}, is computed using the inner products of spike trains:

$$\nabla E_{oh} = \int_{\Gamma} \left[f_{s_o^a}(t) - f_{s_o^d}(t) \right] f_{s_h}(t) dt = F(s_o^a, s_h) - F(s_o^d, s_h) \tag{16}$$

For a postsynaptic neuron h in the hidden layer, the derivative of the error $E(t)$ with respect to the synaptic weight w_{hi} is computed using the chain rule as:

$$\frac{\partial E(t)}{\partial w_{hi}} = \frac{\partial E(t)}{\partial f_{s_h}(t)} \frac{\partial f_{s_h}(t)}{\partial w_{hi}} \tag{17}$$

where w_{hi} represents the synaptic weight between the hidden neuron h and input neuron i. The first term of the right-hand part of the above equation models the dependence of the instantaneous network error on the smoothed spike train $f_{s_h}(t)$ of the hidden neuron h at time t. From Eq. (8), the instantaneous network error is backpropagated from all neurons in the output layer, and the first partial derivative term in Eq. (17) is expanded using the chain rule as:

$$\frac{\partial E(t)}{\partial f_{s_h}(t)} = \sum_{o=1}^{N_O} \frac{\partial E(t)}{\partial f_{s_o^a}(t)} \frac{\partial f_{s_o^a}(t)}{\partial f_{s_h}(t)} \tag{18}$$

The second partial derivative term of the right-hand part of Eq. (18) is computed from Eq. (7):

$$\frac{\partial f_{s_o^a}(t)}{\partial f_{s_h}(t)} = \frac{\partial \left[\sum_{h=1}^{N_H} w_{oh} f_{s_h}(t) \right]}{\partial f_{s_h}(t)} = w_{oh} \tag{19}$$

By combining Eqs. (13) and (19), (18) is rewritten as:

$$\frac{\partial E(t)}{\partial f_{s_h}(t)} = \sum_{o=1}^{N_O} \left[f_{s_o^a}(t) - f_{s_o^d}(t) \right] w_{oh} \tag{20}$$

For the linear Poisson neuron model, using again the linear summation relationship of the input and output spike trains, the second partial derivative term of the right-hand part of Eq. (17) is computed as:

$$\frac{\partial f_{s_h}(t)}{\partial w_{hi}} = f_{s_i}(t) \tag{21}$$

where $s_i \in S(\Gamma)$ is the spike train fired by the input neuron i. From Eqs. (20) and (21), the derivative of the error function $E(t)$ at time t to the hidden neurons becomes:

$$\frac{\partial E(t)}{\partial w_{hi}} = \sum_{o=1}^{N_O} \left[f_{s_o^a}(t) - f_{s_o^d}(t) \right] f_{s_i}(t) w_{oh} \tag{22}$$

Since the total synaptic weight change is the integration of derivative of the instantaneous network error over the duration of the spike trains, the error gradient for adjusting the synaptic weights between the hidden and input neurons is computed as:

$$\begin{aligned} \nabla E_{hi} &= \int_{\Gamma} \sum_{o=1}^{N_O} \left[f_{s_o^a}(t) - f_{s_o^d}(t) \right] f_{s_i}(t) w_{oh} dt \\ &= \sum_{o=1}^{N_O} \left[\left\langle f_{s_o^a}(t), f_{s_i}(t) \right\rangle - \left\langle f_{s_o^d}(t), f_{s_i}(t) \right\rangle \right] w_{oh} \\ &= \sum_{o=1}^{N_O} \left[F(s_o^a, s_i) - F(s_o^d, s_i) \right] w_{oh} \end{aligned} \tag{23}$$

The spike train inner products in Eqs. (16) and (23) can be represented as the accumulation form of inner products of spike time pairs using Eq. (6). Therefore, the new learning rules based on the inner products of spike trains for multilayer SNNs are obtained.

(1) The adjustment rule of synaptic weight between a neuron in the output layer and a neuron in the hidden layer is:

$$\begin{aligned} \Delta w_{oh} &= -\eta \left[F\left(s_o^a, s_h\right) - F\left(s_o^d, s_h\right) \right] \\ &= \eta \left[\sum_{m=1}^{N_o^d} \sum_{n=1}^{N_h} \kappa\left(t_m^d, t_n^h\right) - \sum_{m=1}^{N_o^a} \sum_{n=1}^{N_h} \kappa\left(t_m^a, t_n^h\right) \right] \end{aligned} \tag{24}$$

(2) The adjustment rule of synaptic weight between a neuron in the hidden layer and a neuron in the input layer is:

$$\Delta w_{hi} = -\eta \sum_{o=1}^{N_O} \left[F\left(s_o^a, s_i\right) - F\left(s_o^d, s_i\right) \right] w_{oh}$$
$$= \eta \sum_{o=1}^{N_O} \left[\sum_{m=1}^{N_o^d} \sum_{n=1}^{N_i} \kappa\left(t_m^d, t_n^i\right) - \sum_{m=1}^{N_o^a} \sum_{n=1}^{N_i} \kappa\left(t_m^a, t_n^i\right) \right] w_{oh} \quad (25)$$

4. Simulations

In this section, several experiments are presented to demonstrate the learning capabilities of Multi-STIP algorithm. At first, the algorithm is applied to the learning sequences of spikes, by demonstrating its ability to associate a spatio-temporal spike pattern with a target spike train. Furthermore, we analyze the factors that may influence the learning performance, such as the learning rate, the length of desired output spike trains, the number of synaptic inputs, the number of neurons in the hidden layers, a different parameter σ of Gaussian kernel and different kernel functions. The algorithm is also applied to a classification problem, with the final experiment demonstrating its performance on the Wisconsin Breast Cancer dataset and Fisher Iris dataset.

The network used in the simulations is feedforward architecture with three layers. The neurons are described by the short term memory SRM so that only the last firing spike contributes to the refractoriness. [26] (See Appendix A.1 for details of parameter settings.) For the Multi-STIP algorithm, we set the learning rate $\eta = 0.005$. The kernel function used in its learning rules is Gaussian kernel. We have also performed simulations using the Multi-ReSuMe algorithm (see Appendix A.2 for a complete description), in order to compare it to the proposed Multi-STIP algorithm, and its learning rate $\eta = 0.5$.

4.1. Learning sequences of spikes

In this experiment, we demonstrate the learning ability of the proposed Multi-STIP algorithm for training an SNN to reproduce the desired spatio-temporal spike pattern. Unless stated otherwise, the basic parameter settings of the SNNs are: the number of input neurons $N_I = 50$, the number of hidden neurons $N_H = 100$ and one output neuron. Initially, the synaptic weights are generated as the uniform distribution in the interval $(0, 0.2)$. Every input spike train in the input layer is generated randomly by a homogeneous Poisson process with rate $r = 20$ Hz within the time interval of $(0, T)$, and we set $T = 100$ ms here. The desired output spike train is generated with a specific SNN that is assigned randomly in each learning. The network error is calculated using Eq. (9), which expresses the similarity between the desired and actual output spike trains. The results are averaged over 100 trials, and on each testing trial the learning algorithm is applied for a maximum of 500 learning epochs or until the network error $E = 0$.

Fig. 1 shows the learning process of the Multi-STIP algorithm with Gaussian kernel. Fig. 1(a) shows the SNN architecture used in this experiment. Fig. 1(b) shows the complete learning process, which includes the desired output spike train, the initial output spike train and the actual output spike trains at some learning epochs during the learning process. The evolution of the total network error in the time interval T is represented in Fig. 1(c). We note that the error function value decreases rapidly at the beginning of the learning, and is reduced to 0 after 50 learning epochs. Fig. 1(d) is the diagram of synaptic weight changes before and after the training process. In this simulation, there are 5100 synaptic weights in total. For convenience of description, we have shown the changes of 100 synaptic weights between the hidden layer and the output layer. The results show that the SNN can successfully

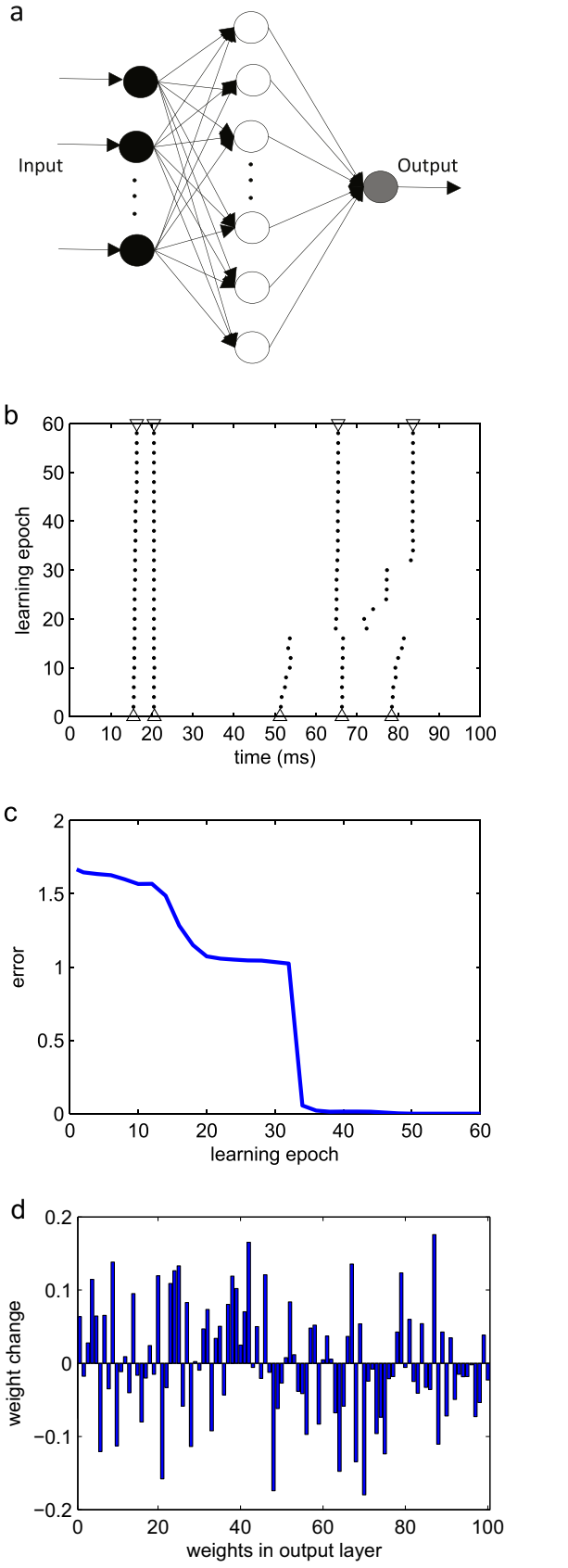

Fig. 1. The learning process of Multi-STIP with Gaussian kernel. (a) The SNN architecture. (b) The complete learning process. ▵, initial actual output spike train; ▿, desired output spike train; •, actual output spike trains at some learning epochs. (c) The evolution of the network error. (d) Synaptic weight changes before and after training process.

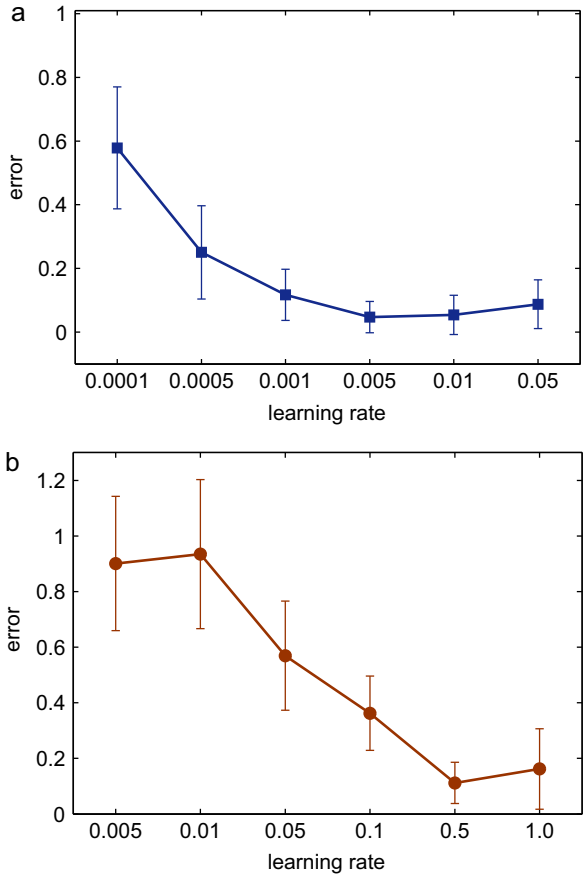

Fig. 2. The learning results and comparison of learning performance between Multi-STIP and Multi-ReSuMe with different learning rates. (a) The minimum error of Multi-STIP. (b) The minimum error of Multi-ReSuMe.

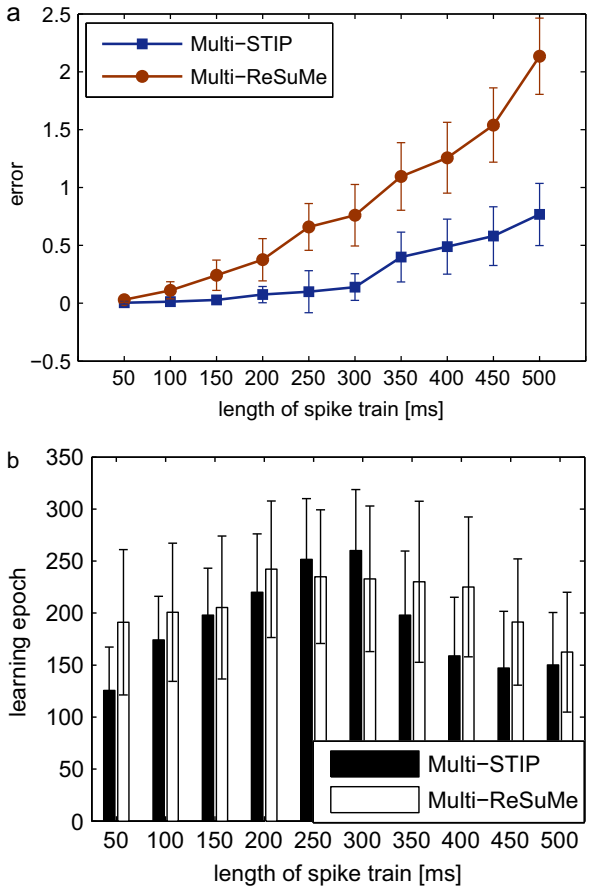

Fig. 3. The learning results and comparison of learning performance between Multi-STIP and Multi-ReSuMe with different lengths of desired output spike trains. (a) The minimum error. (b) The learning epochs when the error reaches the minimum value.

learn the desired spike train using the Multi-STIP algorithm.

Fig. 2 shows the learning results with different learning rates. In this experiment, we compare the Multi-STIP algorithm with the Multi-ReSuMe algorithm [34]. The parameters in the Multi-ReSuMe algorithm are the same as those described by Sporea and Grüning [34]. Fig. 2(a) shows the spike sequence learning results of Multi-STIP with learning rates 0.0001, 0.0005, 0.001, 0.005, 0.01 and 0.05. From Fig. 2(a) we can see that the learning error decreases at first, then increases when the learning rate increases gradually, so the suitable learning rate for Multi-STIP is 0.005. Fig. 2(b) shows the spike sequence learning results of Multi-ReSuMe with learning rates 0.005, 0.01, 0.05, 0.1, 0.5 and 1.0. From Fig. 2(b) we can see that the learning error decreases at first, then increases when the learning rate increases gradually, so the suitable learning rate for Multi-ReSuMe is 0.5.

Fig. 3 shows the learning results with different lengths of desired output spike trains. The length of desired output spike trains increases gradually, while the other parameter settings remain the same. The numbers of input neurons and hidden neurons are 50 and 100 respectively. The length of the desired output spike trains is increased from 100 ms to 500 ms with an interval of 50 ms. Fig. 3(a) shows the minimum error after 500 learning epochs while Fig. 3(b) shows the learning epochs when the error reaches the minimum value. Fig. 3(a) illustrates that both the Multi-STIP algorithm and the Multi-ReSuMe algorithm can learn with high accuracy when the length of the desired output spike trains is moderate. The learning accuracy of the two algorithms decreases when the length of the desired output spike trains increases gradually. However, the Multi-STIP algorithm can learn with higher accuracy than the Multi-ReSuMe algorithm when the length of the desired output spike trains increases. For example, the minimum error of 0.1399 for the Multi-STIP algorithm is smaller than 0.7606 for the Multi-ReSuMe algorithm when the length of the desired spike trains is 300 ms. The minimum error of 0.7677 for Multi-STIP is smaller than 2.1340 for Multi-ReSuMe when the length of the desired spike trains is 500 ms. From Fig. 3(b) we can see that the learning epochs of the two algorithms tend to increase firstly and then decrease. This is because when the length of the desired output spike trains increases, the learning accuracy of two algorithms decreases. But the Multi-STIP algorithm has smaller learning epochs than those of the Multi-ReSuMe algorithm.

Fig. 4 shows the learning results with different numbers of synaptic inputs. The number of synaptic inputs increases from 30 to 210 with an interval of 20, while the other settings remain the same. Fig. 4(a) shows the minimum error after 500 learning epochs and Fig. 4(b) shows the learning epochs when the error reaches the minimum value. Through Fig. 4(a) we can see that both the Multi-STIP algorithm and the Multi-ReSuMe algorithm can learn with high accuracy when the number of synaptic inputs is few. The learning accuracy of the two algorithms decreases when the number of synaptic inputs increases gradually. This is because when the number of synaptic inputs increased, the length of the desired output spike trains increased too, and the learning results became worse. However, the Multi-STIP algorithm can learn with higher accuracy than the Multi-ReSuMe algorithm when the number of synaptic inputs increases. For example, the minimum error of 0.3612 for Multi-STIP is smaller than 0.8953 for Multi-ReSuMe when the number of synaptic inputs is 150. The minimum error of 0.3840 for Multi-STIP is smaller than 1.0254 for

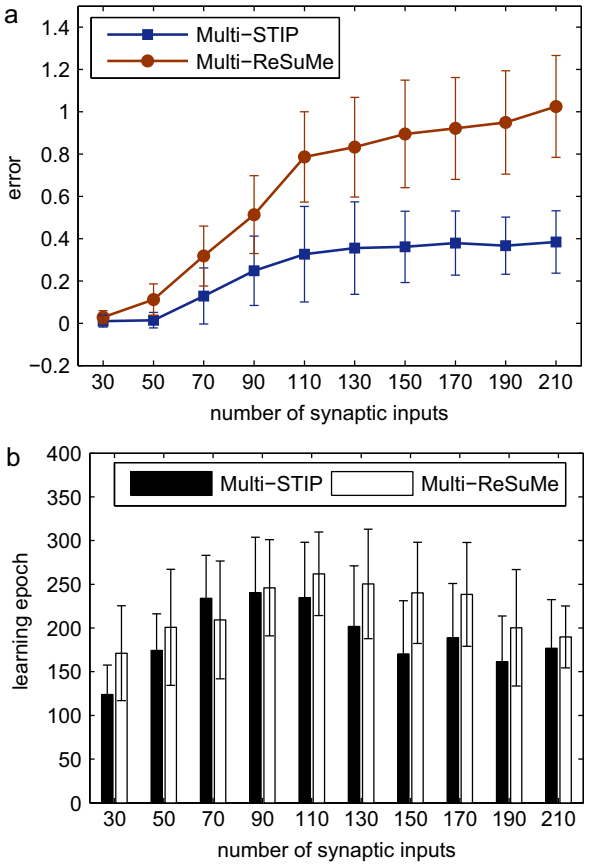

Fig. 4. The learning results with different numbers of synaptic inputs for the Multi-STIP algorithm and the Multi-ReSuMe algorithm after 500 learning epochs. (a) The minimum error. (b) The learning epochs when the error reaches the minimum value.

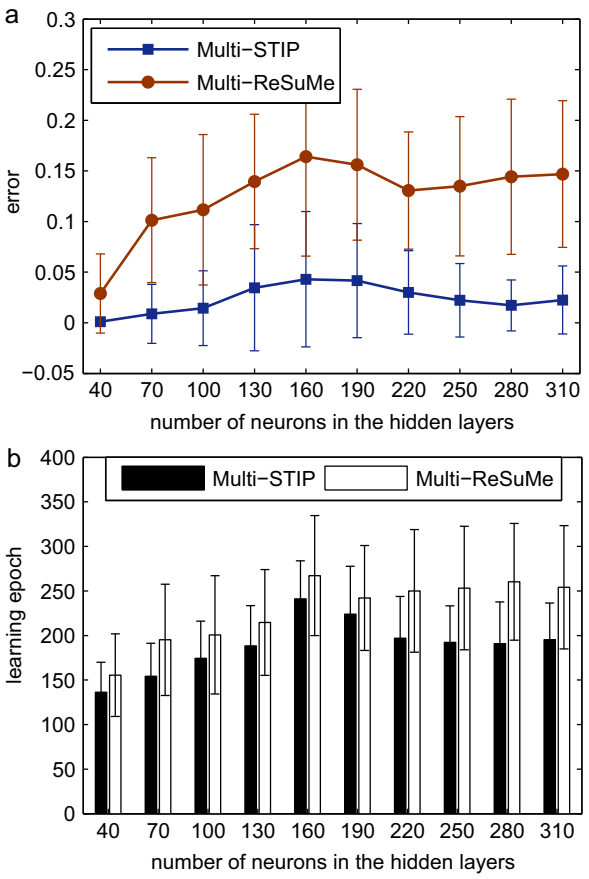

Fig. 5. The learning results with different numbers of neurons in the hidden layer of the Multi-STIP algorithm and the Multi-ReSuMe algorithm after 500 learning epochs. (a) The minimum error. (b) The learning epochs when the error reaches the minimum value.

Multi-ReSuMe when the number of synaptic inputs is 210. From Fig. 4(b) we can see that the learning epochs of the Multi-STIP algorithm are smaller than those of the Multi-ReSuMe algorithm.

Fig. 5 shows the learning results with different numbers of neurons in the hidden layer. The number of neurons in the hidden layer is increased from 40 to 310 with an interval of 30, while the other settings remain the same. Fig. 5(a) shows the minimum error after 500 learning epochs and Fig. 5(b) shows the learning epochs when the error reaches the minimum value. From Fig. 5 (a) we note that both Multi-STIP and Multi-ReSuMe can learn with high accuracy. The learning accuracy of the two algorithms tends to decrease slowly and does not change too much when the number of neurons in the hidden layers increases gradually. This is because the length of the desired output spike trains increased slightly when the number of neurons in the hidden layer increased. Additionally, the training time increased when the number of neurons in the hidden layer increased. However, the Multi-STIP algorithm can learn with higher accuracy than the Multi-ReSuMe algorithm. For example, the minimum error of 0.0416 for Multi-STIP is smaller than 0.1560 for Multi-ReSuMe when the number of neurons in the hidden layer is 190. The minimum error of 0.0225 for Multi-STIP is smaller than 0.1468 for Multi-ReSuMe when the number of neurons in the hidden layer is 310. From Fig. 5(b) we can see that the learning epochs of the Multi-STIP algorithm are smaller than those of the Multi-ReSuMe algorithm.

As for Multi-STIP, learning rules of the synaptic weights are finally presented as the inner products of spike trains. In order to investigate the influence of different kernel functions on the learning performance of Multi-STIP, we choose the Gaussian kernel, Laplacian kernel, inverse multiquadratic kernel and α-kernel to test the algorithm. The kernel functions used in this paper are listed in Table 1.

Because Gaussian kernel was used in the earlier experiments, we first experimented to validate the learning results of the Multi-STIP algorithm with different parameters σ of Gaussian kernel. The training results are shown in Fig. 6, for parameter values σ of Gaussian kernel 0.1, 0.5, 1.0, 2.0, 5.0 and 10.0. Fig. 6(a) shows the minimum error after 500 learning epochs and Fig. 6(b) shows the learning epochs when the error reaches the minimum value. From Fig. 6(a) we note that the minimum error of the Multi-STIP algorithm is 2.1790 when $\sigma = 0.1$. With the increasing of the kernel function parameter σ, the learning accuracy increases gradually, until the minimum error of the Multi-STIP algorithm reaches its minimum value, 0.0144, when $\sigma = 2.0$. But when the kernel function parameter σ increases further, the learning accuracy decreases gradually. From Fig. 6(b) it can be seen that the learning epochs of Multi-STIP decrease gradually as the parameter σ of Gaussian kernel increases. The learning epoch of the Multi-STIP algorithm is 118.61 when $\sigma = 0.5$. The learning epoch of the Multi-

Table 1
Different kernel functions in the Multi-STIP algorithm.

Kernel	Expression
Gaussian	$\kappa(x, y) = \exp(-\lvert x - y\rvert^2/2\sigma^2)$
Laplacian	$\kappa(x, y) = \exp(-\lvert x - y\rvert/\sigma)$
Inverse multiquadratic	$\kappa(x, y) = 1/\sqrt{\lvert x - y\rvert^2 + c^2}$
α-function	$\kappa(x, y) = \frac{e^{\lvert x-y\rvert}}{\sigma} \exp(-\lvert x - y\rvert/\sigma)$

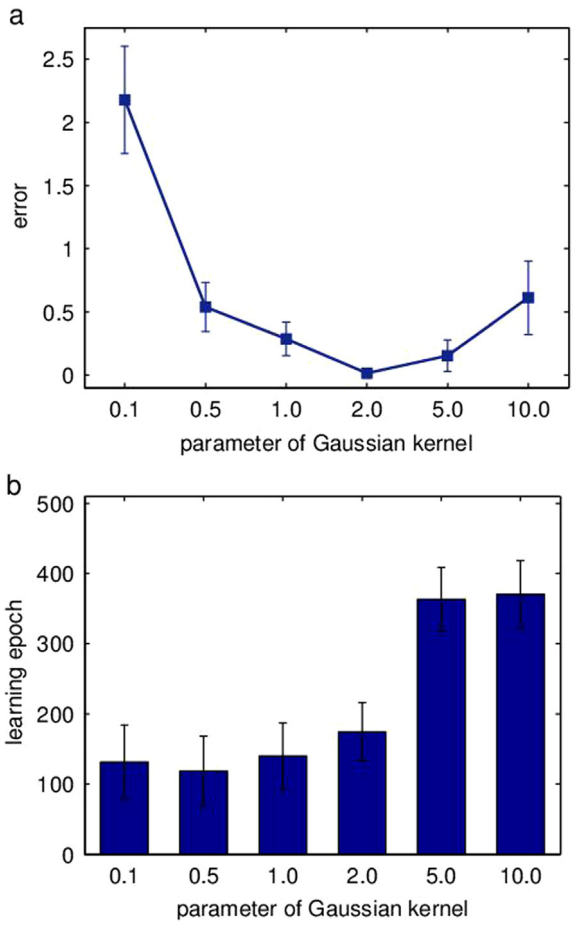

Fig. 6. The learning results of the Multi-STIP algorithm with different parameters σ of Gaussian kernel. (a) The minimum error. (b) The learning epochs when the error reaches the minimum value.

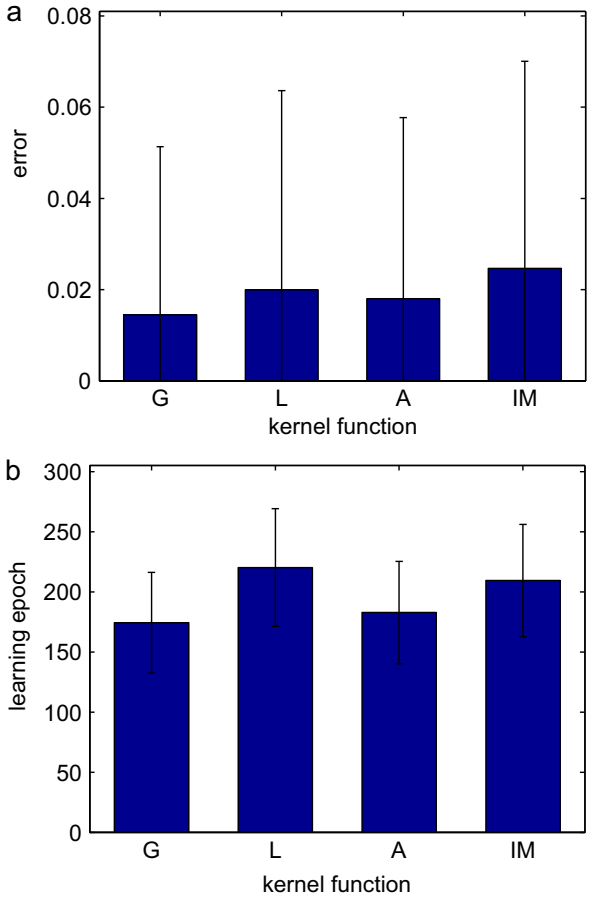

Fig. 7. The learning results of the Multi-STIP algorithm with different kernel functions. (a) The minimum error. (b) The learning epochs when the error reaches the minimum value.

STIP algorithm is 174.28 when $\sigma = 2.0$. Based on the above analysis, the parameter σ of Gaussian kernel has a great influence on learning accuracy. In our experiment, the optimal parameter σ of Gaussian kernel is 2.0.

Fig. 7 shows the learning results of the Multi-STIP algorithm with different kernel functions. In this experiment, the parameters of Gaussian kernel, Laplacian kernel, α-kernel and inverse multiquadratic kernel are $\sigma = 2.0$, $\sigma = 5.0$, $\sigma = 0.5$, and $c = 1.0$ respectively. Fig. 7(a) shows the minimum error after 500 learning epochs, finally the minimum error of Gaussian kernel (G), Laplacian kernel (L), α-kernel (A) and inverse multiquadratic kernel (IM) are 0.0144, 0.0199, 0.0180 and 0.0246, respectively. Fig. 7 (b) shows the learning epochs when the error reaches the minimum value, finally the learning epochs of Gaussian kernel, Laplacian kernel, α-kernel and inverse multiquadratic kernel are 174.28, 220.22, 182.82 and 209.43, respectively. Through the experiment we found that polynomial kernel, Sigmoid kernel and multiquadratic kernel are not suitable for the Multi-STIP algorithm because of learning failure or poor results. Other functions such as Gaussian kernel, Laplacian kernel, α-kernel and inverse multiquadratic kernel can obtain an ideal result.

4.2. Nonlinear pattern classification

In this experiment the learning algorithm is tested on nonlinear pattern classification problems to validate the algorithm for solving actual nonlinear pattern classification problems. The algorithm is simulated for the Wisconsin Breast Cancer and Fisher Iris classification benchmark classification problems to test the learning ability.

The general scheme of the experiment is similar to that described in Section 4.1. There are 50 neurons in the hidden layer and one output neuron. The synaptic weights are randomly generated uniformly in the range (0, 1.0). The results are averaged over 20 trials, and on each testing trial the learning algorithm is applied for a maximum of 100 learning epochs.

The Wisconsin Breast Cancer dataset is a commonly used benchmark in the pattern classification field. The dataset includes 699 samples comprising 458 benign tumors and 241 malignant tumors. There are 16 samples in the dataset for which some data are missing, and these samples have been removed. Each sample consists of nine feature values and one output value. Each feature value is represented by an integer in the range of [1,10], the output values 2 and 4 represent benign tumor and malignant tumor, respectively. It can be judged whether the tumor is benign or malignant by such nine features of it like the size of the tumor and the others.

The Wisconsin Breast Cancer dataset is divided into two groups randomly where one group is the training set consisting of 342 samples and the other group is the testing set consisting of 341 samples. In our experiment, the input feature values are set using a linear coding scheme. Firstly, the nine feature values of each sample are normalized and encoded in the frequency range of [30, 50 Hz] with a linear coding scheme. Then, each frequency value is converted into spike trains in the interval of [0, 50 ms]. Finally, the

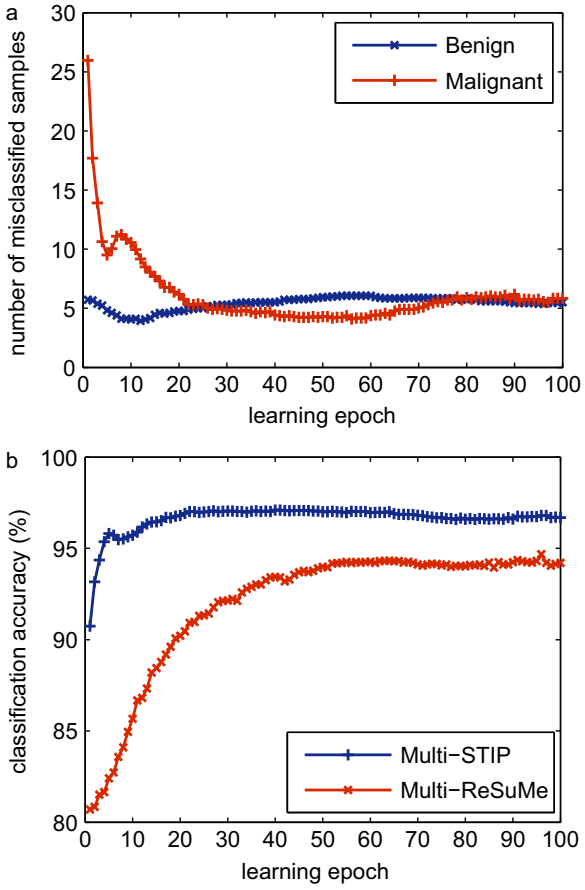

Fig. 8. The training results of the Multi-STIP algorithm and the Multi-ReSuMe algorithm for the Wisconsin Breast Cancer dataset after 100 learning epochs. (a) The evolution of the number of misclassified samples on the training set of the Multi-STIP algorithm. (b) The evolution of classification accuracy on the training set of the Multi-STIP algorithm and the Multi-ReSuMe algorithm.

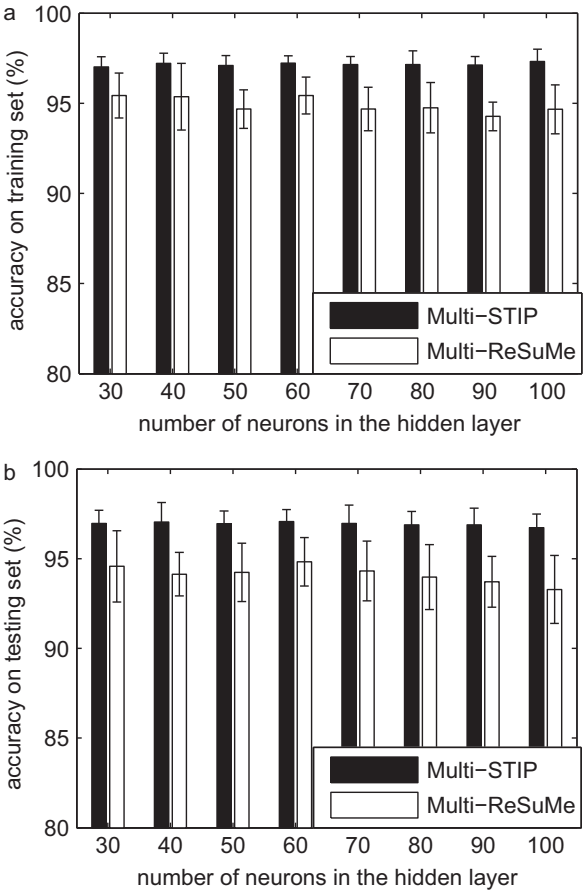

Fig. 9. The learning results of the Multi-STIP algorithm and the Multi-ReSuMe algorithm for the Wisconsin Breast Cancer dataset with different numbers of neurons in the hidden layer. (a) The highest classification accuracy on the training set. (b) The average classification accuracy on the testing set.

nine feature values of each sample are encoded into nine linear input spike trains. The output values 2 and 4 are encoded into target spike trains with frequencies 32 Hz and 38 Hz respectively. We assume that the training samples are classified correctly if the number of spikes of the output spike train for every training sample is larger than the given number of spikes fired; otherwise, it is a wrong classification.

Fig. 8 shows the training results of Multi-STIP and Multi-ReSuMe for the Wisconsin Breast Cancer dataset after 100 learning epochs. Fig. 8(a) shows the evolution of the number of misclassified samples on the training set of Multi-STIP algorithm. The number of misclassified samples of benign and malignant finally reached 3.90 and 4.10 respectively. Fig. 8(b) shows the classification accuracy on the training set of Multi-STIP and Multi-ReSuMe. We note that the accuracy of the Multi-STIP algorithm on the training set is higher than that of the Multi-ReSuMe algorithm with final classification accuracies of 97.11% for Multi-STIP and 94.68% for Multi-ReSuMe.

Fig. 9 shows the learning results of the Multi-STIP algorithm and the Multi-ReSuMe algorithm for the Wisconsin Breast Cancer dataset with different numbers of neurons in the hidden layer after 100 learning epochs. Fig. 9(a) shows the highest classification accuracy on the training set while Fig. 9(b) shows the average classification accuracy on the testing set. The number of neurons in the hidden layer is increased from 30 to 100 with an interval of 10. From the learning results we can see that for the Multi-STIP algorithm, classification accuracy on both the training set and the testing set is higher than that of the Multi-ReSuMe algorithm. For example, when the number of neurons in the hidden layer is 40, the classification accuracy on the training set and the testing set of the Multi-STIP algorithm is 97.22% and 97.04% respectively, which is higher than 95.37% and 94.13% of Multi-ReSuMe algorithm respectively. When the number of neurons in the hidden layer is 70, Multi-STIP's classification accuracy on the training set and the testing set is 97.15% and 96.96% respectively, and both values are higher than the 94.68% and 94.31% of the Multi-ReSuMe algorithm.

Table 2 compares the classification accuracy of Multi-STIP against existing algorithms for the Wisconsin Breast Cancer dataset [31]. For this dataset, the classification accuracy is comparable to that of the other approaches. when the number of neurons in the hidden layer is 60, the classification accuracy on the training set and the testing set of the Multi-STIP algorithm is 97.22% and 97.05% respectively, which is higher than 95.42% and 94.82% of Multi-ReSuMe algorithm respectively.

Table 2
Comparison of training algrithm: results for Wisconsin Breast Cancer dataset.

Algorithm	Training set	Testing set
SpikeProp	97.6% ± 0.2	97.0% ± 0.6
SWAT	96.2% ± 0.4	96.7% ± 2.3
Multi-ReSuMe	95.4% ± 1.0	94.8% ± 1.4
Multi-STIP	97.2% ± 0.4	97.1% ± 0.7

The Fisher Iris dataset is another commonly used benchmark in the pattern classification field. The dataset contains three different irises: setosa, versicolor and virginica. Each flower has 50 samples, a total of 150 samples. For each sample, the data represents 4 attributes of the flower: the length of the sepals, the width of the sepals, the length of the petals and the width of the petals. The Fisher Iris dataset is divided into two groups randomly, where one group is the training set consisting of 75 samples and the other group is the testing set consisting of 75 samples. In the experiment, the input feature values were set using a linear coding scheme. Firstly, the four feature values of each sample are normalized and encoded in the frequency range of [30, 45 Hz] by a linear coding scheme. Then each frequency value is converted into the spike trains in the interval of [0, 50 ms]. Finally, the four feature values of each sample are encoded into four linear input spike trains. The output values 1, 2 and 3 are encoded into target spike trains with frequencies of 32 Hz, 36 Hz and 39 Hz respectively. We assume that the training samples are classified correctly if the number of spikes of the output spike train of every training sample is larger than the given number of spikes fired; otherwise, it is a wrong classification.

Fig. 10 shows the training results of the Multi-STIP algorithm and the Multi-ReSuMe algorithm for the Fisher Iris dataset after 100 learning epochs. Fig. 10(a) shows the evolution of the number of misclassified samples on the training set of the Multi-STIP algorithm, finally the number of misclassified samples of setosa, versicolor and virginica is 0, 0.30 and 1.10 respectively. Fig. 10 (b) shows the classification accuracy on the training set of the Multi-STIP algorithm and the Multi-ReSuMe algorithm. We note

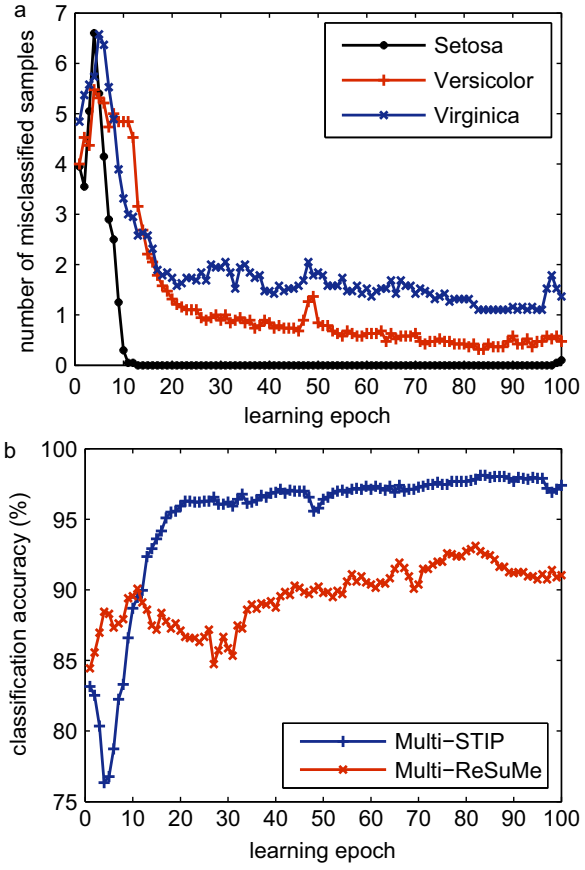

Fig. 10. The training results of the Multi-STIP algorithm and the Multi-ReSuMe algorithm for the Fisher Iris dataset after 100 learning epochs. (a) The evolution of the number of misclassified samples on the training set of the Multi-STIP algorithm. (b) The evolution of classification accuracy on the training set of the Multi-STIP algorithm and the Multi-ReSuMe algorithm.

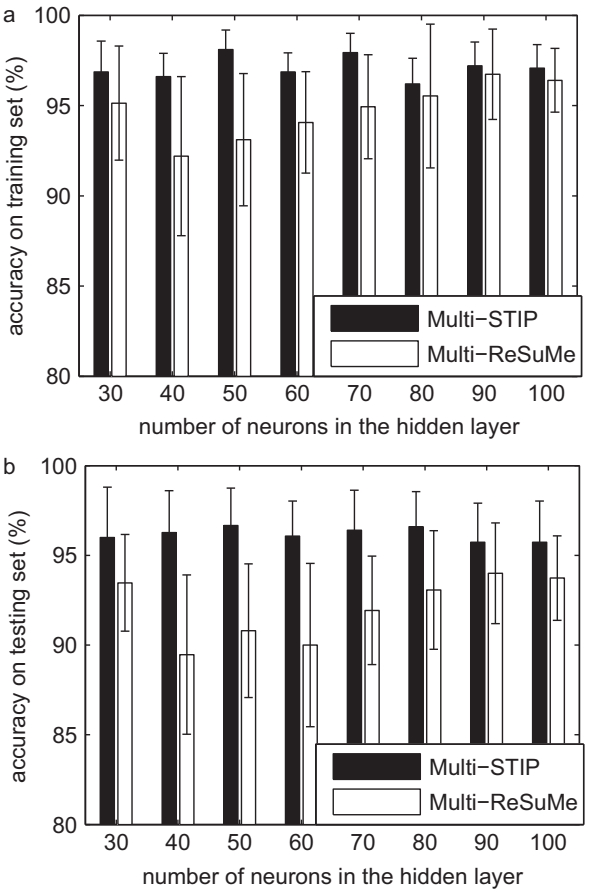

Fig. 11. The learning results of the Multi-STIP algorithm and the Multi-ReSuMe algorithm for the Fisher Iris dataset with different numbers of neurons in the hidden layer. (a) The highest classification accuracy on the training set. (b) The average classification accuracy on the testing set.

that the accuracy of Multi-STIP on the training set is higher than that of Multi-ReSuMe. Finally the classification accuracy on the training set of Multi-STIP is 98.13%, compared with 93.13% for Multi-ReSuMe.

Fig. 11 shows the learning results of the Multi-STIP algorithm and the Multi-ReSuMe algorithm for the Fisher Iris dataset with different numbers of neurons in the hidden layer after 100 learning epochs. Fig. 11(a) shows the highest classification accuracy on the training set while Fig. 11(b) shows the average classification accuracy on the testing set. The number of neurons in the hidden layer is increased from 30 to 100 with an interval of 10. From the learning results we can see that for the Multi-STIP algorithm, classification accuracy on both the training set and the testing set is higher than that of the Multi-ReSuMe algorithm. For example, when the number of neurons in the hidden layer is 40, the classification accuracy on the training set and the testing set for the Multi-STIP algorithm is 96.60% and 96.27% respectively, compared with 92.20% and 89.47% respectively for the Multi-ReSuMe algorithm. When the number of neurons in the hidden layer is 70, classification accuracy on the training set and the testing set for Multi-STIP is 97.93% and 96.40% respectively, which is higher than 94.93% and 91.93% respectively for Multi-ReSuMe.

Table 3 compares the classification accuracy of Multi-STIP against existing algorithms for the Fisher Iris dataset [31]. For this dataset, the classification accuracy is comparable to that of the other approaches. When the number of neurons in the hidden layer is 90, the classification accuracy on the training set and the testing set of the Multi-ReSuMe algorithm is 96.73% and 94.0% respectively. When the number of neurons in the hidden layer is

Table 3
Comparison of training algrithm: results for Fisher Iris dataset.

Algorithm	Training set	Testing set
SpikeProp	97.4% ± 0.1	96.1% ± 0.1
SWAT	95.5% ± 0.6	95.3% ± 3.6
Multi-ReSuMe	96.7% ± 2.5	94.0% ± 2.8
Multi-STIP	98.1% ± 1.1	96.7% ± 2.0

50, Multi-STIP's classification accuracy on the training set and the testing set is 98.13% and 96.67% respectively, which is higher than that of the other approaches such as SpikeProp and SWAT.

5. Discussion and conclusion

Analysis of the experiments in Section 4 indicates that the proposed Multi-STIP method can obtain an ideal learning result when training multilayer SNNs. Through the experimental results, we can see two obvious conclusions. First, our method can learn with high accuracy whether in learning sequences of spikes or solving nonlinear pattern classification problems. Second, the length of desired output spike trains and the number of synaptic inputs are important factors that affect the learning accuracy of the SNNs, while the number of neurons in the hidden layers has little effect. We can see that when the length of the desired output spike trains is short, or the number of synaptic inputs is fewer, both the Multi-STIP and Multi-ReSuMe algorithms can learn with high accuracy. As the length of the desired output spike trains or the number of synaptic inputs gradually increases, the Multi-STIP algorithm has higher learning accuracy than the Multi-ReSuMe algorithm. Compared to the Multi-ReSuMe algorithm, another advantage of the Multi-STIP algorithm is its less learning epoch.

Kernel function is very important for our learning method. In the experiments, we tested Gaussian kernel, Laplacian kernel, α-kernel and inverse multiquadratic kernel because these kernels can obtain ideal learning results. Many other kernels such as linear kernel, polynomial kernel, Sigmoid kernel and multiquadratic kernel are not suitable for our learning method. When using these kernels, learning was poor or learning failed. We also tested Gaussian kernel with different parameters of σ. The adjustment rule of synaptic weight between a neuron in the last hidden layer and a neuron in the output layer is similar to SPAN algorithm [38] and PSD algorithm [40]. Our proposed algorithm can be seen as the extending of SPAN and PSD to multilayer feedforward networks. The derivation of our proposed algorithm is independent of the neuron model; it can be applied to any neuron model. It is also a general framework of supervised learning for SNNs based on inner product operators for spike trains.

In this paper, we introduced a new supervised learning algorithm based on spike train inner products for multilayer SNNs. Our learning algorithm uses the inner products of spike trains to construct the error function and deduce the learning rule. The spike train inner products have been used on similarity measures theory for spike trains, but less used for supervised learning algorithms of SNNs at present [37]. The synaptic weight modification rules only depend on the input, output and target spike trains and do not depend on the specific dynamic of the neuron model. The algorithm is tested on different spatio-temporal tasks, such as learning sequences of spikes and nonlinear pattern classification problems. The experimental results indicate that our method is an effective supervised multi-spike learning algorithm for multilayer SNNs. It can learn spike trains with high accuracy, and also achieves high accuracy for practical classification problems.

Acknowledgments

The work is supported by the National Natural Science Foundation of China under Grants nos. 61165002 and 61363059, the Natural Science Foundation of Gansu Province of China under Grant no. 1506RJZA127 and Scientific Research Project of Universities of Gansu Province under Grant no. 2015A-013.

Appendix A. Details of models and simulations

A.1. SRM model and parameter settings

The spiking neuron model SRM [19] is employed in all simulations. It expresses the membrane potential u at time t as an integral over the past, including a model of refractoriness. Assuming that a neuron has N input synapses, the ith synapse transmits a total of G_i spikes and the gth spike ($g \in [1, G_i]$) is fired at time t_i^g. The internal state $u(t)$ of the neuron at time t is given by:

$$u(t) = \eta\left(t - t^f\right) + \sum_{i=1}^{N}\sum_{g=1}^{G_i} w_i \varepsilon\left(t - t_i^g\right) \tag{A1}$$

where w_i is the weight for the ith synapse. The spike response function $\varepsilon(t)$ is expressed as:

$$\varepsilon(t) = \begin{cases} \frac{t}{\tau}\exp\left(1 - \frac{t}{\tau}\right) & \text{if } t > 0 \\ 0 & \text{if } t \le 0 \end{cases} \tag{A2}$$

where τ is the time constant. In addition, $\eta(t)$ is the refractoriness function, which is mainly reflected in the effect that only the last output spike t^f contributes to refractoriness:

$$\eta(t) = \begin{cases} -\vartheta \exp\left(-\frac{t}{\tau_R}\right) & \text{if } t > 0 \\ 0 & \text{if } t \le 0 \end{cases} \tag{A3}$$

where ϑ is the neuron threshold and τ_R is the time constant.

The parameter values of the SRM neurons used in the experiments are: the time constant of postsynaptic potential $\tau = 5$ ms, the time constant of refractory period $\tau_R = 50$ ms, the neuron threshold $\vartheta = 1$, and the length of the absolute refractory period $t_R = 1$ ms.

A.2. Learning rules of the Multi-ReSuMe algorithm

The learning rules of the Multi-ReSuMe algorithm can be interpreted as the biologically plausible STDP and anti-STDP processes [32,34]. The synaptic weight modifications at time t for the output neurons are expressed as:

$$\begin{aligned}\Delta w_{oh}(t) = \frac{1}{N_H} s_h(t)\left[\int_\Gamma a^{pre}(s)\left[s_o^d(t-s) - s_o^a(t-s)\right]ds\right] \\ + \frac{1}{N_H}\left[s_o^d(t) - s_o^a(t)\right]\left[a + \int_\Gamma a^{post}(s) s_h(t-s)ds\right]\end{aligned} \tag{A4}$$

where N_H is the number of hidden neurons and $a = 0.05$ is a non-Hebbian term. The synaptic weight modifications at time t for the hidden neurons are expressed as:

$$\begin{aligned}\Delta w_{hi}(t) = \frac{1}{N_I N_H} s_i(t) \sum_{o=1}^{N_O}\left[\int_\Gamma a^{pre}(s)[s_o^d(t-s) - s_o^a(t-s)]ds\right] w_{oh} \\ + \frac{1}{N_I N_H}\sum_{o=1}^{N_O}\left[s_o^d(t) - s_o^a(t)\right]\left[a + \int_\Gamma a^{post}(s) s_i(t-s)ds\right] w_{oh}\end{aligned} \tag{A5}$$

where N_I and N_O are the numbers of input neurons and output

neurons respectively. The kernels a^{pre} and a^{post} define the learning window $W(s)$:

$$W(s) = \begin{cases} a^{pre}(-s) = -A_- \exp\left(\frac{s}{\tau_-}\right) & \text{if } s \le 0 \\ a^{post}(s) = +A_+ \exp\left(\frac{-s}{\tau_+}\right) & \text{if } s > 0 \end{cases} \tag{A6}$$

where $A_+ = 1.2$, $A_- = 0.5$ are the amplitudes and $\tau_+ = 5$ ms, $\tau_- = 5$ ms are the time constants of the learning window.

References

[1] S.S. Haykin, Neural Networks and Learning Machines, 3, Pearson Education, Upper Saddle River, 2009.
[2] D.E. Rummelhart, Learning representations by back-propagating errors, Nature 323 (9) (1986) 533–536.
[3] Y. Chauvin, D.E. Rumelhart, Backpropagation: theory, architectures, and applications, Psychology Press, 1995.
[4] S.M. Bohte, The evidence for neural information processing with precise spike-times: a survey, Nat. Comput. 3 (2) (2004) 195–206.
[5] K. Whalley, Neural coding: timing is key in the olfactory system, Nat. Rev. Neurosci. 14 (7) (2013), 458–458.
[6] E.M. Izhikevich, Which model to use for cortical spiking neurons? IEEE Trans. Neural Netw. 15 (5) (2004) 1063–1070.
[7] X. Lin, T. Zhang, Dynamical properties of piecewise linear spiking neuron model, Acta Electron. Sin. 37 (6) (2009) 1270–1276.
[8] W. Maass, Networks of spiking neurons: the third generation of neural network models, Neural Netw. 10 (9) (1997) 1659–1671.
[9] S. Ghosh-Dastidar, H. Adeli, Spiking neural networks, Int. J. Neural Syst. 19 (4) (2009) 295–308.
[10] W. Maass, Lower bounds for the computational power of networks of spiking neurons, Neural Comput. 8 (1) (1996) 1–40.
[11] W. Maass, Fast sigmoidal networks via spiking neurons, Neural Comput. 9 (2) (1997) 279–304.
[12] A.D. Almási, S. Woźniak, V. Cristea, Y. Leblebici, T. Engbersen, Review of advances in neural networks: neural design technology stack, Neurocomputing 174 (2016) 31–41.
[13] E.I. Knudsen, Supervised learning in the brain, J. Neurosci. 14 (7) (1994) 3985–3997.
[14] E.I. Knudsen, Instructed learning in the auditory localization pathway of the barn owl, Nature 417 (6886) (2002) 322–328.
[15] S. Guediche, L.L. Holt, P. Laurent, S.J. Lim, J.A. Fiez, Evidence for cerebellar contributions to adaptive plasticity in speech perception, Cerebral Cortex, 2014, bht428.
[16] X. Lin, X. Wang, N. Zhang, H. Ma, Supervised learning algorithms for spiking neural networks: a review, Acta Electron. Sin. 43 (3) (2015) 577–586.
[17] R.M. Memmesheimer, R. Rubin, B.P. Ölveczky, H. Sompolinsky, Learning precisely timed spikes, Neuron 82 (4) (2014) 925–938.
[18] S.M. Bohte, J.N. Kok, H.La. Poutre, Error-backpropagation in temporally encoded networks of spiking neurons, Neurocomputing 48 (1) (2002) 17–37.
[19] W. Gerstner, W.M. Kistler, Spiking neuron models: single Neurons, populations, plasticity, Cambridge University Press, 2002.
[20] J. Xin, M.J. Embrechts, Supervised learning with spiking neural networks, in: International Joint Conference on Neural Networks, IEEE, 2001, vol. 3, pp. 1772–1777.
[21] S. McKennoch, D. Liu, L.G. Bushnell, Fast modifications of the spikeprop algorithm, in: International Joint Conference on Neural Networks, IEEE, 2006, pp. 3970–3977.
[22] S. McKennoch, T. Voegtlin, L. Bushnell, Spike-timing error backpropagation in theta neuron networks, Neural Comput. 21 (1) (2009) 9–45.
[23] H. Fang, J. Luo, F. Wang, Fast learning in spiking neural networks by learning rate adaptation, Chin. J. Chem. Eng. 20 (6) (2012) 1219–1224.
[24] O. Booij, H. tat Nguyen, A gradient descent rule for spiking neurons emitting multiple spikes, Inf. Process. Lett. 95 (6) (2005) 552–558.
[25] S. Ghosh-Dastidar, H. Adeli, A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection, Neural Netw. 22 (10) (2009) 1419–1431.
[26] Y. Xu, X. Zeng, L. Han, J. Yang, A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks, Neural Netw. 43 (2013) 99–113.
[27] R.V. Florian, The chronotron: a neuron that learns to fire temporally precise spike patterns, PLoS One 7 (8) (2012) e40233.
[28] E. Kuriscak, P. Marsalek, J. Stroffek, P.G. Toth, Biological context of Hebb learning in artificial neural networks,a review, Neurocomputing 152 (2015) 27–35.
[29] N. Caporale, Y. Dan, Spike timing-dependent plasticity: a hebbian learning rule, Annu. Rev. Neurosci. 31 (2008) 25–46.
[30] R. Legenstein, C. Naeger, W. Maass, What can a neuron learn with spike-timing-dependent plasticity? Neural Comput. 17 (11) (2005) 2337–2382.
[31] J.J. Wade, L.J. McDaid, J.A. Santos, H.M. Sayers, SWAT: a spiking neural network training algorithm for classification problems, IEEE Trans. Neural Netw. 21 (11) (2010) 1817–1830.
[32] F. Ponulak, A. Kasinski, Supervised learning in spiking neural networks with resume: sequence learning, classification, and spike shifting, Neural Comput. 22 (2) (2010) 467–510.
[33] F. Ponulak, Analysis of the ReSuMe learning process for spiking neural networks, Int. J. Appl. Math. Comput. Sci. 18 (2) (2008) 117–127.
[34] I. Sporea, A. Grüning, Supervised learning in multilayer spiking neural networks, Neural Comput. 25 (2) (2013) 473–509.
[35] P. Dayan, L.F. Abbott, Theoretical neuroscience: computational and mathematical modeling of neural systems, J. Cognit. Neurosci. 15 (1) (2003) 154–155.
[36] I.M. Park, S. Seth, A. Paiva, L. Li, J. Principe, Kernel methods on spike train space for neuroscience: a tutorial, IEEE Signal Process. Mag. 30 (4) (2013) 149–160.
[37] A. Carnell, D. Richardson, linear algebra for time series of spikes, in: ESANN, 2005, pp. 363–368.
[38] A. Mohemmed, S. Schliebs, S. Matsuda, N. Kasabov, Span: spike pattern association neuron for learning spatio-temporal spike patterns, Int. J. Neural Syst. 22 (4) (2012) 786–803.
[39] A. Mohemmed, S. Schliebs, S. Matsuda, N. Kasabov, Training spiking neural networks to associate spatio-temporal input-output spike patterns, Neurocomputing 107 (2013) 3–10.
[40] Q. Yu, H. Tang, K.C. Tan, H. Li, Precise-spike-driven synaptic plasticity: learning hetero-association of spatiotemporal spike patterns, PLoS One 8 (11) (2013) e78318.
[41] Q. Yu, H. Tang, K.C. Tan, H. Yu, A brain-inspired spiking neural network model with temporal encoding and learning, Neurocomputing 138 (2014) 3–13.
[42] I.M. Park, S. Seth, M. Rao, J.C. Príncipe, Strictly positive-definite spike train kernels for point-process divergences, Neural Comput. 24 (8) (2012) 2223–2250.
[43] A.R. Paiva, I. Park, J.C. Príncipe, A reproducing kernel hilbert space framework for spike train signal processing, Neural Comput. 21 (2) (2009) 424–449.
[44] R. Gütig, R. Aharonov, S. Rotter, H. Sompolinsky, Learning input correlations through nonlinear temporally asymmetric hebbian plasticity, J. Neurosci. 23 (9) (2003) 3697–3714.
[45] S. Cash, R. Yuste, Linear summation of excitatory inputs by ca1 pyramidal neurons, Neuron 22 (2) (1999) 383–394.
[46] N. Brunel, V. Hakim, P. Isope, J.P. Nadal, B. Barbour, Optimal information storage and the distribution of synaptic weights: perceptron versus purkinje cell, Neuron 43 (5) (2004) 745–757.

Xianghong Lin received the B.Eng. degree in Computer Science and Technology from Northwest Normal University, Lanzhou, China, in 1998. He received the M.Eng. and Ph.D. degrees in Computer Science and Technology from Harbin Institute of Technology, Harbin, China, in 2004 and 2009, respectively. He is currently a Professor with the School of Computer Science and Engineering, Northwest Normal University, Lanzhou, China. His current research interests include neural networks, evolutionary computation, artificial life and image processing.

Xiangwen Wang received the B.Eng. degree in Computer Science and Technology, and M.Eng. degree in Software Engineering from Northwest Normal University, Lanzhou, China, in 2013 and 2015, respectively. He is currently an Assistant Engineer with the School of Computer Science and Engineering, Northwest Normal University, Lanzhou, China. His current research interests include neural networks and machine learning

Zhanjun Hao received the B.Eng. and M.Eng. degrees in Computer Science and Technology from Xidian University, Xian, China and Northwest Normal University, Lanzhou, China, in 2002 and 2011, respectively. He is currently a Senior Engineer with the School of Computer Science and Engineering, Northwest Normal University, Lanzhou, China. His current research interests include wireless sensor network and intelligent computation.