A Supervised Learning Algorithm for Spiking Neurons Using Spike Train Kernel Based on a Unit of Pair-Spike

GUOJUN CHEN AND GUOEN WANG
School of Urban Design, Wuhan University, Wuhan 430072, China
Corresponding author: Guoen Wang (150251932@qq.com)
This work was supported by the National Natural Science Foundation of China under Grant 51878516.

ABSTRACT In recent years, neuroscientists have discovered that the neural information is encoded by spike trains with precise times. Supervised learning algorithm based on the precise times for spiking neurons becomes an important research field. Although many existing algorithms have the excellent learning ability, most of their mechanisms still have some complex computations and certain limitations. Moreover, the discontinuity of spiking process also makes it very difficult to build an efficient algorithm. This paper proposes a supervised learning algorithm for spiking neurons using the kernel function of spike trains based on a unit of pair-spike. Firstly, we comprehensively divide the intervals of spike trains. Then, we construct an optimal selection and computation method of spikes based on the unit of pair-spike. This method avoids some wrong computations and reduces the computational cost by using each effective input spike only once in every epoch. Finally, we use the kernel function defined by an inner product operator to solve the computing problem of discontinue spike process and multiple output spikes. The proposed algorithm is successfully applied to many learning tasks of spike trains, where the effect of our optimal selection and computation method is verified and the influence of learning factors such as learning kernel, learning rate, and learning epoch is analyzed. Moreover, compared with other algorithms, all experimental results show that our proposed algorithm has the higher learning accuracy and good learning efficiency.

INDEX TERMS Direct computation, spike selection, spike train kernel, spiking neural networks, spiking neurons, supervised learning, pair-spike.

I. INTRODUCTION
Neurocomputing science explores the mathematical basis of neurocomputing by studying the mechanism and process of neural intelligent system [1], [2]. As the research object of neurocomputing science, artificial neural networks (ANNs) abstract and simulate the structure and function of biological nervous system, and play an important role in information processing and pattern recognition [3]–[5]. An artificial neural network consists of many artificial neuron models [6]. Every neuron model represents an activation function, and every connection between arbitrary two neuron models represents a synaptic weight. Every neuron in biological nervous system can transmit the information to other neurons by emitting spike trains [7]. Information in traditional neuron models is encoded and processed based on the firing rate of spike trains, because there is a large amount of information in the rate [8]. Related encoding methods include spike count, spike density, and population activity [2]. Related activation functions include Sigmoid function, Rectified Linear Unit (ReLU), and piecewise linear function [3]. However, the firing rate is too simple to describe neuronal activities, and neuroscientists suggested that the precisely firing time of spikes is also essential for informational encoding [9], [10]. Then, many researchers proposed a variety of precisely temporal encoding methods such as time to first spike, latency phase, and rank order [2]. Based on these encoding methods, spiking neuron models can simulate biological neurons in the temporal dimension and have more biological plausibility than traditional neuron models [11], [12]. Composed of spiking neurons, spiking neural networks (SNNs) as a new generation of neural networks provide the stronger ability...
to approximate arbitrary continuous functions [12]–[14], and can process many complex spatio-temporal patterns well [15]–[19].

Supervised learning in ANNs has a learning mechanism which furnishes the desired output data with given input data, produces the actual output data, and makes a comparison of two output data sets to regulate the synaptic weights [2]. When the sample has changed, supervised learning can make the network adapt to a new circumstance by regulating synaptic weights. These data sets used in the training are the sets of discrete spikes, which are called spike train. With precisely temporal encoding, the aim of supervised learning for spiking neurons or SNNs is to output corresponding desired spike trains [20]. Researchers verified that the mechanism of supervised learning naturally exists in the biological nervous system, even though several details are still unknown [2]. The referent activity template is one of the current explanations for supervised learning, which is mimicked by the circuits of interest [21]. Moreover, the desired output signal of supervised learning can represent an instantiation of reinforcement learning by operating only on a much shorter duration [22]–[24].

Supervised learning algorithms for spiking neurons or SNNs mainly include three types, gradient descent, synaptic plasticity, and spike train convolution [25]. Algorithms based on gradient descent firstly use the gradient value and back-propagation (BP) to gain a degree of association between spike trains. This degree is applied to adjust the synaptic weight so that the deviation between the actual output spike train and desired output spike train tends to zero. SpikeProp [26] as the first algorithm of gradient descent for SNNs can learn with a single spike. Then, Xu et al. [27] proposed gradient-descent-based (GDB) which can learn multiple spikes in every layer. Henceforth, the algorithm based on gradient descent for SNNs can process the complex spatial-temporal pattern of spike trains. Algorithms based on the synaptic plasticity are more biologically plausible. Based on the mechanism of synaptic plasticity proposed by Knott [28], spike-timing-dependent plasticity (STDP) mechanism is useful for spike train learning [29], [30]. Combining the Bienenstock-Cooper-Munro (BCM) rule and STDP, Wade et al. constructed an algorithm named synaptic weight association training (SWAT) [31]. Ponulak and Kasinski [32] proposed a remote supervised method (ReSuMe), which adjusts all synaptic weights according to STDP and anti-STDP. Because the learning mechanism of ReSuMe only depends on the spike trains and STDP, ReSuMe can adapt many types of spiking neuron models [33]. Based on ReSuMe, Lin et al. [34] proposed T-ReSuMe by combining triple spikes. Taherkhani et al. [35] constructed DL-ReSuMe by merging a delay shift approach and ReSuMe. The above two types are only based on the discontinuous points [25]. However, algorithms based on spike train convolution can transform all spike trains into a continuous form by using specific spike train convolution kernel. Hence, spike trains can be calculated by using common mathematical operators [36].

Carnell and Richardson [37] mapped all spike trains into a vector space and made use of linear algebra to construct a supervised algorithm. Mohammed et al. [38] came up with spike pattern association neuron (SPAN) by combining Hebbian interpretation of the convolution and Widrow-Hoff (WH) rule. Based on SPAN algorithm, Yu et al. [39] proposed a precise-spike-driven (PSD) rule, which associates all input spike trains with the given desired spike train and only uses one convolution of input spike trains. Then, Lin et al. [40] proposed a spike train kernel learning rule (STKLR) based on the definition kernel functions for inner product of spike trains. Gardner and Grüning [41] proposed an optimal algorithm Filtered-error synaptic plasticity rule (FILT), where a convolution with an exponential filter is considered. Recently, Wang et al. [42] constructed a variable delay learning algorithm based on the inner product of spike trains.

We focus on the supervised learning algorithm of spike trains for spiking neurons in this paper. Firstly, multiple spikes from input, actual output, and desired output spike trains make up a complete data set of spikes to regulate the synaptic weights, so spike selection in this set is an important research part. In addition, the discontinuous internal state variables in neuron models also bring many difficulties into the construction of an efficient algorithm. In this paper, we analyze some relationships between all spikes in the above set, and discover that existing algorithms have some redundancies in the spike selection and computation. To eliminate the repetitive spike computation, we construct a direct regulation method of synaptic weights based on a unit of pair-spike, which can be applied to all supervised learning algorithms of spike trains. Then, using the representation of spike train kernel, we transform all discrete spike trains into the continuous function. Thus, the unified representation of spike trains and formal definition of spike trains similarity measure are realized by mapping all spike trains into a high-dimension space corresponding to the kernel function. Finally, we use the spike train kernel based on a unit of pair-spike to construct a supervised learning algorithm for spiking neurons. We denote the algorithm as unit of pair-spike (UPS).

The rest of this paper is organized as follows. In section II, the spiking neuron model and learning architecture are introduced. Section III presents the divide of intervals for spike trains, a spike selection and computation method called pair-spike, and UPS. In section IV, many experiments are carried out to demonstrate the learning capability of UPS and compare UPS with other algorithms. Section V presents the discussion and section VI describes the conclusion and future work.

II. SPIKING NEURON MODEL AND LEARNING ARCHITECTURE

A. SPIKING NEURON MODEL

Spiking neuron model has many types such as spike response model (SRM), integrate-and-fire model (IF), leaky integrate-and-fire model (LIF), and Hodgkin–Huxley model (HH) [2].
They are all constructed to explain the spiking mechanism of biological neurons. SRM is applied more and more widely, because it can intuitively express the internal state of neurons and easily describe the learning algorithms [27]. Furthermore, SRM has more biological plasticity than other spiking neuron models such as LIF and integrate-and-fire (IF) [43]. Thus, we select SRM as the basic spiking neuron model.

Based on the cortical interneuron, SRM proposed by Jolivet et al. [43] is a spiking neuron model with a varying neuronal threshold and like a nonlinear IF. However, SRM uses an integral over the past to express the membrane potential, which is different from the nonlinear IF. Then, this standard SRM was simplified by setting a fixed neuronal threshold to use easily [44]. Furthermore, by only using the firing time of since output spike to express the refractory period, SRM is convenient in many researches [34], [36], [45]. Thus, we use this convenient SRM. In SRM, the internal state as an integral over the past is represented by a variable membrane potential $u$ at time $t$. If there is no input spike, $u$ is at a resting state. When input spikes arrive from presynaptic neurons via synapses, if $u$ defined as a sum of postsynaptic potential (PSP) is below the setting threshold $V_{\text{ thresh}}$, $u$ will be induced by these input spikes. Suppose that some input spiking neurons connect with the current neuron through $N$ neural synapses, $G_i$ spikes are transmitted by $i$th synapse. We denote the arrival time of these spikes at the current neuron as $\{t_1^i, t_2^i, \ldots, t_G^i\}$, the firing time of $i$th spike as $t_i^u$ and the firing time of the most recent output spike prior to current time as $\hat{t}$. The internal state $u(t)$ at $t$ is,

$$u(t) = \rho(t - \hat{t}) + \sum_{i=1}^{G} \sum_{g=1}^{N} w_{i} \varepsilon(t - t_i^g) \quad (1)$$

where $w_i$ is a synaptic weight for $i$th synapse, $i \in \mathbb{Z}^+$. PSP is determined by a spike response function $\varepsilon(t)$ expressed as:

$$\varepsilon(t) = \begin{cases} \frac{t}{\tau} e^{-\frac{t}{\tau}} & \text{if } t \geq 0 \\ 0 & \text{if } t < 0 \end{cases} \quad (2)$$

where $\tau$ is a time decay constant that determines the shape of this function. After an accumulation of input spikes, SRM fires an output spike at time $t^f$ when $u(t)$ exceeds $V_{\text{ thresh}}$. The membrane potential $u(t)$ immediately falls back to a resting potential $u_{\text{ rest}} = 0$ at $\hat{t}$. The current neuron must not fire a spike even receiving many input spikes. Then, there are two processes, absolute refractory period (ARP) and relative refractory period (RRP). The phase named ARP is a repolarization. At the end of ARP, current neuron comes into RRP, whose $u(t)$ is lower than $u_{\text{ rest}}$ and cannot reach $V_{\text{ thresh}}$. Therefore, firing spike is difficult for the current neuron. RRP is a hyper polarization. In the expression of the neuronal internal state, a refractoriness function $\rho(t - \hat{t})$ depicts this RRP. The function $\rho(t - \hat{t})$ is given by,

$$\rho(t - \hat{t}) = \begin{cases} -2V_{\text{ thresh}} e^{-\frac{t}{\tau}} & \text{if } t \geq 0 \\ 0 & \text{if } t < 0 \end{cases} \quad (3)$$

where $\tau_r$ is a time decay. The refractoriness is mainly induced by $\hat{t}$, which ignores all earlier output spikes.

### B. LEARNING ARCHITECTURE

Fig. 1 presents the learning architecture, which makes up of two components, input layer and output layer. The input layer includes many input units and output layer includes one output unit. Input units are independent of each other. Every input unit connects with the output unit. The synaptic weight represents the connection strength. $w_1$, $w_2$, and $w_i$ are 1st, 2nd, and $i$th weights between input units and the output unit respectively, where $i \in \mathbb{Z}^+$. All input units generate the input train of discrete spikes. The output unit generates an actual output spike train, where the desired output spike train is set in advance. The supervised learning is implemented by adjusting all synaptic weights. Descriptions of the learning architecture are as follows: The output unit generates a desired output spike train when every input unit generates only an input spike train. Then, all input spike trains generated from the input layer transfer to the output layer. After a transformation through the synapses, all input spikes accumulate in the output unit to output an actual output spike train. When the actual output spike train is outputted, it is applied to compare with the desired output spike train. If they are different, the error of them will be back to control the regulation of all synaptic weights. Because there is usually still a new difference after the regulation in an epoch, the process has to be iterative until the two output spike trains are absolutely equal or the number of iterations is up to a setting value. Many experiments have proved the learning capability of this architecture [31]–[40].

**FIGURE 1.** The learning architecture used in this paper.

### III. THE SUPERVISED LEARNING ALGORITHM: UPS

#### A. THE DIVIDE OF INTERVALS FOR SPIKE TRAINS

The spike train $s = \{t^f \in \Gamma : f = 1, 2, \ldots, F\}$ represents a sequence of spiking times in an interval $[0, T]$. It is shown formally as:

$$s(t) = \sum_{f=1}^{F} \delta(t - t^f) \quad (4)$$

where $F$ is the number of spikes and $t^f$ is the $f$th spike firing time. $\delta(t)$ is the Dirac delta function. If $t = 1$, $\delta(t) = 1$,
otherwise $\delta(t) = 0$. $s_i$, $s_d$ and $s_g$ express $i$th input spike train, the actual output spike train, and the desired output spike train respectively. $t_d^g$ and $t_d^h$ denote the firing times of $g$th spike in $s_d$ and $h$th spike in $s_g$ respectively, where $g \in \mathbb{Z}^+$ and $h \in \mathbb{Z}^+$. $G$ is the number of all spikes in $s_d$ and $H$ is the number of all spikes in $s_g$, where $G \in \mathbb{Z}^+$ and $H \in \mathbb{Z}^+$.

We denote every time interval of two nearest desired output spikes as desired time interval (DTI). The divide of intervals for $s_d$ is shown in Fig.2. Except the first DTI is $[0, t_d^1]$, the $g$th interval is $[t_d^{g-1}, t_d^g]$. We denote every time interval of two nearest spikes in $s_d$ as actual time interval (ATI). The first ATI is $[0, t_a^1]$ and the $h$th interval is $[t_a^{h-1}, t_a^h]$.

**FIGURE 2.** The divide of time intervals for a desired output spike train. A time interval between two nearest desired output spikes is a desired time interval (DTI), where the first DTI starts at 0 ms.

By combining $s_a$ with $s_d$, we divide $s_i$ into some intervals that include the input spikes. Because DTI is fixed in the learning process, we select DTI as the basic interval. There are three types of intervals in a DTI. The first type named first input time interval (FITI) is from the firing time of the first desired output spike to the first actual output spike, the second type named median input time interval (MITI) is between two nearest actual output spikes, and the third type named last input time interval (LITI) is from the last actual output spike to the second desired output spike. There is no MITI when only one actual output spike exists in a DTI. Especially, if there is no actual output spike in a DTI, the DTI only equals to a LITI. Some intervals in Fig.3 include this condition.

In Fig.3, 0 millisecond (ms) as the initial time means the firing times of actual output spikes and desired output spikes in interval $[0, 0]$ or $[\infty, 0]$ are equal. Set 0 ms as the firing time of the last desired output spike in $[0, 0]$. The interval from 0 ms as the firing time of a desired output spike to $t_d^1$ is a DTI $[0, t_d^1]$. The interval from 0 ms as the firing time of an actual output spike to $t_a^1$ is a LITI $[0, t_a^1]$. Thus, a DTI without actual output spike equals to a LITI.

**FIGURE 3.** The divide of intervals for $s_i$. $[t_d^1, t_d^2]$ is a FITI, $[t_d^2, t_d^3]$ and $[t_a^2, t_a^3]$ belong to MITI, $[0, t_a^2]$ and $[t_a^2, t_a^3]$ belong to LITI.

### B. THE SELECTION AND COMPUTATION OF PAIR-SPIKE

For the supervised learning of spiking neuron models, the adjustment of all synaptic weights depends on the relative temporal relationship among input spike trains, actual output spike trains, and desired output spike trains [25], [40]. With WH rule, the changed value of $w_i$ at $t$ time is expressed as:

$$\Delta w_i(t) = \eta s_i(t) [s_d(t) - s_a(t)] \quad (5)$$

where $\eta$ is the learning rate (LR), $s_i(t)$ is an input train, $s_d(t)$ is an actual output train, and $s_a(t)$ is a desired output train. Generally, $w_i$ increases when the running time equals to the firing time of desired output spike at $t_d^g$ and decreases when the time equals to the firing time of the actual output spike at $t_a^h$. We set $t_i^g$ as the firing time of $f$th spike and $f_i$ as the number of spikes in $s_i$, then the update rule of $w_i$ for every $t_i^g$ is given as follows:

$$\Delta w_i(t_i^g) = \begin{cases} \sum_g \phi(t_d^g - t_i^g), & t_i^g < t_d^g \\ - \sum_h \phi(t_a^h - t_i^g), & t_i^g < t_a^h \end{cases} \quad (6)$$

where $\phi(\cdot)$ is the update function of $w_i$. We can know from (6) that every $t_i^g$ earlier than $t_d^g$ is selected to compute with $t_d^g$ to increase $w_i$. If $t_i^g$ is earlier than $t_d^g$, $t_i^g$ will be selected $G$ times. Similarly, $t_i^g$ earlier than $t_a^h$ is selected to compute with $t_a^h$ to decrease $w_i$. If $t_i^g$ is earlier than $t_a^h$, $t_i^g$ will be selected $H$ times. Except the use of the same $s_i$, two parts of (6) have no direct connections with each other. That is to say, if $t_i^g$ is not only earlier than $t_d^g$ but also earlier than $t_a^h$, $t_i^g$ will firstly compute with $t_d^g$ to increase $w_i$ and then compute with $t_a^h$ to decrease $w_i$. This spike selection and computation method is widely used in existing algorithms [25], [32], [34] and called multi-spike in this paper. The total adjustment of $w_i$ for $t_i^g$ equals to the sum of two parts in (6), so the corresponding update value is expressed as follows:

$$\Delta w_i(t_i^g) = \sum_{t_i^g < t_d^g} \phi(t_d^g - t_i^g) - \sum_{t_i^g < t_a^h} \phi(t_a^h - t_i^g) \quad (7)$$

Fig.4 shows an example of multi-spike. However, there is a problem of wrong adjustment. In Fig.4, $t_i^1$, $t_i^2$ in FITI are
selected twice, one is computed with \( t_d^1 \) to decrease \( w_i \), and the other is computed with \( t_d^3 \) to increase \( w_i \). \( t_d^1 \), \( t_d^3 \), and \( t_d^4 \) in LITI are just computed once with \( t_d^1 \) to increase \( w_i \). The final update value of \( w_i \) in the first learning epoch is obtained as:

\[
\Delta w_i = \Delta w_i(t_d^1) + \Delta w_i(t_d^3) + \Delta w_i(t_d^4)
\]

\[
= \phi(t_d^1 - t_d^1) - \phi(t_d^1 - t_d^1) + \phi(t_d^3 - t_d^3) - \phi(t_d^3 - t_d^3)
+ \phi(t_d^4 - t_d^2) + \phi(t_d^4 - t_d^4)
= \phi(t_d^1 - t_d^1) + \phi(t_d^3 - t_d^3) + \phi(t_d^4 - t_d^4)
- \left( \phi(t_d^1 - t_d^1) + \phi(t_d^3 - t_d^3) \right)
\]

(8)

We can see from Fig. 4 that the accumulation of input spikes fired at \( t_d^1 \) and \( t_d^2 \) is enough to produce an output spike at \( t_d^3 \). After that, only the accumulation of input spikes fired at \( t_d^1 \), \( t_d^3 \) is insufficient to produce an output spike at \( t_d^4 \). Support the accumulation of input spikes fired at \( t_d^1 \) and \( t_d^2 \) fails to produce an output spike at \( t_d^3 \), then the accumulation of input spikes fired at \( t_d^1 \), \( t_d^2 \), \( t_d^4 \), and \( t_d^3 \) may produce an output spike at \( t_d^4 \). That is to say, the total increase \( \phi(t_d^1 - t_d^1) + \phi(t_d^3 - t_d^3) + \phi(t_d^4 - t_d^4) \) may be greater than \( \phi(t_d^1 - t_d^1) + \phi(t_d^3 - t_d^3) \) in (8). Thus, \( w_i \) may finally increase, which is opposite to the desired adjustment.

Based on the same principle, there is also the similar problem when we exchange \( t_d^2 \) and \( t_d^3 \). Then, \( t_d^1 \) and \( t_d^2 \) are selected twice, one is computed with \( t_d^1 \) to increase \( w_i \), and the other is computed with \( t_d^4 \) to decrease \( w_i \). \( t_d^3 \) and \( t_d^4 \) are just computed once with \( t_d^1 \) to decrease \( w_i \). The final update value of \( w_i \) in the first learning epoch is obtained as:

\[
\Delta w_i = \Delta w_i(t_d^1) + \Delta w_i(t_d^2) + \Delta w_i(t_d^3) + \Delta w_i(t_d^4)
\]

\[
= \phi(t_d^1 - t_d^1) - \phi(t_d^1 - t_d^1) + \phi(t_d^2 - t_d^2) - \phi(t_d^2 - t_d^2)
- \phi(t_d^3 - t_d^3) - \phi(t_d^4 - t_d^4)
= \phi(t_d^1 - t_d^1) + \phi(t_d^3 - t_d^3) + \phi(t_d^4 - t_d^4)
+ \left( \phi(t_d^1 - t_d^1) + \phi(t_d^3 - t_d^3) \right)
\]

(9)

Only the accumulation of input spikes fired at \( t_d^1 \) and \( t_d^2 \) is insufficient to produce an output spike at \( t_d^1 \) whereas the accumulation of input spikes fired at \( t_d^1 \), \( t_d^2 \), \( t_d^3 \), and \( t_d^4 \) is enough to produce an output spike at \( t_d^4 \). Thus, the total decrease \( \phi(t_d^1 - t_d^1) + \phi(t_d^3 - t_d^3) + \phi(t_d^4 - t_d^4) \) is greater than \( \phi(t_d^1 - t_d^1) + \phi(t_d^3 - t_d^3) \) in (9). The \( w_i \) in evidence should decrease because \( t_d^1 \) is later than \( t_d^1 \). However, \( w_i \) finally decreases, which is opposite to the desired adjustment.

Here, we analyze the causes of above problems. Back to the example shown in Fig. 4, when \( t_d^1 \) is earlier than \( t_d^3 \), the immediate cause of this problem is that \( t_d^1 \) is computed with unrelated \( t_d^1 \) and \( t_d^2 \) in FITI to increase \( w_i \). The primary cause is that the accumulation of only input spikes fired at \( t_d^1 \) and \( t_d^4 \) in LITI is insufficient to produce an output spike at \( t_d^1 \). If there is a MITI, input spikes in MITI will be in the same state with input spikes in FITI. When \( t_d^1 \) is later than \( t_d^1 \), the direct reason is that \( t_d^1 \) is computed with unrelated \( t_d^1 \) and \( t_d^2 \) in FITI to increase \( w_i \). The primary cause is that on the basis of the accumulation of input spikes fired at \( t_d^1 \) and \( t_d^2 \) in FITI, an additional accumulation of only input spikes fired at \( t_d^1 \) and \( t_d^4 \) in LITI leads to produce an output spike at \( t_d^1 \). Therefore, it is a feasible method to remove these unrelated computations.

We set \( t_d^2 \) and \( t_d^4 \) also as the firing time of the first desired output spike and the first actual output spike later than \( t_d^1 \) respectively. According to the above analyses, we design a direct spike selection and computation method based on the input spikes: \( t_d^1, t_d^2 \) and its first output spike later than \( t_d^1 \) (\( t_d^3 \) or \( t_d^4 \)) make up a unit of pair-spike computed only once to adjust \( w_i \) in every epoch. We denote this method as pair-spike. The update value of \( w_i \) for \( t_d^3 \) is changed as follows:

\[
\Delta w_i(t_d^3) = \frac{\phi(t_{min} - t_d^3)(t_d^3 - t_d^4)}{|t_d^3 - t_d^4|}, \quad \phi(t_{min} - t_d^3) \neq 0, \quad t_d^3 = t_d^4
\]

(10)

where \( (t_d^3 - t_d^4)/|t_d^3 - t_d^4| \) determines the increase or decrease. \( t_{min} = \min(t_d^3, t_d^4) \) in (10) is a function to obtain the minimum value between \( t_d^3 \) and \( t_d^4 \), where \( t_d^3 \) and \( t_d^4 \) are initially set to positive infinity or a same number much larger than the known duration \( T \). The final update value of \( w_i \) in one learning epoch is obtained as:

\[
\Delta w_i = \left\{ \begin{array}{ll}
\eta \sum_{f=1}^{F_i} \left[ \phi(t_{min} - t_d^3)(t_d^3 - t_d^4)/|t_d^3 - t_d^4| \right], & t_d^3 \neq t_d^4 \\
0, & t_d^3 = t_d^4
\end{array} \right.
\]

(11)

For a long \( s_d \) with many firing times, \( t_d^3 \) in FITI and MITI is selected by using pair-spike to only compute with \( t_d^4 \) to decrease \( w_i \) and \( t_d^3 \) in LITI is selected by using pair-spike to only compute with \( t_d^4 \) to increase \( w_i \). Like multi-spike, the pair-spike as a general method is useful for all supervised learning algorithms of spike trains.

Here, we use the example of Fig. 4 to explain pair-spike well. \( t_d^1 \) and \( t_d^2 \) in FITI are unsellected by using pair-spike to compute with \( t_d^4 \) to increase \( w_i \). Thus, \( t_d^1 \) and \( t_d^2 \) earlier than \( t_d^4 \) in FITI are selected to only compute with \( t_d^4 \) to decrease \( w_i \). \( t_d^3 \) and \( t_d^4 \) later than \( t_d^4 \) but earlier than \( t_d^3 \) in LITI are selected to only compute with \( t_d^3 \) to increase \( w_i \). The final update value of \( w_i \) in the first learning epoch is obtained as:

\[
\Delta w_i = \Delta w_i(t_d^1) + \Delta w_i(t_d^3) + \Delta w_i(t_d^4) + \Delta w_i(t_d^4)
\]

\[
= -\phi(t_d^2 - t_d^2) - \phi(t_d^3 - t_d^3) + \phi(t_d^4 - t_d^4) + \phi(t_d^4 - t_d^4)
= \phi(t_d^4 - t_d^4) + \phi(t_d^4 - t_d^4) - \left( \phi(t_d^1 - t_d^1) + \phi(t_d^1 - t_d^1) \right)
\]

(12)

The accumulation of input spikes fired at \( t_d^1 \) and \( t_d^2 \) is enough to produce an output spike at \( t_d^1 \) whereas only the accumulation of input spikes fired at \( t_d^3 \) and \( t_d^4 \) is insufficient to produce an output spike at \( t_d^3 \). Thus, \( \phi(t_d^3 - t_d^4) + \phi(t_d^4 - t_d^4) \) is greater than \( \phi(t_d^3 - t_d^4) + \phi(t_d^4 - t_d^4) \) in (12). \( w_i \) finally increases, which is our desired adjustment.

After we exchange \( t_d^2 \) and \( t_d^4 \) in Fig. 4, the computation of \( t_d^1 \) and \( t_d^2 \) with \( t_d^4 \) is removed by using pair-spike. \( t_d^1 \) and \( t_d^2 \) earlier
than \( t_j^1 \) are selected to only compute with \( t_j^1 \) to increase \( w_i \)
whereas \( t_i^3 \) and \( t_i^4 \) later than \( t_j^1 \) but earlier than \( t_i^4 \) are selected
to only compute with \( t_i^4 \) to decrease \( w_i \). Then,
\[
\Delta w_i = \Delta w_i(t_j^1) + \Delta w_i(t_i^2) + \Delta w_i(t_i^3) + \Delta w_i(t_i^4)
\]
\[
= \phi(t_j^1 - t_i^1) + \phi(t_i^2 - t_i^3) + \phi(t_i^3 - t_i^4)
\]
\[
= \phi(t_j^1 - t_i^1) + \phi(t_i^2 - t_i^3) + \phi(t_i^3 - t_i^4)
\]
\[
\text{as in (13), which can directly avoid}
\]
\[
\text{the wrong adjustment in (9).}
\]

Only the accumulation of input spikes fired at \( t_i^1 \) and \( t_i^2 \)
is insufficient to produce an output spike at \( t_j^1 \). The accu-
cumulation of input spikes fired at \( t_j^3 \) and \( t_j^4 \) is also insuffi-
cient to produce an output spike at \( t_j^1 \). The total decrease
\[
\phi(t_j^1 - t_i^1) + \phi(t_i^2 - t_i^3) + \phi(t_i^3 - t_i^4)
\]
\[
\text{in (13), can which directly avoid}
\]
\[
\text{the wrong adjustment in (9).}
\]

C. AN ALGORITHM USING SPIKE TRAIN KERNEL BASED
ON THE UNIT OF PAIR-SPIKE

Kernel method (KM) [46] is one of the effective ways to solve
the problem of non-linear pattern. The core idea of KM is to
convert the original data into an appropriate high-dimensional
kernel feature space by using a non-linear mapping, which
can discover the relationship among data [47], [48]. Suppose
\( x_1 \) and \( x_2 \) are two points in the initial space. The nonlinear
function \( \lambda(\cdot) \) realizes the mapping from an input space \( X \) to a
feature space \( P \). The kernel function can be expressed as the
following inner product form:
\[
\kappa( x_1, x_2 ) = \langle \lambda( x_1 ), \lambda( x_2 ) \rangle 
\]
where \( \kappa( x_1, x_2 ) \) is a kernel function. For the firing times \( t^m \)
and \( t^n \) of two spikes, the kernel function form is given by:
\[
\kappa( t^m, t^n ) = \langle \delta( t - t^m ), \delta( t - t^n ) \rangle , \forall t^m, t^n \in \Gamma 
\]
where \( \kappa \) must be symmetric, shift-invariant and positive de-
finitive such as Gaussian kernel, Laplacian kernel [2], [47].

By combining (15), \( \phi(\cdot) \) in (11) is set to the kernel function.
Then, (11) can be expressed as follows:
\[
\Delta w_i = \left\{ \begin{array}{ll}
\eta \sum_{f=1}^{F_i} \left[ \kappa( t_{\min}^f, t_i^f ) \left( t_a^h - t_a^g \right) / \left| t_a^g - t_a^h \right| \right], & t_a^g \neq t_a^h \\
0, & t_a^g = t_a^h 
\end{array} \right. 
\]
(16)

where \( \kappa(\cdot) \) is a kernel function. Since the use of spike train
kernel is based on the unit of pair-spike, we denote this
algorithm of (16) as unit of pair-spike (UPS).

IV. SIMULATION RESULTS

In this section, we carry out several experiments. Firstly, we
implement a simple example to show the learning process.
Then, we make a comparison to demonstrate the affection
of pair-spike. Moreover, we analyze some learning factors that
exert influences on the learning performance of UPS, includ-
ing the learning kernel \( \kappa(\cdot) \), LR, and learning epoch. Finally,
UPS compares with other algorithms.

The parameters of SRM are the time constant of post-
synaptic potential \( \tau = 7 \) ms, time constant of refractory period
\( \tau_R = 80 \) ms, and neuronal threshold \( V_{\text{thresh}} = 1 \) mV. Due to
all experiments are implemented by the computer program
with the discrete time, SRM has to progress in a discrete
mode. In this paper, the discrete precision is 0.1 ms.

In order to evaluate the learning performance, a
correlation-based measure \( C \) is set to measure the distance
between the desired and actual output spike trains [49], [50].
Like Victor-Purpura distance [51], \( C \) is popular in many
papers [31]–[34], [38]–[40]. In all following experiments,
\( C = 1 \) means SRM precisely fires the desired output spike
train, and the maximum \( C \) is replaced by \( C \) only when \( C \) is
greater than the maximum \( C \).

A. A SIMPLE EXAMPLE OF UPS

In the simple example, \( \kappa(\cdot) \) is set to a Laplacian kernel [48].
Then, the mathematical representation of UPS is given as
follows:
\[
\Delta w_i = \left\{ \begin{array}{ll}
\eta \sum_{f=1}^{F_i} \left[ \exp \left( \frac{-\left( t_{\min}^f - t_i^f \right)}{\tau^+} \right) \right], & t_a^g \neq t_a^h \\
0, & t_a^g = t_a^h 
\end{array} \right. 
\]
(17)

where \( \tau^+ = 7 \) ms. The basic parameters are the number of
input spike trains \( N_I = 200 \) and the number of output
spike trains \( N_O = 1 \). Initial weights are generated as a
uniform distribution in an interval \( (0, 0.2) \). All input spike
trains and the desired output spike train are Poisson spike
trains generated randomly with a Poisson rate 20 Hz of a time
interval \( (0, T) \), where the length of all spike trains \( T \) is set to
100ms. Because this is a simple experiment, the number of
upper limited iterative epoch \( U \) is set to a smaller number 100.

The complete learning process is presented in Fig.5 (a),
including the initial output spikes (\(^\star\)), desired output spikes
(pentagram), actual output spikes after the learning of last
epoch (\( \times \)), and actual output spikes at all learning epochs
during learning process (\( \cdot \)). As shown in Fig.5 (a), there are
many incorrect spikes in the early period of this learning
process. With the number of incorrect spikes decreasing, all
actual firing times gradually tend to the desired times. Finally,
this learning successfully fires the desired spike train, which
proves the learning ability of UPS to spike train learning.
Fig. 5(b) shows the change of \( C \) in this learning process.
With three obvious oscillations, \( C \) has an upward trend. Finally,
UPS learns successfully at 63 epochs. The oscillation from
20 epochs to 30 epochs is sharper than others. By combining
the learning accuracy with the learning process, we can see
that the incorrect spikes are generated later than 60 ms from
20 epochs to 30 epochs. At the same time, the firing times
between 40 ms and 50 ms fluctuates obviously. From the
appearance to the disappearance of these incorrect spikes, this
is exactly the performance of an effective regulation.
The rhythmic generation of spike doublet discharge is one of main reasons for the above oscillatory burst pattern [52]. The spike doublet also appears in the learning algorithms of spike count and spike time. In fact, the subset of spikes plays a predominant role in the regulation process of synaptic weights [53]. That is to say, UPS with pair-spike is more oscillatory than other algorithms without pair-spike, which can also be seen by comparing Fig. 5(b) with other figures of learning accuracy in [25], [27], [32]–[34] et al. However, spike doublet is unrelated to the initial condition of synaptic weight. The converging of these weights is related to the input set and parameters. The coupling temporal correlation can lead to the spike doublet just in certain parameter regimes [53]. Thus, by setting some suitable parameters, such as $LR$, we can avoid the oscillation or limit it in a small range.

### B. EFFECT OF PAIR-SPike FOR UPS

We carry out a group of experiments to compare the difference between UPS with multi-spike and UPS with pair-spike. In these experiments, $N_I$ is set to 400 and $T$ is set to 400 ms. Due to the increase of experimental complexity, $U$ is set to a slightly larger number 500 and $LR$ is set to a smaller number 0.01. Other settings remain the same as the settings in section IV.A. Without any special explanation, 30 experiments are performed at each case in this paper.

The mean and standard deviation (SD) of maximum $C$ ($M_C$ and $SD_C$ respectively) are presented in Fig. 6(a), where $M_C$ and $SD_C$ are used to analyze the learning accuracy and its stability respectively. We can see from Fig. 6(a) that UPS with pair-spike has a bigger $M_C$ and a smaller $SD_C$ than UPS with multi-spike, which means the learning accuracy of UPS with pair-spike is higher and stabler. Especially, $SD_C$ of UPS with multi-spike is so more than $SD_C$ of UPS with pair-spike, which more reveals that multi-spike can sometimes lead to an abnormal regulation. Fig. 6(b) presents that UPS with pair-spike has the less $M_E$ and $SD_E$ than UPS with multi-spike, which means the learning efficiency of UPS with pair-spike is higher and stabler. Thus, it can be proved that pair-spike has a positive effect on the supervised learning of spike trains.

### C. INFLUENCES OF LEARNING FACTORS

#### 1) INFLUENCE OF LEARNING KERNEL

In the previous experiments, learning kernel is all set to a Laplacian kernel. However, different kernels may have
different effects on learning performance. Whereas there are many kernels, we select Gaussian, Laplacian, and Inverse Multi-quadratic (IM) frequently used to prove the influence of $\kappa(\cdot)$ [36], [47]. When $\kappa(\cdot)$ is a Gaussian kernel, the mathematical representation of UPS is given by,

$$
\Delta w_i = \begin{cases} 
\eta \sum_{f=1}^{F_i} \left[ \exp \left( -\frac{(t_{\text{min}} - t_f')^2}{2\tau_g \cdot \tau_h} \right) \right] \\
(t_{h_i}^g - t_{d_i}^g) / \left| t_{d_i}^g - t_{h_i}^g \right|, & t_{d_i}^g \neq t_{h_i}^g \\
0, & t_{d_i}^g = t_{h_i}^g 
\end{cases}
$$

(18)

where $\tau_g = 2$ ms. When $\kappa(\cdot)$ is an IM kernel, the mathematical representation is obtained as:

$$
\Delta w_i = \begin{cases} 
\eta \sum_{f=1}^{F_i} \left( \frac{1}{\sqrt{(t_{\text{min}} - t_f')^2 + c^2}} \right) \\
(t_{h_i}^g - t_{d_i}^g) / \left| t_{d_i}^g - t_{h_i}^g \right|, & t_{d_i}^g \neq t_{h_i}^g \\
0, & t_{d_i}^g = t_{h_i}^g 
\end{cases}
$$

(19)

where $c = 1$ ms. In these experiments, basic parameters $LR$s for Gaussian, Laplacian and IM kernel are set to 0.025, 0.01, and 0.015 respectively. Other settings remain the same with the settings in IV.B. Fig. 7 shows the experimental results.

Fig. 7 shows that UPS using each kernel all can learn well. With the biggest $M_C$ and smallest $M_E$, Gaussian kernel has the best performance. IM kernel makes the same accuracy with Laplacian kernel and has a better efficiency. With the smallest $M_C$ and biggest $M_E$, Laplacian kernel is worst relatively. Although the performance of Gaussian kernel is best, Gaussian kernel as the best one cannot represent the general standard of UPS. On the contrary, if UPS with Laplacian kernel has an excellent performance, we can prove the general advantage of UPS still further and the potential disadvantages can be more obviously for comprehensive analyses. To keep the generality of UPS, we select Laplacian kernel as a representation in all following experiments. Laplacian kernel can also prove the universal advantage of UPS better when UPS compares with other algorithms.

2) INFLUENCE OF LEARNING RATE
In the previous experiments, $LR$ is directly set by researchers. However, different values of $LR$s may bring different results for the sensitive learning of spike trains. In this section, we test $LR$ in the range of benchmark values to select the best $LR$. Except $LR$ is set to 0.0001, 0.0005, 0.001, 0.005 and 0.025, other experimental settings are unchanged. Fig.8 presents the learning results.

Fig.8 (a) shows that the learning accuracy with $LR = 0.0001$ is very low. With $LR$ increasing, $M_C$ increases, $SD_C$ decreases, and all $M_C$ can reach 0.9. It means a larger learning rate can lead to a higher and stabler learning accuracy and UPS has less dependence on $LR$ in a range. In order to validate this increment, we set a larger $LR = 0.015$. The learning result with $LR = 0.015$ are also presented in Fig. 8. However, $M_C$ decreases and $SD_C$ increases when $LR$ increases to 0.015.

Fig.8 (b) presents the influence of $LR$ on the learning efficiency. With $LR$ increasing, $M_E$ decreases firstly, then increases, and has a minimum with $LR = 0.01$, which shows $M_E$ has a negative correlation with $M_C$. Thus, an appropriate $LR$ in a scale can make a better learning performance. In the following experiments, $LR$ is set by a self-adaptive method used in [36], [40]. This method is based on the firing rate, which needs a minimum of benchmark interval of $FR$, a maximum of benchmark interval of $FR$, and benchmark $LR \eta^*$. Based on the above experimental parameters, results, and analyses, $\eta_{\text{min}}$, $\eta_{\text{max}}$, and $\eta^*$ are set to 20 Hz, 40 Hz, and 0.01 respectively. In addition, this self-adaptive method can also make those oscillations mentioned in IV.A in a very small range to obtain a better learning result [2].

3) INFLUENCE OF LEARNING EPOCH
In the previous experiments, $U$ is all set to a number no more than 500. However, a large $U$ may influence the learning
result. In this section, $U$ increases from 100 to 600 with an interval of 100. To obviously show the influence, Poisson rate of all spike trains is up to 40 Hz. Other basic parameters are the same with those in section IV.B. Fig. 9 shows the learning accuracy and efficiency.

We can see from Fig. 9(a) that $M_C$ increases and $SD_C$ decreases gradually along with an increasing $U$. It means more learning epochs can truly improve the learning accuracy. However, the same increase of $U$ has less and less influences on this improvement at the same time, which reduces the average effect of learning epochs. For example, $M_C$ can increase by 0.133 from $U = 100$ to $U = 300$ whereas $M_C$ can only increase by 0.029 from $U = 300$ to $U = 500$. Thus, the learning accuracy of UPS is not sensitive enough to the learning epoch that is enough, which is also shown in other algorithms such as ReSuMe [27], [32].

Fig. 9(b) presents the influence of $U$ on the efficiency. With $U$ increasing, $M_E$ and $SD_E$ all become larger and larger, have a positive correlation with $M_C$, and have a negative correlation with $SD_C$, which means an increase of computational cost and an unstable learning efficiency. Thus, $U$ should be set to a suitable value to satisfy the demand of valuable $M_C$, which can also reduce the computational cost.

D. COMPARISON OF UPS AND OTHER ALGORITHMS

In section IV.C, we analyze the influence of learning factors for UPS. On this basis, we appropriately select the parameters such as kernel, $LR$, and $U$. To better demonstrate the learning performance, UPS compares with other algorithms through more complex experiments of spike train learning in this section. The compared algorithms include ReSuMe, SPAN, and PSD using pair-spike. We select these algorithms because they belong to different types or they are independent of spiking neuron models. ReSuMe is based on the synapse plasticity when SPAN and PSD use different convolutions of spike trains. Algorithms based on the gradient descent are unused, because they usually require an analytical expression of state variables and much more learning epochs to obtain the higher learning accuracy [25]. Parameters of ReSuMe in [32] are the time constant $a = 0.002$, an amplitude $A = 1$, and the time constant $\tau_s = 7$ ms. The convolution kernel of SPAN is a $\alpha$ kernel where a time constant $\tau = 7$ ms [38]. The parameters for PSD algorithm are $\tau_s = 10$ ms [39]. The $\eta^*$ of ReSuMe, SPAN and PSD is set to 0.01, 0.0005, and 0.01 respectively. We take many experiments with some transitional attributes of spike trains, such as $N_I$, $T$, and $FR$, which can comprehensively verify the learning ability of spike trains [25], [27].
1) COMPARISON OF UPS AND OTHER ALGORITHMS WITH TRANSITIONAL NUMBER OF INPUT SPIKE TRAINS

This section includes two group of experiments, where every initial weight is at a random distribution with the interval (0, 0.2), and the output spike train and all input spike trains are Poisson spike trains with $T = 500$ ms. In the first group, $FR = 50$ Hz and $N_I$ increases from 100 to 500 with an interval of 50. Without any special explanations, $U$ of all experiments is set to 1000 for all complex experiments in section D.

Fig. 10(a) shows the learning accuracy. When $N_I$ is small, all input spike trains fail to provide enough spikes to make the spiking neuron model emit all desired output spikes. The learning ability of all four algorithms is worse. With $N_I$ increasing, the learning accuracy of all algorithms increases. Especially, UPS and ReSuMe have a relatively larger increase. However, when $N_I$ is less than 250, the learning accuracy of four algorithms is still lower. When $N_I$ is more than 200, all algorithms except PSD have a good learning ability, but the growth rate of learning accuracy of all algorithms is lower. In general, all algorithms can make $M_C$ tend to 1 when $N_I$ is close to 500. UPS has the higher and stabler learning accuracy than other three algorithms in most cases (150, 250, 350-500 input trains), which reveals the stronger ability of UPS to control spiking times.

Fig. 10(b) presents the learning efficiency, where $M_E$ of all four algorithms fluctuates within a certain range. With the increase of $N_I$, the learning efficiency of UPS increases firstly and then decreases whereas other algorithms have no obvious rules. However, the learning accuracy and learning efficiency of all algorithms have no special correlation. The learning efficiency of UPS is higher than that of other three algorithms in most cases (150 and 300-500 input trains), which also reveals the stronger ability of UPS to control spiking times.

In the second group of experiments, all input spike trains are Poisson spike trains with $FR = 10$ Hz and the desired spike train is a Poisson spike train with $FR = 100$ Hz. Since $FR$ of input spike trains is much smaller than $FR$ of the desired spike train, $N_I$ and its interval should be larger to show the trend. $N_I$ increases from 200 to 1000 with an interval of 100.

Fig. 11(a) presents the learning accuracy. We can see that the learning accuracy of all algorithms in Fig. 11(a) has the same trend with the learning accuracy in Fig. 10(a) when $N_I$ increases. All algorithms have the worse learning ability relatively when $N_I$ is few. With the increase of $N_I$, learning accuracy of all four algorithms increases and UPS, ReSuMe have relatively higher growth rates. However, the learning accuracy of all algorithms is still low when $N_I$ is less than 500. When $N_I$ is up to 500, all algorithms have the similar learning accuracy again. All algorithms can make the learning result tend to the success when $N_I$ is close to 1000. In general, UPS has the stronger ability to control the firing time of spikes because UPS has the higher learning accuracy than other three algorithms in most cases (300-400, 600-1000 input trains).

The learning efficiency is shown in Fig. 11(b), where four algorithms present different trends. With the increase of $N_I$, the learning efficiency of algorithms SPAN and PSD fluctuates within a certain range, whereas the learning efficiency of UPS and ReSuMe has a trend to decrease, especially when $N_I$ is no less than 500. Combining Fig. 11(b) with Fig. 11(a), we can see that it is from $N_I = 500$ that the learning accuracy of UPS and ReSuMe start to grow faster than that of SPAN and PSD again. UPS has the higher learning efficiency than other three algorithms in four cases (300-600 input trains), which shows that UPS also has a good efficiency to control the firing times of spikes.

2) COMPARISON OF UPS AND OTHER ALGORITHMS WITH TRANSITIONAL LENGTH OF SPIKE TRAINS

Two groups of experiments are carried out in this section, where $T$ changes from 100 ms to 1000 ms with an interval $I = 50$ ms. In the first group, $FR = 50$ Hz and $N_I$ increases from 100 to 500 with an interval of 50. Without any special explanations, $U$ of all experiments is set to 1000 for all complex experiments in section D.

Fig. 12(a) shows the learning accuracy. When $N_I$ is small, all input spike trains fail to provide enough spikes to make the spiking neuron model emit all desired output spikes. The learning ability of all four algorithms is worse. With $N_I$ increasing, the learning accuracy of all algorithms increases. Especially, UPS and ReSuMe have a relatively larger increase. However, when $N_I$ is less than 250, the learning accuracy of four algorithms is still lower. When $N_I$ is more than 200, all algorithms except PSD have a good learning ability, but the growth rate of learning accuracy of all algorithms is lower. In general, all algorithms can make $M_C$ tend to 1 when $N_I$ is close to 500. UPS has the higher and stabler learning accuracy than other three algorithms in most cases (150, 250, 350-500 input trains), which reveals the stronger ability of UPS to control spiking times.

Fig. 12(b) presents the learning efficiency, where $M_E$ of all four algorithms fluctuates within a certain range. With the increase of $N_I$, the learning efficiency of UPS increases firstly and then decreases whereas other algorithms have no obvious rules. However, the learning accuracy and learning efficiency of all algorithms have no special correlation. The learning efficiency of UPS is higher than that of other three algorithms in most cases (150 and 300-500 input trains), which also reveals the stronger ability of UPS to control spiking times.
of 100 ms. In the first group, $N_I$ is set to 400 and other settings remain the same as the settings in the first group of experiments in section IV.D1).

Fig. 12(a) reveals that all four algorithms can learn to fire. When $T$ is 100 ms, all algorithms can learn well. However, with the gradual increase of $T$, their learning accuracies decrease. When $T$ is less than 400 ms, the learning accuracy of UPS, ReSuMe, and SPAN is still similar. At the same time, the learning accuracy of PSD decreases rapidly. Since $T$ is up to 400 ms, the difference of learning accuracy between SPAN and UPS has become obvious. The gap of learning accuracy between UPS and ReSuMe also increases gradually when $T$ is more than 600 ms. In general, ReSuMe and UPS have the higher learning accuracy than SPAN and PSD. Especially, UPS has the higher learning accuracy than other three algorithms in all cases. ReSuMe has the stabler learning accuracy than other three algorithms in four cases (300, 500, 700, and 900 ms) while UPS has the stabler learning accuracy than other three algorithms in three cases (100, 400, and 600 ms). Therefore, UPS has the stronger ability than other algorithms to control the firing time of spikes.

Fig. 12(b) shows the learning efficiency. With $T$ increasing, the learning efficiency of all algorithms decreases and tends to be stabler gradually. PSD has the lowest but stabllest learning efficiency in most cases. Generally, UPS has a good learning efficiency to control the firing time of spikes.
because UPS has the higher learning efficiency than other three algorithms in three cases (200, 500, and 700 ms) and an approximately highest learning efficiency in other three cases (400, 600, and 800 ms).

In the second group of experiments, \( N_I \) is set to 600. Other settings remain the same as the settings in the second group of experiments in section IV.D1). Fig. 13 shows the learning results.

We can see from Fig. 13(a) that all four algorithms have a good learning accuracy when \( T = 100 \text{ ms} \). However, the learning accuracy of all algorithms gradually decreases when \( T \) increases, especially from \( T = 500 \text{ ms} \) to \( T = 700 \text{ ms} \). The difference between SPAN and UPS in the learning accuracy has become obvious since \( T = 200 \text{ ms} \). The gap between ReSuMe and UPS in the learning accuracy gradually increases when \( T = 300 \text{ ms} \). Although the learning accuracy of all algorithms in Fig. 13(a) has the same trend as that in Fig. 12(a), the difference of learning accuracy between four algorithms remains within a certain range in Fig. 13(a). Especially, SPAN even has the higher learning efficiency than ReSuMe when \( T = 900 \text{ ms} \) and \( T = 1000 \text{ ms} \). In general, UPS has higher learning accuracy in all cases and the stabler learning accuracy in three cases (100, 200, and 1000 ms) than other three algorithms, so UPS has a stronger ability to control the firing time of spikes.

Fig. 13(b) presents the learning efficiency. With an increase of \( T \), the learning efficiency of UPS and ReSuMe fluctuates within a certain range while the learning efficiency of PSD has a trend to decrease. When \( T = 1000 \text{ ms} \), the difference of learning accuracy between SPAN and UPS is less than their difference when \( T = 100 \text{ ms} \), but the difference of learning efficiency between SPAN and UPS also reaches a maximum at the same time. Thus, even if SPAN has the same learning efficiency with UPS, UPS still has the higher learning efficiency than SPAN. In general, UPS has the higher learning efficiency than other three algorithms in four cases (100-300, 500-600, and 800 ms), which can reveal the good efficiency of UPS to control the firing time of spikes.

3) COMPARISON OF UPS AND OTHER ALGORITHMS WITH TRANSITIONAL FIRING RATE OF SPIKE TRAINS

There are two groups of experiments in this section, where \( FR \) of all spike trains changes from 10 Hz to 100 Hz with an interval of 10 Hz. In the first group of experiments, \( N_I \) is set to 400 and other settings remain the same as the settings in the first group of experiments in section IV.D1). Fig. 14 shows the experimental results.

Fig. 14(a) shows the learning accuracy. When \( FR \) is few, all algorithms can learn well. However, with the increase of \( FR \), more and more complex spike trains make the learning accuracy of all algorithms gradually decrease. The stability of learning accuracy of all algorithms also becomes worse when \( FR \) increases. By comparing different algorithms, we can see that the gap of learning accuracy among ReSuMe, PSD, and UPS gradually increases since \( FR = 30 \text{ Hz} \) while the difference of learning efficiency between SPAN and UPS also reaches a maximum at the same time. On the whole, UPS has the higher learning accuracy than other three algorithms in all cases except \( FR = 20 \text{ Hz} \) and stabler learning accuracy than other three algorithms in four cases (100, 300, 400, and 600 ms). Besides, UPS can learn to precisely output the desired spike train in 18 experiments and \( M_C \) is very close to 1 in other 12 experiments when \( FR = 10 \text{ Hz} \). In the same condition, SPAN and ReSuMe can both successfully learn in 15 experiments, and PSD just can successfully learn in 5 experiments. Therefore, UPS has the stronger ability to control the firing time of spikes than other three algorithms.

Fig. 14(b) presents the learning efficiency. With the increase of \( FR \), the stability of learning efficiency of each algorithm is better and better. For example, SD\(_E\) of
ReSuMe decreases from 295 with $FR = 10$ Hz to 101 with $FR = 100$ Hz. However, the learning efficiency of all algorithms decreases at the same time. For example, $M_E$ of UPS increases from 544 with $FR = 10$ Hz to 839 with $FR = 100$ Hz. In general, UPS has the higher learning efficiency than other three algorithms in most cases (20-50, 70 and 90-100 Hz), so UPS has the stronger ability to control the firing time of spikes.

In the second group, $s_d$ is a linear spike train [25]. Other settings remain the same as the settings in the first group of experiments in section IV.D3). The Poisson input and linear output can reveal the ability of directional and practical learning [2].

Fig. 15(a) presents the learning accuracy. We can see that although the learning accuracy of all algorithms in Fig. 15(a) has the same trend with the learning accuracy in Fig. 14(a), the learning accuracy in Fig. 15(a) all has an improvement in each case. For instants, $M_E$ of PSD is 0.752 in Fig. 14(a) whereas $M_E$ of PSD is 0.909 in Fig. 15(a) when $FR = 100$ Hz. Because $M_E$ of all algorithms is more than 0.9 in every case, all algorithms have a good learning accuracy. However, no algorithm can learn to precisely output the desired spike train whatever the $FR$ is. In general, UPS has the higher learning accuracy than other three algorithms in all cases and has the better stability of learning accuracy than other algorithms in all cases except $FR = 10$ Hz, which
reveals that UPS has the stronger ability for the directional and practical learning.

Fig. 15(b) presents the learning efficiency. We can see that the learning efficiency of all algorithms in Fig. 15(b) has the same trend as the learning efficiency in Fig. 14(b). In general, there is a negative correlation between FR and the learning efficiency as a whole. The learning efficiency of UPS is higher than other three algorithms in most cases (10-20 and 60-100 Hz), which also reveals the stronger ability of UPS for the directional and practical learning.

V. DISCUSSION
In this paper, using the spike train kernel based on a unit of pair-spike, we propose a supervised learning algorithm. The proposed algorithm named UPS uses the kernel method of pair-spike to directly adjust the weights. All experimental results indicate that UPS is an effective algorithm. Moreover, we can obtain two conclusions from these results. Firstly, UPS has the relatively higher learning accuracy. When \( N_I \) is enough, \( T \) is short or \( FR \) is fewer, UPS and other algorithms all can learn with the higher learning accuracy. As \( T, FR \) increase or \( N_I \) decreases gradually, UPS has the higher learning accuracy than other algorithms in most cases. Secondly, UPS has a good learning efficiency. In section IV.D, the learning efficiency of UPS is higher than that of other three algorithms in 31 of all 58 cases, which reveals that UPS has the good efficiency to spike train learning. Another advantage of UPS is the independence. The derivation of UPS is only based on the firing time of spikes and independent of the spiking neuron models.

Generally, many algorithms consume the computational resource on a useful input spike many times in every learning epoch. Following the thought that all spikes in different intervals have different functions to regulate the weights, we analyze the problem of original spike selection and computation method named multi-spike and its causes. Then, we use every useful input spike once in every learning epoch to construct a direct spike selection and computation method named pair-spike. The derivation of pair-spike is only on the basis of spike trains, so pair-spike adapts to all supervised learning algorithms of spike trains. Experimental results in section IV.B prove that pair-spike can improve the learning accuracy and efficiency. Moreover, compared with multi-spike, pair-spike also reduces the computational cost. The computational cost of multi-spike in every learning epoch is based on the number of input spikes and the number of all output spikes. For the step alone, the time complexity of multi-spike is \( O(n^2) \) at least. If we regard the time complexity of a circle of the two output spike trains \( s_{q} \) and \( s_{d} \) as \( O(n) \), the time complexity of multi-spike is \( O(n^2) \). However, the computational cost of pair-spike in every learning epoch is only based on the number of input spikes. The number of computational times in an epoch is equal to the number of input spikes earlier than the last output spike and no more than the number of total input spikes. For the step alone, the time complexity of pair-spike is only \( O(n) \). Therefore, with the same software engineering techniques, pair-spike has a computational advantage than multi-spike, even if pair-spike may need little more learning epochs than multi-spike.

Kernel method plays an important role in UPS. In section IV.C.1), we test Gaussian kernel, Laplacian kernel, and IM kernel. These kernels all can make an effective learning result. Some other kernels such as the linear kernel and Sigmoid kernel are unsuitable for UPS, because the learning accuracy of these kernels is lower or the learning is failed.

UPS also has some disadvantages such as local best. For example, it is difficult for UPS to reach \( C = 1 \) when \( C \) closes to 1 sometimes, which means UPS is failed to make a successful learning in theory. As shown in Fig. 9, due to the average effect of every epoch for the learning accuracy decreases with the learning epoch increasing, \( M_C \) has never reached 1 after 600 epochs based on 30 experimental statistics. In fact, this problem is not unique for UPS, which also exists in other learning algorithms [31]–[41]. For instants, the learning accuracy of all algorithms has the problem in the second group of experiments in section IV.D3). Besides, UPS sometimes truly needs a little more epochs to obtain the greater maximum \( C \) close to 1 than other algorithms such as ReSuMe with pair-spike in the second group of experiments in section IV.D2). In our view, some methods such as multiple kernels or multiple convolutions may avoid these problems of UPS [2].

VI. CONCLUSION AND FUTURE WORK
A supervised learning algorithm called UPS is constructed by using the spike train kernel based on a unit of pair-spike in this paper. Combining the advantages of kernel method and the direct computation of our proposed method pair-spike, UPS regulates the weights directly during the running process. As an effective and efficient supervised learning algorithm, UPS shows a better learning performance than several existing algorithms. In the future work, the above problems of UPS are important aspects to solve, where a higher accuracy with the fewer epochs is our desired result. Meanwhile, the relationship between pair-spike and different types of algorithms will be studied deeply and concretely in many aspects such as sensitivity and robustness. Moreover, the proposed algorithm UPS expanded for SNNs in future will be constructed and utilized in some actual databases and practical applications such as image classification, urban computing, and biomedical signal processing.

ACKNOWLEDGMENT
The authors would like to thank editors and reviewers for this manuscript. They would also like to thank members of Urbaneuron lab for their discussions, especially Prof. Xianghong Lin and Ziyi Chen for designing images in section II. Competing interests: The authors declare no conflict of interest.
[45] M. Zhang, H. Qu, A. Belatreche, Y. Chen, and Z. Yi, “A highly effective and robust membrane potential-driven supervised learning method for spiking neurons,” IEEE Trans. Neural Netw. Learn. Syst., vol. 30, no. 1, pp. 123–137, Jan. 2019.

[46] K. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf, “An introduction to kernel-based learning algorithms,” IEEE Trans. Neural Networks., vol. 12, no. 2, pp. 181–201, Mar. 2001.

[47] I. M. Park, S. Seth, A. R. C. Paiva, L. Li, and J. C. Principe, “Kernel methods on spike train space for neuroscience: A tutorial,” IEEE Signal Process. Mag., vol. 30, no. 4, pp. 149–160, Jul. 2013.

[48] T. Tezuka, “Multineuron spike train analysis with R-convolution linear combination kernel,” Neural Netw., vol. 102, pp. 67–77, Jun. 2018.

[49] S. Schreiber, J. M. Fellous, D. Whitmer, P. Tiesinga, and T. J. Sejnowski, “A new correlation-based measure of spike timing reliability,” Neurocomputing, vols. 52–54, pp. 925–931, Jun. 2002.

[50] J. A. Freund and A. Cerquera, “How spurious correlations affect a correlation-based measure of spike timing reliability,” Neurocomputing, vol. 81, pp. 97–103, Apr. 2012.

[51] J. D. Victor and K. P. Purpura, “Metric-space analysis of spike trains: Theory, algorithms and application,” Netw., Comput. Neural Syst., vol. 8, no. 2, pp. 127–164, 2009.

[52] N. Lemon and R. W. Turner, “Conditional spike backpropagation generates burst discharge in a sensory neuron,” J. Neurophysiol., vol. 84, no. 3, pp. 1519–1530, Sep. 2000.

[53] G. Hernández, J. Rubin, and P. Munro, “The effect of spike redistribution in a reciprocally connected pair of neurons with spike timing-dependent plasticity,” Neurocomputing, vols. 52–54, pp. 347–353, Jun. 2003.

GUOJUN CHEN was born in Dezhou, Shandong, China, in 1991. He received the B.S.E. degree from Shijiazhuang Tiedao University, Shijiazhuang, China, in 2014, and the M.Eng. degree in software engineering from Northwest Normal University, Lanzhou, China, in 2016. He is currently pursuing the Ph.D. degree with the School of Urban Design, Wuhan University, Wuhan, China. He is the author of several articles and a reviewer of several international journals. His current research interests include neural networks and urban computing.

GUOEN WANG received the B.S., M.S., and Ph.D. degrees from Tongji University, Shanghai, China, in 1985, 1990, and 2006, respectively. From 1997 to 2005, he was an Associate Professor with the School of Architecture and Urban Planning, Huazhong University of Science and Technology, Wuhan, China. Since 2011, he has been a Professor with the School of Urban Design, Wuhan University, Wuhan. He is the author of several books and more than 100 articles. His current research interests include urban planning, urban computing, and neural networks.