THE PUBLISHING HOUSE
OF THE ROMANIAN ACADEMY

PROCEEDINGS OF THE ROMANIAN ACADEMY, Series A,
Volume 26, Number 4/2025, pp. 417–427
DOI: 10.59277/PRA-SER.A.26.4.10

SYNAPTIC DELAY LEARNING ALGORITHM FOR FEEDFORWARD SPIKING NEURAL NETWORKS BASED ON SPIKE TRAIN KERNEL

Xiangwen WANG[1,2], Xun LI[2], Li ZOU[3], Jingxing FAN[1]

[1] Gansu Minzu Normal University, College of Information Engineering, Hezuo, 747000, China
[2] Northwest Normal University, College of Computer Science and Engineering, Lanzhou, 730070, China
[3] Gansu Provincial Meteorological Information and Technical Equipment Support Center, Lanzhou, 730020, China
Corresponding author: Xiangwen WANG, E-mail: wangxw2015@nwnu.edu.cn

Abstract. In the study of spiking neural networks (SNNs), synaptic delay emerges as a critical factor influencing the transmission of information between neurons, significantly affecting the dynamic properties and information processing capabilities of the network. However, traditional SNN models frequently overlook the learning and optimization of synaptic delays, thereby constraining their capacity to process complex spike train data. To address this limitation, we propose a synaptic delay learning algorithm for multilayer feedforward SNNs that is based on spike train kernels, including both online and offline learning rules. The algorithm employs spike train kernels to characterize the temporal dependencies among neurons and incorporates synaptic delays as learnable parameters, thereby enhancing the spatiotemporal information processing capabilities of SNNs. The results demonstrate that the proposed algorithm achieves reduced network error in spike train learning tasks with fewer learning epochs and attains high classification accuracy in tasks involving UCI datasets. These findings indicate that the integration of synaptic delay learning can significantly enhance the performance of SNNs in learning spatiotemporal patterns.

Keywords: feedforward spiking neural network, supervised learning, synaptic delay learning, spike train kernel.

1. INTRODUCTION

Inspired by biological neural systems, spiking neural networks (SNNs) have emerged as a distinctive class of brain-like computational models. With the rapid development of deep learning technology, significant progress has been made in the training strategies of SNNs and their applications in complex spatiotemporal pattern recognition tasks [1]. SNNs utilize spike trains to encode and transmit neural information, and this information processing mode is also widely applied in spiking neural P systems [2]. These models offer both biological interpretability and computational efficiency [3]. Within the framework of supervised learning, SNNs facilitate accurate prediction and inference of output data by establishing a precise mapping relationship between input and output spike trains, while iteratively optimizing the network's learnable parameters. In recent years, researchers have developed various supervised learning algorithms for SNNs tailored to different network topologies. These algorithms predominantly employ mechanisms such as error backpropagation and spike-timing dependent plasticity (STDP) to formulate learning rules for synaptic weights [4].

Synaptic delay plasticity refers to the variability in the time interval between the firing of spikes by pre-synaptic neurons and the reception of these spikes by post-synaptic neurons, a variability that can be dynamically adjusted in response to neuronal activity patterns and the learning processes of the network. This mechanism is crucial for optimizing the transmission timing of spike signals and enhancing the efficiency of information processing within the network [5]. However, the research on supervised learning in traditional SNNs has predominantly concentrated on synaptic weight plasticity, often overlooking the significance of synaptic delay plasticity. In recent years, recognizing the importance of synaptic delay plasticity, researchers have developed several supervised learning algorithms that incorporate this aspect into SNNs. For single-layer SNNs, learnable synaptic delays have been integrated into existing synaptic weight learning algorithms, leading to the construction of corresponding delay learning mechanisms. Notable examples of these synaptic delay learning algorithms include the ReSuMe-based algorithm [6] and the spike train kernel-based algorithm [7]. In the context of multilayer feedforward SNNs, researchers have proposed synaptic delay learning algorithms that leverage the STDP mechanism [8] as well as gradient descent optimization techniques [9]. The results demonstrate that the incorporation of learnable synaptic delays can significantly enhance the learning performance of SNNs.

Despite notable advancements in synaptic delay learning, current research on synaptic delay learning algorithms for feedforward SNNs continues to encounter significant challenges. Most of the existing algorithms are derived using the gradient descent rule based on spike times, which lacks biological interpretability and is closely related to specific spiking neuron models. To overcome this limitation, researchers have used the kernel function representation of spike trains to convert discrete spikes into continuous functions, and then applied approaches such as the Widrow-Hoff rule and gradient descent to derive learning rules for synaptic weights and delays [7]. The kernel function representation of spike trains can interpret the spike signals as specific neurophysiological signals, such as the postsynaptic potentials of neurons or the density function of spike firing, which enhances the biological interpretability of supervised learning algorithms. Furthermore, a unified representation of spike trains can be established within the framework of reproducing kernel Hilbert space, which facilitates the formal definition of spike train similarity metrics [10]. While several synaptic weight learning algorithms for SNNs based on spike train kernels have been proposed [11, 12], the development of synaptic delay learning algorithms for feedforward SNNs remains inadequate and warrants further in-depth investigation.

In light of the limitations inherent in current synaptic delay learning algorithms for feedforward SNNs, and recognizing the distinct advantages offered by spike train kernel function representations in the development of supervised learning algorithms for SNNs, this paper proposes a novel supervised learning algorithm for synaptic delays in multilayer feedforward SNNs based on spike train kernels. The primary contributions of this work are summarized as follows: (1) We propose a biologically interpretable synaptic delay learning algorithm for multilayer feedforward SNNs that simultaneously simulates the mechanisms of synaptic weight plasticity and synaptic delay plasticity observed in biological neural systems. (2) We construct a supervised learning framework based on spike train kernels, which effectively learns and characterizes complex spatiotemporal patterns within spike trains through both online and offline learning manner. (3) The derivation process of the proposed algorithm is independent of the specific mathematical formulation of the internal states of neurons, thereby demonstrating broad applicability across various neuron models.

The remainder of this paper is structured as follows. Section 2 provides a detailed derivation of the proposed delay learning algorithm. Section 3 demonstrates the learning performance of the proposed algorithm through spike train learning tasks and pattern recognition tasks utilizing UCI datasets. Finally, Section 4 presents a discussion and concludes the findings of this study.

2. METHODOLOGY

This section first introduces the spike train and its kernel function representation, followed by the derivation of online and offline learning rules for synaptic weights and synaptic delays. Finally, pseudo-code for online and offline supervised learning of multilayer feedforward SNNs based on the proposed synaptic weights and synaptic delays updating rules is provided.

2.1. Spike train and its kernel function representation

All spikes fired by a neuron during the simulation time interval T constitute a spike train, which serves as the fundamental unit of information transfer in SNNs. Taking into account the synaptic delay $d(t)$ at time t, the spike train can be formally defined as follows:

$$s\left(t,d(t)\right)=\sum_{f=1}^{N}\delta\left(t,t^{f}+d(t)\right) \tag{1}$$

where t^f denotes the f-th spike and N represents the total number of spikes. The function $\delta(x,y)$ is the Dirac delta function, defined such that $\delta(x,y)=1$ when $x=y$ and $\delta(x,y)=0$ otherwise.

In this paper, we employ the kernel function representation of spike trains to develop a supervised learning algorithm for feedforward SNNs. By selecting a specific kernel function $\kappa(x,y)$ and applying convolution operations, the discrete spike train can be transformed into a continuous function, thereby facilitating subsequent analysis and processing. The continuous function corresponding to the spike train $s(t, d(t))$ can be expressed as follows:

$$f_s\left(t,d(t)\right)=s\left(t,d(t)\right)\otimes\kappa\left(t,d(t)\right)=\sum_{f=1}^{N}\kappa\left(t,t^f+d(t)\right) \tag{2}$$

where $\otimes$ denotes the convolution operation. Furthermore, for any two spike trains $s_i(t,d_i(t))$ and $s_j(t,d_j(t))$ with synaptic delays $d_i(t)$ and $d_j(t)$, the inner product of corresponding continuous functions $f_{s_i}(t,d_i(t))$ and $f_{s_j}(t,d_j(t))$ on the $L_2(T)$ space can be defined as follows:

$$\begin{aligned}F\left(s_i(t,d_i(t)),s_j(t,d_j(t))\right)&=\left\langle f_{s_i}\left(t,d_i(t)\right),f_{s_j}\left(t,d_j(t)\right)\right\rangle_{L_2(T)}\\&=\int_T f_{s_i}\left(t,d_i(t)\right)\cdot f_{s_j}\left(t,d_j(t)\right)\mathrm{d}t=\sum_{a=1}^{N_i}\sum_{b=1}^{N_j}\kappa\left(t_i^a+d_i,t_j^b+d_j\right)\end{aligned} \tag{3}$$

where t_i^a and t_j^b denote the spikes, while N_i and N_j denote the number of spikes in the spike trains $s_i(t,d_i(t))$ and $s_j(t,d_j(t))$, respectively.

The primary advantage of representing discrete spike trains as continuous functions is that it allows for the straightforward definition of a spike train-level error function, which is essential for employing the gradient descent rule to derive update rules for synaptic weights and synaptic delays in feedforward SNNs. By utilizing the continuous functions corresponding to the actual output spike train $s_o(t)$ and the desired output spike train $\hat{s}_o(t)$, the error function of the feedforward SNNs can be defined as follows:

$$E=\int_T E(t)\mathrm{d}t=\int_T\frac{1}{2}\sum_{o=1}^{N_O}\left[f_{s_o}(t)-f_{\hat{s}_o}(t)\right]^2\mathrm{d}t=\frac{1}{2}\sum_{o=1}^{N_O}\left[F\left(s_o(t),s_o(t)\right)-2F\left(s_o(t),\hat{s}_o(t)\right)+F\left(\hat{s}_o(t),\hat{s}_o(t)\right)\right] \tag{4}$$

where N_O is the number of output neurons.

The second advantage of this representation lies in the ability to conveniently depict the relationship between presynaptic and postsynaptic spike trains, which is crucial for the design of supervised learning algorithms for feedforward SNNs. Based on the architecture of the feedforward SNN model, the continuous functions corresponding to the actual output spike train $s_o(t)$ and the hidden spike train $s_h(t,d_{ho}(t))$ can be expressed as follows:

$$f_{s_o}(t)=\sum_{h=1}^{N_H}w_{ho}(t)f_{s_h}\left(t,d_{ho}(t)\right) \tag{5}$$

$$f_{s_h}\left(t,d_{ho}(t)\right)=\sum_{i=1}^{N_I}w_{ih}(t)f_{s_i}\left(t,d_{ih}(t)\right) \tag{6}$$

where N_I and N_H denote the number of input and hidden neurons, respectively. $w_{ho}(t)$ and $d_{ho}(t)$ are the synaptic weight and delay between the hidden and output neurons. $w_{ih}(t)$ and $d_{ih}(t)$ are the synaptic weight and delay between the input and hidden neurons, respectively. $f_{s_h}(t,d_{ho}(t))$ and $f_{s_i}(t,d_{ih}(t))$ are continuous functions corresponding to the hidden spike train $s_h(t,d_{ho}(t))$ and the input spike train $s_i(t,d_{ih}(t))$, respectively.

2.2. Synaptic weight learning rule

By employing the gradient descent rule and the chain rule, the change in synaptic weight $\Delta w_{ho}(t)$ between the hidden neuron h and the output neuron o at time t can be computed as the gradient of the error function $E(t)$ with respect to the synaptic weight $w_{ho}(t)$ scaled by the learning rate for synaptic weights η_w:

$$\Delta w_{ho}(t) = -\eta_w \frac{\partial E(t)}{\partial w_{ho}(t)} = -\eta_w \frac{\partial E(t)}{\partial f_{s_o}(t)} \frac{\partial f_{s_o}(t)}{\partial w_{ho}(t)} \tag{7}$$

Based on the error function defined in Eq. (4), the first partial derivative term on the right side of Eq. (7) can be computed as follows:

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

Utilizing the linear relationship between the output and hidden spike trains as defined in Eq. (5), the second derivative term on the right side of Eq. (7) can be calculated as follows:

$$\frac{\partial f_{s_o}(t)}{\partial w_{ho}(t)} = \frac{\partial \left[\sum_{h=1}^{N_H} w_{ho}(t) f_{s_h}\left(t, d_{ho}(t)\right) \right]}{\partial w_{ho}(t)} = f_{s_h}\left(t, d_{ho}(t)\right) \tag{9}$$

By substituting Eqs. (8) and (9) into Eq. (7), we can derive the online learning rule for the synaptic weights between the hidden and output neurons, which is expressed as follows:

$$\Delta w_{ho}(t) = \eta_w \left[f_{\hat{s}_o}(t) - f_{s_o}(t) \right] f_{s_h}\left(t, d_{ho}(t)\right) \tag{10}$$

Further integrating the online learning rule in Eq. (10), we can derive the offline learning rule for the synaptic weights between the hidden and output neurons, which is expressed as follows:

$$\begin{aligned} \Delta w_{ho} &= \int_T \Delta w_{ho}(t) \mathrm{d}t = \eta_w \left[F\left(\hat{s}_o(t), s_h\left(t, d_{ho}(t)\right)\right) - F\left(s_o(t), s_h\left(t, d_{ho}(t)\right)\right) \right] \\ &= \eta_w \left[\sum_{n=1}^{\hat{N}_o} \sum_{g=1}^{N_h} \kappa\left(\hat{t}_o^n, t_h^g + d_{ho}\right) - \sum_{m=1}^{N_o} \sum_{g=1}^{N_h} \kappa\left(t_o^m, t_h^g + d_{ho}\right) \right] \end{aligned} \tag{11}$$

where: t_h^g, t_o^m and $\hat{t}_o^n$ are spikes in $s_h(t,d_{ho}(t))$, $s_o(t)$, and $\hat{s}_o(t)$, respectively. N_h, N_o and $\hat{N}_o$ are the total number of spikes in the corresponding spike train.

By applying the gradient descent rule in conjunction with the chain rule, the change in synaptic weight $\Delta w_{ih}(t)$ between the input neuron i and the hidden neuron h at time t can be computed as the gradient of the error function $E(t)$ with respect to the synaptic weight $w_{ih}(t)$ scaled by the learning rate η_w:

$$\Delta w_{ih}(t) = -\eta_w \frac{\partial E(t)}{\partial w_{ih}(t)} = -\eta_w \sum_{o=1}^{N_O} \frac{\partial E(t)}{\partial f_{s_o}(t)} \frac{\partial f_{s_o}(t)}{\partial f_{s_h}\left(t, d_{ih}(t)\right)} \frac{\partial f_{s_h}\left(t, d_{ih}(t)\right)}{\partial w_{ih}(t)} \tag{12}$$

The first partial derivative term on the right side of Eq. (12) is calculated according to Eq. (8). Utilizing Eq. (5), the second derivative term on the right side of Eq. (12) can be calculated as follows:

$$\frac{\partial f_{s_o}(t)}{\partial f_{s_h}\left(t, d_{ih}(t)\right)} = \frac{\partial \left[\sum_{h=1}^{N_H} w_{ho}(t) f_{s_h}(t, d_{ho}(t)) \right]}{\partial f_{s_h}\left(t, d_{ih}(t)\right)} = w_{ho}(t) \tag{13}$$

According to Eq. (6), the third derivative term on the right side of Eq. (12) can be calculated as follows:

$$\frac{\partial f_{s_h}\left(t, d_{ih}(t)\right)}{\partial w_{ih}(t)} = \frac{\partial \left[\sum_{i=1}^{N_I} w_{ih}(t) f_{s_i}\left(t, d_{ih}(t)\right) \right]}{\partial w_{ih}(t)} = f_{s_i}\left(t, d_{ih}(t)\right) \tag{14}$$

By substituting Eqs. (8), (13) and (14) into Eq. (12), we can derive the online learning rule for the synaptic weights between the input and hidden neurons, which is expressed as follows:

$$\Delta w_{ih}(t) = \eta_w \sum_{o=1}^{N_O} \left[f_{\hat{s}_o}(t) - f_{s_o}(t) \right] f_{s_i}\left(t, d_{ih}(t)\right) w_{ho}(t) \tag{15}$$

Further integrating the online learning rule in Eq. (15), we can derive the offline learning rule for the synaptic weights between the input and hidden neurons, which is expressed as follows:

$$\begin{aligned} \Delta w_{ih} &= \int_T \Delta w_{ih}(t)\mathrm{d}t = \eta_w \sum_{o=1}^{N_O} \left[F\left(\hat{s}_o(t), s_i\left(t, d_{ih}(t)\right)\right) - F\left(s_o(t), s_i\left(t, d_{ih}(t)\right)\right) \right] w_{ho} \\ &= \eta_w \sum_{o=1}^{N_O} \left[\sum_{n=1}^{\hat{N}_o} \sum_{g=1}^{N_h} \kappa\left(\hat{t}_o^n, t_i^f + d_{ih}\right) - \sum_{m=1}^{N_o} \sum_{g=1}^{N_h} \kappa\left(t_o^m, t_i^f + d_{ih}\right) \right] w_{ho} \end{aligned} \tag{16}$$

where t_i^f and N_i are the spike and the total number of spikes in $s_i(t, d_{ih}(t))$, respectively.

2.3. Synaptic delay learning rule

Similar to the derivation of the synaptic weight learning rule, the change in synaptic delays $\Delta d_{ho}(t)$ between the hidden neuron h and the output neuron o at time t can be computed as the gradient of the error function $E(t)$ with respect to the synaptic delays $d_{ho}(t)$ scaled by the learning rate for synaptic delays η_d:

$$\Delta d_{ho}(t) = -\eta_d \frac{\partial E(t)}{\partial d_{ho}(t)} = -\eta_d \frac{\partial E(t)}{\partial f_{s_o}(t)} \frac{\partial f_{s_o}(t)}{\partial d_{ho}(t)} \tag{17}$$

The first partial derivative term on the right side of Eq. (17) is calculated according to Eq. (8). Utilizing Eq. (5), the second derivative term on the right side of Eq. (17) can be calculated as follows:

$$\frac{\partial f_{s_o}(t)}{\partial d_{ho}(t)} = \frac{\partial \left[\sum_{h=1}^{N_H} w_{ho}(t) f_{s_h}\left(t, d_{ho}(t)\right) \right]}{\partial d_{ho}(t)} = w_{ho}(t) \frac{\partial \left[\sum_{g=1}^{N_h} \kappa\left(t, t_h^g + d_{ho}(t)\right) \right]}{\partial d_{ho}(t)} \tag{18}$$

For simplicity, the Laplacian kernel function $\kappa(x,y)=\exp(-|x-y|/\tau_k)$ is employed to compute the partial derivative term on the right side of Eq. (18), which can be calculated as follows:

$$\frac{\partial \left[\sum_{g=1}^{N_h} \kappa\left(t, t_h^g + d_{ho}(t)\right) \right]}{\partial d_{ho}(t)} = \frac{1}{\tau_k} \sum_{g=1}^{N_h} \exp\left(-\frac{\left| t - t_h^g - d_{ho}(t) \right|}{\tau_k} \right) = \frac{1}{\tau_k} f_{s_h}\left(t, d_{ho}(t)\right) \tag{19}$$

By substituting Eqs. (8), (18), and (19) into Eq. (17), we can derive the online learning rule for synaptic delays between the hidden and output neurons, which is expressed as follows:

$$\Delta d_{ho}(t) = \eta_d \frac{1}{\tau_k} w_{ho}(t) \left[f_{\hat{s}_o}(t) - f_{s_o}(t) \right] f_{s_h}\left(t, d_{ho}(t)\right) = \eta_d \frac{1}{\tau_k} \frac{1}{\eta_w} w_{ho}(t) \Delta w_{ho}(t) \tag{20}$$

Further integrating the online learning rule in Eq. (20), we can derive the offline learning rule for synaptic delays between the hidden and output neurons, which is expressed as follows:

$$\Delta d_{ho} = \int_T \Delta d_{ho}(t)\mathrm{d}t = \eta_d \frac{1}{\tau_k} \frac{1}{\eta_w} w_{ho} \Delta w_{ho} \tag{21}$$

By employing the gradient descent rule and the chain rule, the change in synaptic delays $\Delta d_{ih}(t)$ between the input neuron i and the hidden neuron h at time t can be calculated as the gradient of the error function $E(t)$ with respect to the synaptic delays $d_{ih}(t)$ scaled by the learning rate η_d:

$$\Delta d_{ih}(t) = -\eta_d \frac{\partial E(t)}{\partial d_{ih}(t)} = -\eta_d \sum_{o=1}^{N_O} \frac{\partial E(t)}{\partial f_{s_o}(t)} \frac{\partial f_{s_o}(t)}{\partial f_{s_h}\left(t, d_{ih}(t)\right)} \frac{\partial f_{s_h}\left(t, d_{ih}(t)\right)}{\partial d_{ih}(t)} \tag{22}$$

The first two partial derivative terms on the right side of Eq. (22) are calculated according to Eqs. (8) and (13), respectively. According to Eq. (6), the third partial derivative term on the right side of Eq. (22) can be calculated as follows:

$$\frac{\partial f_{s_h}\left(t, d_{ih}(t)\right)}{\partial d_{ih}(t)} = \frac{\partial \left[\sum_{i=1}^{N_I} w_{ih}(t) f_{s_i}\left(t, d_{ih}(t)\right) \right]}{\partial d_{ih}(t)} = \frac{\partial \left[\sum_{i=1}^{N_I} w_{ih}(t) \sum_{f=1}^{N_i} \kappa\left(t, t_i^f + d_{ih}(t)\right) \right]}{\partial d_{ih}(t)} = w_{ih}(t) \frac{1}{\tau_k} f_{s_i}\left(t, d_{ih}(t)\right) \tag{23}$$

By substituting Eqs. (8), (13) and (23) into Eq. (22), we derive the online learning rule for synaptic delays between the input and hidden neurons, which is expressed as follows:

$$\Delta d_{ih}(t) = \eta_d w_{ho}(t) w_{ih}(t) \frac{1}{\tau_k} \sum_{o=1}^{N_O} \left[f_{\hat{s}_o}(t) - f_{s_o}(t) \right] f_{s_i}\left(t, d_{ih}(t)\right) = \eta_d \frac{1}{\tau_k} \frac{1}{\eta_w} w_{ih}(t) \Delta w_{ih}(t) \tag{24}$$

Further integrating the online learning rule in Eq. (24), we can derive the offline learning rule for synaptic delays between the input and hidden neurons, which is expressed as follows:

$$\Delta d_{ih} = \int_T \Delta d_{ih}(t)\,\mathrm{d}t = \eta_d \frac{1}{\tau_k} \frac{1}{\eta_w} w_{ih} \Delta w_{ih} \tag{25}$$

Since η_d, τ_k and η_w in Eqs. (20) and (24) are constants set before training the network, we assume that $\eta_d = \eta_d/(\tau_k \cdot \eta_w)$.

2.4. Supervised learning for feedforward SNNs

Figure 1 illustrates the process of training a multilayer feedforward SNN utilizing the proposed online and offline supervised learning algorithm for synaptic weights and delays. For ease of description, the online and offline learning algorithms for weight-delay co-learning are named FFOnDD and FFOffDD, respectively, while the online and offline learning algorithms for weight-only learning are named FFOnSD and FFOffSD. The primary difference between offline learning algorithms (FFOffSD and FFOffDD) and online learning algorithms (FFOnSD and FFOnDD) lies in the timing of updating synaptic weights and delays. The offline learning algorithms FFOffSD and FFOffDD update synaptic weights and delays after a complete learning epoch, meaning that network parameters are updated only once per learning epoch. In contrast, the online learning algorithms FFOnSD and FFOnDD update synaptic weights and delays immediately upon encountering an actual or desired output spike, resulting in multiple updates of network parameters within a learning epoch. For weight-only learning algorithms (FFOnSD and FFOffSD) and weight-delay co-learning algorithms (FFOnDD and FFOffDD), the primary difference lies in the parameters updated during the learning process. The weight-only learning algorithms FFOnSD and FFOffSD update only the network's synaptic weights during the learning process, while the synaptic delays remain unchanged. In contrast, the weight-delay co-learning algorithms FFOnDD and FFOffDD update both the network's synaptic weights and delays simultaneously during the learning process. Specifically, the learning process of FFOffDD is shown in Fig. 1a, while FFOffSD only calculates Δw_{ho} and Δw_{ih} according to Eqs. (11) and (16) in step 10 of Fig. 1a, and updates the network's synaptic weights in step 11. Similarly, the learning process of FFOnDD is shown in Fig. 1b, while FFOnSD only calculates $\Delta w_{ho}(t)$ and $\Delta w_{ih}(t)$ according to Eqs. (10) and (15) in step 12 of Fig. 1b, and updates the network's synaptic weights in step 13.

The process of training a multilayer feedforward SNN utilizing the proposed algorithm is as follows. Initially, all parameters of the SNN are initialized, which include the spiking neuron model and its parameters, input spike trains, desired output spike trains, as well as synaptic weights and delays. Subsequently, based on the input spike trains and the spiking neuron model, the hidden spike trains and the actual output spike trains of the SNN are computed. The kernel function representation of the spike trains is then employed to calculate the continuous functions corresponding to all spike trains. Finally, the network error is determined by evaluating the difference between the actual and desired output spike trains, and the weights and delays of all synapses are adjusted according to the proposed update rules for synaptic weights and delays. The learning process will continue until the network error is reduced to zero or the preset maximum learning epoch is reached, thus ending the training process.

```
(a) FFOffDD: Offline Supervised Learning Algorithm
Input: Untrained SNN
Output: Trained SNN
1.  Set up a multilayer feedforward SNN
2.  Initialize all synaptic weights and delays
3.  Initialize input spike trains $s_i(t,d_{ih}(t))$ and desired output spike trains $\hat{s}_o(t)$
4.  Calculate continuous functions $f_{s_i}(t,d_{ih}(t))$ and $f_{\hat{s}_o}(t)$ using Eq. (2)
5.  repeat
6.    Input all $s_i(t,d_{ih}(t))$ into SNN
7.    Calculate hidden spike trains $s_h(t,d_{ho}(t))$ and actual output spike trains $s_o(t)$
      according to neuronal internal state
8.    Calculate continuous functions $f_{s_h}(t,d_{ho}(t))$ and $f_{s_o}(t)$ using Eq. (2)
9.    Calculate network error $E$ using Eq. (4)
10.   Calculate $\Delta w_{ho}$, $\Delta w_{ih}$, $\Delta d_{ho}$ and $\Delta d_{ih}$ using Eqs. (11), (16), (21) and (25), respectively
11.   Update all synaptic weights and delays
12. until network error $E$=0 or the maximum learning epoch is reached
```

```
(b) FFOnDD: Online Supervised Learning Algorithm
Input: Untrained SNN
Output: Trained SNN
1.  Set up a multilayer feedforward SNN
2.  Initialize all synaptic weights and delays
3.  Initialize input spike trains $s_i(t,d_{ih}(t))$ and desired output spike trains $\hat{s}_o(t)$
4.  Calculate continuous functions $f_{s_i}(t,d_{ih}(t))$ and $f_{\hat{s}_o}(t)$ using Eq. (2)
5.  repeat
6.    for $t \leftarrow 1$ to $T$ do
7.      Input all $s_i(t,d_{ih}(t))$ into SNN
8.      Calculate hidden spike trains $s_h(t,d_{ho}(t))$ and actual output spike trains $s_o(t)$
        according to neuronal internal state
9.      Calculate continuous functions $f_{s_h}(t,d_{ho}(t))$ and $f_{s_o}(t)$ using Eq. (2)
10.     Calculate network error $E$ using Eq. (4)
11.     if $t == t_o^m \vee t == \hat{t}_o^n$ then
12.       Calculate $\Delta w_{ho}(t)$, $\Delta w_{ih}(t)$, $\Delta d_{ho}(t)$ and $\Delta d_{ih}(t)$ using Eqs. (10), (15), (20)
          and (24), respectively
13.       Update all synaptic weights and delays
14.     end if
15.   end for
16. until network error $E$=0 or the maximum learning epoch is reached
```

Fig. 1 – Pseudocode of offline and online supervised learning for feedforward SNNs using the proposed synaptic weight and delay update rules.

3. RESULTS

This section validates the performance of the proposed algorithm outlined in Section 2.4 through spike train learning tasks and nonlinear pattern recognition tasks. To highlight the advantages of delay learning, a comparative analysis is conducted between weight-delay co-learning (including the online learning algorithm FFOnDD and the offline learning algorithm FFOffDD) and weight-only learning (including the online learning algorithm FFOnSD and the offline learning algorithm FFOffSD).

3.1. Spike train learning

In this subsection, the learning performance of the proposed algorithm is evaluated through spike train learning tasks. This is a training process, not a testing process, so there is no overfitting issue. A feedforward SNN consisting of 200 input neurons, 60 hidden neurons and 1 output neuron is constructed using spiking response models. Within the neuron model, the time constants for the spike response function and the refractory period function are set to 2 ms and 30 ms, respectively. The spike firing threshold is set to 1, while the length of the absolute refractory period is 1 ms. The synaptic weights and delays are randomly initialized within the ranges of [0, 1] and [0, 10] ms, respectively. The learning rates η_w and η_d of synaptic weights and delays are one of the important factors affecting the performance of the supervised learning algorithm of SNNs. After several tests, it is found that the learning performance of the proposed learning algorithm is optimal when η_w and η_d are set to 0.01 and 5, respectively. Therefore, the values of the learning rate are set to $\eta_w = 0.001$ and $\eta_d = 5$. Both the input and desired output spike trains are generated randomly within a simulation time interval of $T = 100$ ms using the homogeneous Poisson encoding approach with a frequency of 20 Hz. Each learning task is repeated 20 times, with a maximum learning epoch of 100 set for each trial. The evaluation of learning performance is based on two key metrics: one is to measure the learning accuracy through the network error function defined in Eq. (4); and the second is to evaluate the convergence speed and program running time of the algorithm by the learning epoch required to minimize the network error.

Firstly, the effect of network size on the learning performance of the proposed algorithm is analyzed, including the number of neurons in the input and hidden layers. The corresponding results are presented in Fig. 2. Figures 2a and 2b illustrate the network error and learning epoch as the number of input neurons varies, while Figs. 2c and 2d show the network error and learning epoch when the number of hidden neurons varies.

It can be seen that the network errors of the online learning algorithms FFOnDD and FFOnSD are consistently lower than those of the offline learning algorithms FFOffDD and FFOffSD. This is because online learning algorithms update the network parameters multiple times within a learning epoch, while offline learning algorithms only update the network parameters once within a learning epoch. As a result, online learning algorithms achieve smaller network errors than offline learning algorithms. Moreover, the network errors and required learning epochs of the weight-delay co-learning algorithms FFOnDD and FFOffDD are also smaller than those of their corresponding weight-only learning algorithms FFOnSD and FFOffSD. This aligns with our theoretical claim that synaptic delay learning can significantly improve the learning performance of SNNs. The results indicate that the weight-delay co-learning algorithms achieves lower network error in fewer learning epochs under different network sizes, thereby providing robust evidence for the efficacy of synaptic delay learning.

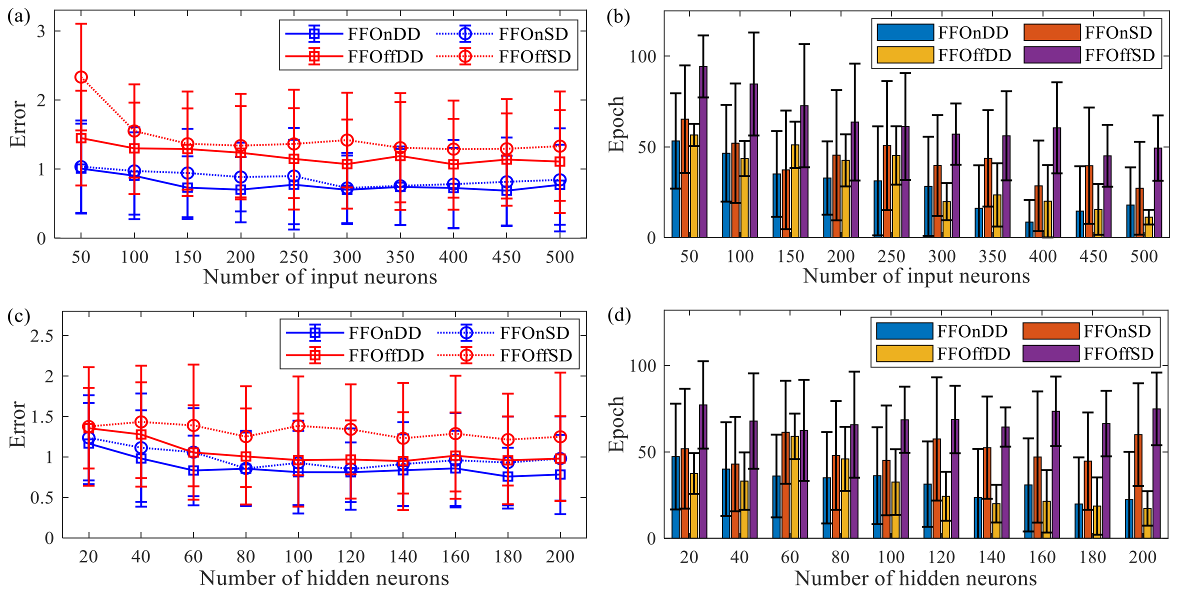

Fig. 2 – Spike train learning results with different network sizes.

Next, the effect of the firing rate and length of spike trains on the learning performance of the proposed algorithm is analyzed, and the corresponding results are shown in Fig. 3. Figures 3a and 3b show the network error and learning epoch as the firing rate of input spikes varies. It is observed that the network error for all four algorithms remains relatively stable with increasing the firing rate of input spikes, while the required learning epoch exhibits a gradual decrease. Furthermore, the network error and learning epochs for FFOnDD and FFOffDD algorithms are consistently lower than those of FFOnSD and FFOffSD algorithms. This indicates that the weight-delay co-learning algorithms demonstrate superior learning efficiency and accuracy across different firing rates of input spikes. Figures 3c and 3d show the network error and learning epoch when the length of spike trains is different. It can be seen that as the length of spike trains gradually increases, the network error of the four algorithms gradually increases, while the required learning epochs show varying degrees of fluctuation. Notably, the network error and learning epochs for the weight-delay co-learning algorithms FFOnDD and FFOffDD remain lower than those of the corresponding weight-only algorithms FFOnSD and FFOffSD. These results demonstrate that the weight-delay co-learning algorithms outperform their corresponding weight-only learning algorithms in terms of both network error and learning epochs across different spike train parameters, thereby further validating the effectiveness of delay learning in enhancing the learning performance of SNNs.

Overall, synaptic delay plasticity enables spiking neurons to integrate input information from different presynaptic neurons more effectively. Therefore, weight-delay co-learning SNNs exhibit better learning performance than weight-only learning SNNs. Additionally, online learning algorithms can calculate network errors in real time based on the output of SNNs during the learning process and update the synaptic weights and delays of SNNs, which not only improves the learning accuracy of the algorithm but also accelerates the convergence speed of the SNN model.

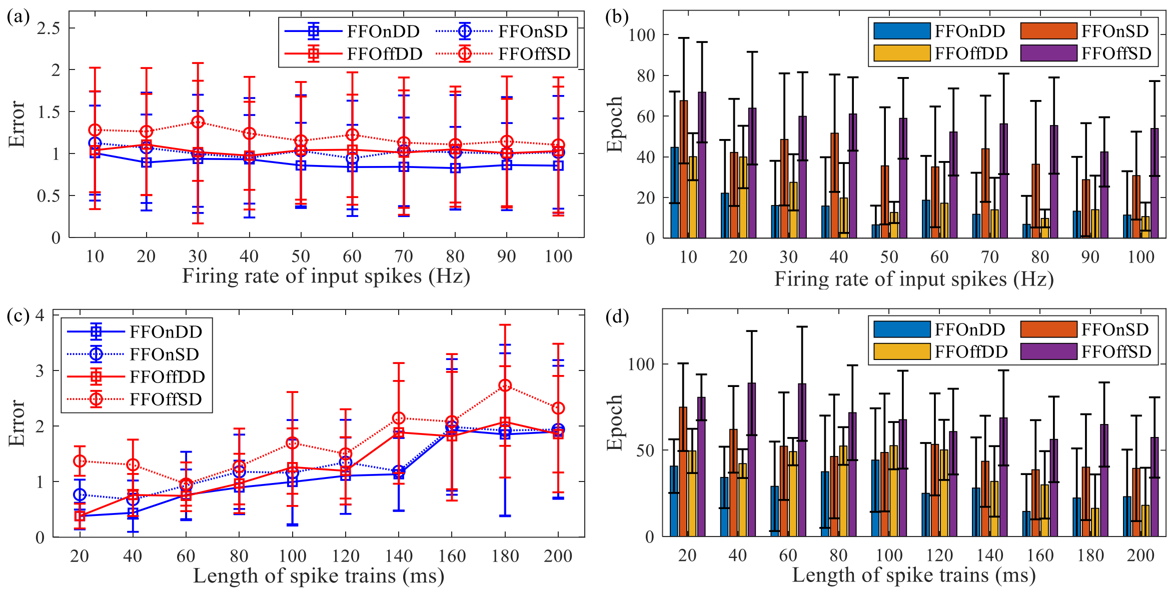

Fig. 3 – Spike train learning results with different spike train parameters.

3.2 Nonlinear pattern classification

In this subsection, the original Wisconsin Breast Cancer Database (WBC) and the Pima Indians Diabetes Database (PIMA) are utilized to evaluate the performance of the proposed algorithm in the domain of nonlinear pattern classification. Each dataset is randomly partitioned into a training set comprising 50% of the samples and a testing set containing the remaining 50%. Figure 4 illustrates the process of employing SNNs to solve nonlinear pattern classification problems, which encompasses three primary phases: 1) *Spike train encoding*. The linear encoding approach is employed to convert each attribute value into a corresponding spike train. Initially, the sample attribute values are normalized to the range of [0, 1], followed by converting them to frequency values within the range of [5,20] Hz. Subsequently, these frequency values are encoded into spike trains within a duration of [0, 50] ms; 2) *Training phase*. The spike trains corresponding to the samples in the training set are input into the SNN for supervised learning. The frequency values 5Hz and10Hz are used to generate desired output spike trains corresponding the label of samples within the dataset. These parameters are determined after several repeated tests. The classification accuracy on the training set is then computed; 3) *Testing phase*. The spike trains corresponding to the samples in the testing set are fed into the trained SNN. Classification is performed based on the similarity between the actual and desired output spike trains, and the classification accuracy on the testing set is subsequently calculated. In the nonlinear pattern classification task, the maximum number of training epochs is set to 100, and 20 repeated tests are conducted for each dataset.

```
Input: Untrained SNN and dataset
Output: Trained SNN and classification accuracy
1.  Set up a multilayer feedforward SNN
2.  Initialize all synaptic weights and delays
3.  Divided the dataset into training set and testing set
4.  repeat    //Training Phase
5.      for all samples in the training set do
6.          Encode each sample into spike trains using the linear encoding method
7.          Train the SNN in supervised learning framework
8.          Calculate the training accuracy
9.      end for
10. until the maximum learning epoch is reached
11. for all samples in the testing set do    //Testing Phase
12.     Encode each sample into spike trains using the linear encoding method
13.     Input spike trains into the trained SNN
14.     Calculate the testing accuracy
15. end for
```

Fig. 4 – Pseudocode of SNN-based nonlinear pattern classification.

Table 1 compares the classification accuracy achieved by the proposed algorithm with several supervised learning algorithms for SNNs on the two datasets. These algorithms are all for multilayer feedforward SNNs with the same network structure, and their classification accuracies are from the original paper. The results indicate that the classification accuracy of the delay learning algorithms FFOnDD and FFOffDD surpasses that of their corresponding weight learning algorithms FFOnSD and FFOffSD on both the training and testing sets of the two datasets. This observation suggests that the integration of learnable delays within feedforward SNNs enhances their efficacy in nonlinear pattern classification tasks. Furthermore, the proposed algorithm demonstrates superior classification accuracy on the testing sets of both datasets when compared to other algorithms. These are not overfitting classification results. This finding underscores the effectiveness and competitiveness of the proposed approach in dealing with complex nonlinear pattern classification problems.

Table 1

Comparison of classification performance on the WBC and PIMA datasets

Algorithms	WBC dataset			PIMA dataset		
	Training Accuracy (%)	Testing Accuracy (%)	Epoch	Training Accuracy (%)	Testing Accuracy (%)	Epoch
MultiDL-ReSuMe [8]	98.2	96.4	100	72.1	70.6	100
Delay-FE [9]	97.6±0.6	97.4±0.5	200	**77.4±1.3**	72.2±1.4	200
Multi-STIP [11]	97.2±0.4	97.1±0.7	100	73.1±1.3	70.5±1.4	100
Multi-SLFA [12]	96.9±0.6	96.4±0.8	150	73.2±1.7	71.3±2.3	200
Multi-OSLFA [12]	97.2±0.7	96.9±0.6	150	72.7±1.1	72.2±1.5	200
Multi-ReSuMe [13]	95.4±1.0	94.8±1.4	100	69.1±3.0	67.3±2.9	100
MPD-AL [14]	**99.9±0.1**	97.2±0.6	200	71.4±1.9	69.6±1.3	200
OnMultiSpikeProp [15]	97.5±0.9	97.1±0.6	100	71.5±1.3	72.2±1.2	100
FFOnSD (Ours)	96.1±0.9	95.1±0.6	100	71.4±1.6	71.0±1.1	100
FFOnDD (Ours)	98.2±0.6	**97.7±0.4**	100	73.6±1.8	**72.9±1.0**	100
FFOffSD (Ours)	95.9±1.0	94.9±1.2	100	71.0±1.2	70.9±1.4	100
FFOffDD (Ours)	98.1±0.5	**97.7±0.5**	100	73.4±1.4	72.3±1.1	100

4. DISCUSSION AND CONCLUSION

This paper proposes a spike train kernel-based supervised delay learning algorithm for multilayer feedforward SNNs to address the synergistic plasticity of synaptic connection strengths and synaptic delays. The advantages and disadvantages of the proposed approach are discussed from the following three aspects: 1) *Comparison with other supervised learning algorithms for feedforward* SNNs. The proposed delay learning algorithm is derived using the kernel function representation of spike trains. In contrast, the delay learning algorithm presented in [8] is constructed based on the STDP rule, while the algorithm in [9] is derived using a discrete spike time based gradient descent rule. Additionally, the spike train kernel-based algorithms proposed in [11] and [12] focus on weight learning but do not address delay learning; 2) *Advantages*. Firstly, the proposed algorithm integrates synaptic delay plasticity and online learning mechanism, providing a level of biological interpretability. Furthermore, the derivation of the algorithm does not depend on the analytical expressions of specific spiking neuron models, enhancing its adaptability across various neuron models. In addition, the test results demonstrate that introducing learnable synaptic delays enables SNNs to achieve higher learning accuracy with fewer learning epochs, thereby significantly improving their overall learning performance; 3) *Disadvantages*. Firstly, due to the relatively simple network structure employed in this study, the performance of the algorithm has not been verified on large-scale complex datasets. Consequently, future research will extend to more diverse real-world datasets to validate the generalizability and robustness of the proposed algorithm. Additionally, due to the difficulty involved in analyzing the computational complexity of SNN supervised learning algorithms and the scarcity of relevant literatures, this paper does not provide a comprehensive analysis of the computational complexity of the proposed algorithm. Nonetheless, it has been shown in [4] that spike train kernel-based algorithms require less program running time compared to those derived from gradient descent rule using discrete spike times. Future investigations will aim to theoretically analyze the computational complexity and convergence properties of the proposed algorithm.

In summary, this paper presents a novel spike train kernel-based delay learning algorithm for multilayer feedforward SNNs. The proposed algorithm enables the simultaneous optimization of weights and delays during the training of SNNs, thereby enhancing their representational capacity and learning efficiency. The results of spike train leaning and nonlinear pattern classification tasks demonstrate that the synaptic weight-delay co-learning algorithm significantly outperforms the weight-only learning algorithm, achieving not only higher learning accuracy but also a substantial reduction in the number of learning epochs required for network convergence. Future research will concentrate on investigating the mechanisms underlying synaptic weight-delay co-learning in deep SNNs with more hidden layers, as well as exploring the application of complex spatiotemporal pattern learning on large-scale real-world datasets, thereby advancing the application and development of brain-like intelligent computational models.

ACKNOWLEDGEMENTS

This study was supported by the Natural Science Foundation of Gansu Province (24JRRA127), the Youth Science and Technology Foundation of Gansu Province (23JRRA1577), and the Scientific Research Ability Enhancement Program for Young Teachers of Northwest Normal University (NWNU-LKQN2024-21).

REFERENCES

[1] Eshraghian JK, Ward M, Neftci EO, Wang X, Lenz G, Dwivedi G, Bennamoun M, Jeong DS, Lu WD. Training spiking neural networks using lessons from deep learning. Proceedings of the IEEE, 2023, 111(9): 1016−1054.

[2] Păun G, Pérez-Jiménez MJ, Rozenberg G. Infinite spike trains in spiking neural P systems. Romanian Journal of Information Science and Technology, 2023, 26 (3-4): 251–275.

[3] Wu J, Wang Y, Li Z, Lu L, Li Q. A review of computing with spiking neural networks. Computers, Materials & Continua. 2024; 78(3): 2909–2939.

[4] Wang X, Lin X, Dang X. Supervised learning in spiking neural networks: a review of algorithms and evaluations. Neural Networks. 2020; 125: 258–280.

[5] Sun P, De Winne J, Zhang M, Devos P, Botteldooren D. Delayed knowledge transfer: Cross-modal knowledge transfer from delayed stimulus to EEG for continuous attention detection based on spike-represented EEG signals. Neural Networks. 2025; 183: 107003.

[6] Zhang M, Wu J, Belatreche A, Pan Z, Xie X, Chua Y, Li G, Qu H, Li H. Supervised learning in spiking neural networks with synaptic delay-weight plasticity. Neurocomputing. 2020; 409: 103–118.

[7] Zou L, Wang S, Zhang H, Liang T, Zhang M, Wang X. Delay learning for spiking neurons with multiple synaptic connections. Proceedings of the Romanian Academy; Series A – Mathematics, Physics, Technical Sciences, Information Science. 2025; 26(1): 89–99.

[8] Taherkhani A, Belatreche A, Li Y, Maguire LP. A supervised learning algorithm for learning precise timing of multiple spikes in multilayer spiking neural networks. IEEE Transactions on Neural Networks and Learning Systems. 2018; 29(11): 5394–5407.

[9] Luo X, Qu H, Wang Y, Yi Z, Zhang J, Zhang M. Supervised learning in multilayer spiking neural networks with spike temporal error backpropagation. IEEE Transactions on Neural Networks and Learning Systems. 2023; 34(12): 10141−10153.

[10] Park IM, Seth S, Paiva AR, Li L, Principe JC. Kernel methods on spike train space for neuroscience: a tutorial. IEEE Signal Processing Magazine. 2013; 30(4): 149–160.

[11] Lin X, Wang X, Hao Z. Supervised learning in multilayer spiking neural networks with inner products of spike trains. Neurocomputing. 2017; 237: 59−70.

[12] Lin X, Hu J, Zheng D, Hu T, Wang X. An online supervised learning algorithm based on feedback alignment for multilayer spiking neural networks. Proceedings of the Romanian Academy; Series A – Mathematics, Physics, Technical Sciences, Information Science. 2022; 23(2): 189−198.

[13] Sporea I, Grüning A. Supervised learning in multilayer spiking neural networks. Neural Computation. 2013; 25(2): 473–509.

[14] Zhang M, Wu J, Chua Y, Luo X, Pan Z, Liu D, Li H. MPD-AL: an efficient membrane potential driven aggregate-label learning algorithm for spiking neurons. In: AAAI Conference on Artificial Intelligence, vol. 33. 2019, pp. 1327−1334.

[15] Lin X, Hu T, Wang X. One-pass online learning based on gradient descent for multilayer spiking neural networks. IEEE Transactions on Cognitive and Developmental Systems. 2023; 15(1): 16−31.

Received April 20, 2025