Int. J. Appl. Math. Comput. Sci., 2025, Vol. 35, No. 3, 493–505
DOI: 10.61822/amcs-2025-0035

ITERATION OVER EVENT SPACE IN TIME–TO–FIRST–SPIKE SPIKING NEURAL NETWORKS FOR TWITTER (𝕏) BOT CLASSIFICATION

MATEUSZ PABIAN [a], DOMINIK RZEPKA [a], MIROSŁAW PAWLAK [a,b,*]

[a]Department of Measurement and Electronics
AGH University of Krakow
al. Mickiewicza 30, 30-059 Kraków, Poland

[b]Department of Electrical and Computer Engineering
University of Manitoba
75 Chancellors Circle, Winnipeg, MB, R3T 5V6, Canada
e-mail: Miroslaw.Pawlak@umanitoba.ca

This study proposes a variant of a time-coding time-to-first-spike spiking neural network (SNN) model with its neurons capable of generating spike trains in response to observed event sequences. This extends an existing model that is limited to generating and observing at most one event per synapse. We explain spike propagation through a model with multiple input and output spikes at each neuron, as well as design training rules for end-to-end backpropagation for event sequence data. The model is trained and evaluated on a Twitter (𝕏) bot detection task where the time of events (tweets and retweets) is the primary carrier of information. This task was chosen to evaluate how the proposed SNN deals with spike train data composed of hundreds of events occurring at timescales differing by almost five orders of magnitude. The impact of various preprocessing steps and training hyperparameter choice on model classification accuracy is analyzed in an ablation study.

Keywords: spiking neural networks, event-based computing, bot detection, supervised learning.

1. Introduction

Spiking neural networks (SNNs) differ from classic artificial neural networks in how the signal is represented and propagated through the network. This in turn determines their applicability to event data (i.e., characterized by event occurrence as the primary information carrier, or data that can be represented in this form by a proper encoding). The SNN process data using impulses which are asynchronously propagated through the network (Pfeiffer and Pfeil, 2018; Nunes *et al*., 2022). This processing scheme stands in contrast to the classic artificial neural networks which require that all neurons within a single layer must finish their computation before the signal flows to the subsequent layer. Overall, the aim of the SNN is to model biological networks in a much more principled way. The training rules for the SNN generally fall into one of three categories: network conversion (mapping each component of the source network to its spiking equivalent (e.g., Rueckauer *et al*., 2017; Midya *et al*., 2019; Stöckl and Maass, 2021)), synaptic-plasticity-aware training (taking advantage of long-term potentiation and long-term depression effects (e.g., Falez *et al*., 2019; Mozafari *et al*., 2019)), or training with backpropagation (e.g., Mostafa, 2017; Wu *et al*., 2018; Rasmussen, 2019). The latter approach necessitates formulating custom training rules that take into account the fact that the activation function of biological neurons—the "all-or-nothing" principle—is not differentiable.

In this paper we use backpropagation to train an SNN based on the time-to-first-spike neurons introduced by Mostafa (2017). We address one of the limitations of this model related to the infinitely-long neuron refractory period τ_{ref}. This shortcoming means that every neuron in the model can elicit at most one spike during a single input example presentation. Crucially, the proposed signal propagation and network training rules are truly event-centric, evaluating the state of the model at each event rather than at fixed points in time determined by the

*Corresponding author

simulation grid. We evaluate our approach by training a classification network on a real-life dataset of legitimate and automated Twitter (𝕏)[1] user activity (Mazza *et al*., 2019). We choose this data for several reasons. First of all, each record is described only in terms of the time of event without any auxiliary information (such as tweet content or sentiment). Secondly, each data sample, i.e., an event sequence (or interchangeably: a spike train) comprised of all records of a given user, is several hundred events long. Lastly, the events occur at timescales differing by many orders of magnitude. This allows us to assess the feasibility of the proposed approach in scenarios commonly encountered when processing this type of data.

The rest of this paper is structured as follows. Sections 1.1 and 1.2 present a brief overview of backpropagation-based algorithms for training the SNN, as well as introduce the Twitter bot detection problem. Section 2 describes the signal propagation and training rules for the proposed SNN, with the main contributions presented in Sections 2.2 and 2.3. The dataset properties, training data selection and preprocessing steps are described in Section 3. Finally, we summarize our results in Section 4, showing the impact of model parameters on model classification accuracy.

1.1. Related works on backpropagation-based training of the SNN. Training the SNN with backpropagation leverages existing methods and best practices developed for artificial nonspiking neural networks and deep learning (Eshraghian *et al*., 2023). In doing so it forgoes biological plausibility, evident in training methods based on spike-timing-dependent plasticity effects (Falez *et al*., 2019).

One of the first gradient-descent-based learning rules for timing-encoding networks was the SpikeProp algorithm (Bohte *et al*., 2002) and its variants. It established key characteristics of backpropagation algorithms present in subsequent works: (i) the SNN is simulated over a finite time window with a fixed time step, recording changes in the internal state (membrane voltage, synaptic current) of all neurons during the forward pass, and (ii) approximating (smoothing) the spike-generation function to avoid a discontinuous derivative (unless this discontinuity is ignored and treated as noise (e.g., Lee *et al*., 2016)). Backpropagation-based approaches for training time-coding SNN can thus be divided into two categories, depending on how much information from the forward pass is needed to compute the backward pass. The first category is event-driven learning, where the error is propagated only through spikes (e.g., Huh and Sejnowski, 2018; Zhang and Li, 2020; Zhu *et al*., 2022). Most notably, EventProp (Wunderlich and Pehle, 2021) defines exact gradients and can be applied to neuron models without an analytical expression for the postsynaptic potential kernels. Nevertheless, similarly to other existing algorithms, EventProp still requires simulating the network in the forward pass in order to compute postsynaptic events. As these algorithms propagate the error only through spike events, they are prone to fail to converge when the network does not generate enough events to process the signal end-to-end.

In contrast to event-driven learning is the RNN-like learning (named after its similarity to nonspiking artificial recurrent neural networks), in which the error information is also propagated through computation time steps which did not elicit a spike (Wu *et al*., 2018; Zheng *et al*., 2021). The SuperSpike (Zenke and Ganguli, 2018) and SLAYER (Shrestha and Orchard, 2018) models are examples of such algorithms. Surrogate gradient methods, commonly used in the RNN-like learning, are an alternative approach for overcoming the discontinuous derivative of the spike-generating function (Neftci *et al*., 2019). Typically, a standard backpropagation-through-time algorithm (Werbos, 1990) is used, as in the RNN, with one minor modification: a continuously differentiable function is used in the backward pass as a surrogate of the spike-generation function derivative. Finding the optimal surrogate gradient function is a topic of an ongoing research (Yin *et al*., 2021). Despite achieving a remarkable success in training deep SNNs (e.g., Fang *et al*., 2021; Kim *et al*., 2022), surrogate gradient methods represent an even further departure from energy-efficient, biologically-inspired learning.

Recently, there has been some research conducted on single-spike time-to-first-spike SNN that specify learning rules which do not require simulating the network over time (e.g., Mostafa, 2017; Kheradpisheh and Masquelier, 2020; Zhou *et al*., 2021). Concretely, Mostafa (2017) trains an SNN with simplistic Integrate-and-Fire (IF) neurons by deriving locally exact gradients of the spike-generating function. A similar idea was explored by Kheradpisheh and Masquelier (2020), who used the instantaneous synaptic current kernel function instead of an exponentially decaying kernel. Furthermore, Zhou *et al*. (2021) applied popular deep learning techniques such as max-pooling and batch normalization to this model type, which alleviates issues with training deeper neural architectures. However, to the best of our knowledge, there has been no research that shows how this time-to-first-spike SNN model can be applied to actual spike trains rather than single spikes.

In this work we derive an algorithm that extends the aforementioned single-spike time-to-first-spike SNN training rules to neurons observing and generating spike

[1]Throughout this paper, we will refer to 𝕏 under its former name, i.e., Twitter, for consistency with previous works, and due to mentions of deprecated vocabulary (e.g., retweet) and services (e.g., Twitter Premium Search API).

trains. In doing so we not only infer locally exact gradients of the spike-generating function, like the EventProp algorithm does, but also express the entire computation in terms of iteratively computing successive spikes (first, second, etc.). This stands in stark contrast to other works which simulate the state of the entire network over a finite time window with a fixed time step.

1.2. Overview of Twitter bot detection. Social bots are automated agents that interact with humans and mimic human behavior (Ferrara *et al*., 2016). In social media environment, such bots may aggregate contents from various sources, automatically respond to custom queries, or even generate content that satisfies some constraints. However, some bot behavior and intent is purposefully misleading or malicious (Cresci *et al*., 2017).

Bot detection systems can be divided into three categories: crowdsourcing, feature-based classification, and social network analysis. Crowdsourcing techniques rely on the ability of human experts and volunteers to manually identify bot-user profiles, or on sending survey requests to the users and analyzing their replies (Cresci *et al*., 2017). Feature-based classification aims to automate the process by analyzing user-level features, usually by aggregating data pertaining to the user profile and to their activity (e.g., Davis *et al*., 2016; Rodríguez-Ruiz *et al*., 2020). It is also possible to perform unsupervised clustering of users based on the temporal similarity of their activity, regardless of their relationship in the social network (Mazza *et al*., 2019). This circumvents the time-consuming and costly process of labeling the data. Additionally, the unsupervised systems do not become outdated when new generations of bots, exhibiting patterns of behavior not present during model training, become more prevalent. However, they treat all unclustered examples as legitimate users, limiting their ability to detect bots exhibiting irregular behavior. The last of the three bot detection categories operates on a large group of users at once (e.g., Minnich *et al*., 2017; Balaanand *et al*., 2019). Similarly to the unsupervised user-level methods, they do not become obsolete due to data drift (Minnich *et al*., 2017). However, given that they analyze the actual community structure formed by users, they take more time to process information, making them more difficult to operate at scale (Cao *et al*., 2012).

As has been observed (e.g., Pan *et al*., 2016; Dutta *et al*., 2018; Mazza *et al*., 2019) automated user behavior can be made evident by analyzing user-level patterns of retweet activity (where a "retweet" is the re-broadcasted original message known as a "tweet"). We follow this approach in our study by allowing the tweet and retweet actions to be the only information available to the model.

2. Methods and algorithms

2.1. Signal propagation in time-to-first-spike SNN. This study focuses on a specific type of the SNN that is sensitive to the timing of input events and not their rate. Therefore, let us briefly summarize the model first described by Mostafa (2017). This type of network uses the IF neurons with exponentially decaying synaptic current kernels. The membrane voltage of an IF neuron with C presynaptic neuron connections is governed by the differential equation

$$\frac{\mathrm{d}V(t)}{\mathrm{d}t} = \sum_{c=1}^{C} w_c i_c(t)\,, \tag{1}$$

where w_c is the weight associated with the c-th synapse (channel), and $i_c(t)$ is the synaptic current driving signal. Assuming that every presynaptic neuron observes a single event (or interchangeably – a spike) at nonnegative time t_c, the presynaptic current is

$$i_c(t) = \exp\left(-\frac{t-t_c}{\tau_{\text{syn}}}\right) u(t-t_c),$$

with τ_{syn} being the synaptic current time constant and $u(t)$ being the step function. The solution to the system of equations defined by (1) is given by

$$V(t) = V_0 + \tau_{\text{syn}} \sum_{c=1}^{C} w_c \left[u(t-t_c) - i_c(t)\right]\,, \tag{2}$$

where $V_0 = V(0)$ is the initial membrane voltage. Note that the set of event times $\{t_c\}$ need not be ordered – the event order is induced by the step function $u(t)$. The neuron is said to fire at time t_{out} if the voltage exceeds a threshold V_{thr}, i.e., $t_{\text{out}} = \arg\min_t V(t) \geq V_{thr}$, after which the IF neuron becomes unresponsive to input signals for some time, called the refractory period τ_{ref}. Once the refractory period subsides, the voltage is reset to zero and the summation over impulses in (2) can resume. For now let us consider the case $\tau_{\text{ref}} \to \infty$, meaning that the neuron is unable to respond to new inputs following output spike generation. This is an implicit assumption in the single-spike SNN (e.g., Mostafa, 2017; Kheradpisheh and Masquelier, 2020; Zhou *et al*., 2021).

Assuming without the loss of generality $V_0 = 0$ and that there exists a subset of input spikes that cause the postsynaptic neuron to fire

$$Q = \{c : t_c < t_{\text{out}}\}\,, \tag{3}$$

i.e., the *causal set* of input spikes, the solution for t_{out} is given in the implicit form

$$z_{\text{out}} = \frac{\sum_{c\in Q} w_c z_c}{\sum_{c\in Q} w_c - \frac{V_{\text{thr}}}{\tau_{\text{syn}}}}\,, \tag{4}$$

where

$$z_c = \exp\left(\frac{t_c}{\tau_{\text{syn}}}\right), \quad z_{\text{out}} = \exp\left(\frac{t_{\text{out}}}{\tau_{\text{syn}}}\right). \tag{5}$$

For completeness we assign $z_{\text{out}} = \infty$ when $Q = \varnothing$. A necessary condition for the postsynaptic neuron to fire is that the sum of weights of the causal set of neurons be strictly larger than the scaled threshold voltage (Mostafa, 2017), i.e.,

$$1 \le z_{\text{out}} < \infty \iff \sum_{c\in Q} w_c > \frac{V_{\text{thr}}}{\tau_{\text{syn}}}. \tag{6}$$

This condition assures that the right-hand side of (4) is positive (Mostafa, 2017). The formula (4) is differentiable with respect to transformed input spike times $\{z_c\}$ and synaptic weights $\{w_c\}$ with partial derivatives

$$\begin{aligned}\frac{\partial z_{\text{out}}}{\partial z_c} &= \frac{w_c}{\sum_{c\in Q} w_c - \frac{V_{\text{thr}}}{\tau_{\text{syn}}}},\\ \frac{\partial z_{\text{out}}}{\partial w_c} &= \frac{z_c - z_{\text{out}}}{\sum_{c\in Q} w_c - \frac{V_{\text{thr}}}{\tau_{\text{syn}}}},\end{aligned} \tag{7}$$

if $c \in Q$ and zero otherwise; therefore, it can be used to train the SNN using the backpropagation algorithm.

The aforementioned description refers to signal propagation rules for a single time-to-first-spike spiking neuron. Each layer of the network is comprised of many such neurons, and it is assumed that all neurons observe an identical set of input spikes.

2.2. Multiple-input, multiple-output (MIMO) SNN. The model summarized in the previous section is a single-spike SNN, unable to process event sequence data. It assumes that every neuron, including input neurons, can elicit at most one spike (by implicitly setting $\tau_{\text{ref}} = \infty$). Our goal is to extend the signal propagation rules of that model so that all neurons are able to produce multiple spikes each. For brevity, we use a MIMO network descriptor throughout this section to refer to the model that supports signal propagation with multiple inputs and multiple outputs. In case of multiple events arriving at the input synapse, Eqn. (1) becomes

$$\frac{dV(t)}{dt} = \sum_{c=1}^{C} w_c \sum_{j=1}^{J_c} i_c^{[j]}(t) = \sum_{c=1}^{C}\sum_{j=1}^{J_c} w_c^{[j]} i_c^{[j]}(t), \tag{8}$$

where J_c is the number of events observed in channel c and

$$i_c^{[j]}(t) = \exp\left((t_c^{[j]} - t)/\tau_{\text{syn}}\right) \quad u(t - t_c^{[j]})$$

for $j = 1, \ldots, J_c, \quad c = 1, \ldots, C$. Note that this formula explicitly highlights the fact that every event $t_c^{[j]}$ of channel c is associated with the same weight $\{j : w_c^{[j]} = w_c\}$. Therefore, a single channel with J_c events is equivalent to J_c virtual (or time-flattened) channels with a single event each. Introducing a new index $\{k : k = 1, \ldots, K\}$ over these virtual input channels such that $K = \sum_{c=1}^{C} J_c$ we obtain the following representation of (8):

$$\frac{\mathrm{d}V(t)}{\mathrm{d}t} = \sum_{k=1}^{K} w_k i_k(t), \tag{9}$$

which is identical to (1). The set of time-flattened event times $\{t_k\}$ need not be ordered. It follows that the equations for forward and backward signal propagation through the network introduced in (4)-(7) still hold for the layer observing spike trains, provided that the indices are substituted where appropriate. The proposed idea of time-flattening the input signal and projecting input layer weights is presented in Fig. 1(a).

Allowing each neuron to spike more than once requires setting a finite nonnegative refractory period τ_{ref}. In that case the causal set of input spikes (3) becomes

$$Q_m = \begin{cases} \{k : t_k < t_{\text{out}}^{[m]}\} & \text{if } m = 1\\ \{k : t_{\text{out}}^{[m-1]} + \tau_{\text{ref}} < t_k < t_{\text{out}}^{[m]}\} & \text{otherwise}\end{cases}, \tag{10}$$

where $\{m : m = 1, \ldots, M\}$ is the index over the sequence of output spikes. Computing the m-th output spike time can be done iteratively, noting that M is at most equal to K (the neuron cannot spike more often than once per input spike). The iterative computation over m can be stopped early when an empty causal set is encountered ($Q_m = \varnothing$). Taking this new definition of a causal set into consideration, the implicit formula for $t_{\text{out}}^{[m]}$ becomes

$$z_{\text{out}}^{[m]} = \frac{\sum_{k\in Q_m} w_k z_k}{\sum_{k\in Q_m} w_k - \frac{V_{\text{thr}}}{\tau_{\text{syn}}}}, \tag{11}$$

where

$$z_k = \exp\left(t_k/\tau_{\text{syn}}\right), \quad z_{\text{out}}^{[m]} = \exp\left(t_{\text{out}}^{[m]}/\tau_{\text{syn}}\right).$$

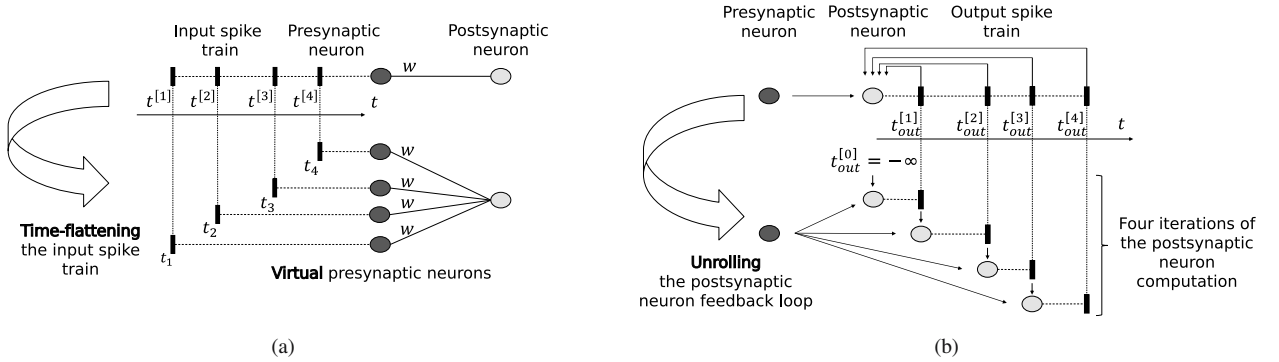
The partial derivatives are

$$\frac{\partial z_{\text{out}}^{[m]}}{\partial z_k} = \begin{cases} \frac{w_k}{\sum_{k\in Q_m} w_k - \frac{V_{\text{thr}}}{\tau_{\text{syn}}}} & \text{if } k \in Q_m\\ 0 & \text{otherwise}\end{cases}, \tag{12}$$

$$\frac{\partial z_{\text{out}}^{[m]}}{\partial w_k} = \begin{cases} \frac{z_k - z_{\text{out}}^{[m]}}{\sum_{k\in Q_m} w_k - \frac{V_{\text{thr}}}{\tau_{\text{syn}}}} & \text{if } k \in Q_m\\ 0 & \text{otherwise}\end{cases}, \tag{13}$$

and additionally

$$\frac{\partial z_{\text{out}}^{[m]}}{\partial w_c} = \sum_{\{k:\, w_k \triangleq w_c\}} \frac{\partial z_{\text{out}}^{[m]}}{\partial w_k}, \tag{14}$$



(a) (b)

Fig. 1. Signal propagation rules in the MIMO SNN: a presynaptic neuron with a weight w observing multiple input spikes can be represented as multiple virtual presynaptic neurons, each having that same weight w and observing a single spike (a), iterative computation of the output spike train by unrolling the postsynaptic neuron feedback loop: the computation is repeated for the same input spike sequence, but with different timestamps of the previously generated events. This cascade proceeds until it is impossible for the postsynaptic neuron to generate an output spike. The implicit output spike at $t_{\text{out}}^{[0]} = -\infty$ designates the initial state of the postsynaptic neuron, i.e., it has not generated a spike yet (b).

where the set $\{k : w_k \triangleq w_c\}$ denotes the subset of all k for which the virtual (time-flattened) weight w_k corresponds to the original w_c. Note the absence of any explicit dependence between two consecutive output spikes

$$\{t_{\text{out}}^{[m]}, t_{\text{out}}^{[m+1]}\}$$

in equations (11)–(14), or equivalently

$$\forall_m \, \partial z_{\text{out}}^{[m+1]} / \partial z_{\text{out}}^{[m]} = 0.$$

This is fundamentally different from signal propagation in the backpropagation-through-time algorithm (Werbos, 1990), and is a direct result of the IF neuron's independence on its own history (i.e., it is memoryless). Instead, the gradients are simply summed over output spikes. This concept of iterating over the M output spikes is summarized in Fig. 1(b).

Overall, the proposed algorithm shows how to simulate and train a time-coding MIMO SNN by time-flattening the presynaptic spike trains and unrolling the postsynaptic neuron feedback loop imposed by the spike causality principle. This procedure is sufficient to process information with spike trains. However, it must be stressed that these concepts specifically refer to operating the model on conventional hardware. Once trained, this SNN (i.e, synaptic weights and neuron-specific hyperparameters) can be realized on neuromorphic devices (Javanshir *et al*., 2022). The virtual presynaptic neurons are no longer needed when running the model on such devices (hence the name) as reusing neurons for multiple input and output events is implied.

2.3. MIMO SNN loss function. The original model proposed by Mostafa (2017) was trained using a risk function with two components. The first one is a modification of the cross-entropy loss with softmax normalization

$$L(z, y) = -\sum_{p=1}^{P} y_p \ln \left(\frac{\exp(-z_p)}{\sum_{p=1}^{P} \exp(-z_p)} \right), \qquad (15)$$

where

- P is the number of output channels,
- y_p is a binary indicator (0 or 1) of the desired output channel p spiking first,
- z_p is the transformed spike time of the p-th output channel.

This loss function encourages the model to use rank-order coding (Thorpe and Gautrais, 1998) at the output layer, without explicitly specifying the time of each spike. The second component of the risk function is a spike regularization term, which promotes network spiking activity by ensuring a nonnegative denominator in (4),

$$R_{\text{spiking}} = \sum_{h=1}^{H} R_h \,, \qquad (16)$$

where

$$R_h = \max \left(0, V_{\text{thr}}/\tau_{\text{syn}} - \sum_{c=1}^{C_h} w_{ch} \right),$$

with the sum $\sum_{c=1}^{C_h} w_{ch}$ being computed over the set of all synaptic connections $\{c : c = 1, \ldots, C_h\}$ between the given postsynaptic neuron h and its presynaptic neurons. The index h runs over all neurons in the network H. The overall risk function minimized by the model is

$$L_{\text{total}}(z, y) = \frac{1}{N} \sum_{n=1}^{N} L_n(z, y) + \gamma R_{\text{spiking}} \,, \qquad (17)$$

where $L_n(z, y)$ is the cross-entropy loss (15) for the n-th example of the batch of size N, and γ is a hyperparameter.

Some adjustments to the loss function components are necessary in order to use them in the MIMO SNN context. The formula (16) can be trivially extended to networks with spike train inputs by substituting the index over input neurons c with an index over virtual input channels k. We can also apply regularization to each neuron output spike m separately. Thus, the spiking activity regularization for the MIMO SNN is

$$R_{\text{spiking}} = \sum_{h=1}^{H} \sum_{m=1}^{M_h} R_h \,, \tag{18}$$

where

$$R_h = \max\left(0, V_{\text{thr}}/\tau_{\text{syn}} - \sum_{k=1}^{K_h} w_{kh}\right).$$

However, this implicitly assumes that every presynaptic weight w_{kh} is associated with an event t_{kh}. As was previously shown by Pabian *et al*. (2022) this need not be the case as the SNN can exhibit sparse neuron activity. It is entirely possible that for some neuron h the inequality

$$\sum_{k=1}^{K_h} w_{kh} > V_{\text{thr}}/\tau_{\text{syn}}$$

holds and yet it does not fire. This scenario might occur if none of the presynaptic neurons observe an event. Thus (18) can be reformulated to make this dependence on input spikes explicit:

$$R_{\text{spiking}} = \sum_{h=1}^{H} \sum_{m=1}^{M_h} R_{mh} \,, \tag{19}$$

where

$$R_{mh} = \max\left(0, V_{\text{thr}}/\tau_{\text{syn}} - \sum_{k \in B_{mh}} w_{kh}\right)$$

with $B_{mh} \subset Q_{mh}$ being the set of valid inputs for the m-th output of the h-th postsynaptic neuron

$$B_{mh} = \begin{cases} \{k : t_{kh} < \infty\} & \text{if } m = 1 \\ \{k : t_{\text{out}_h}^{[m-1]} + \tau_{\text{ref}} < t_{kh} < \infty\} & \text{otherwise} \end{cases}. \tag{20}$$

For completeness $R_{mh} = 0$ if $\{k : t_{kh} < \infty\} = \varnothing$. In our preliminary experiments we found that the penalty term is too strong for $m > 1$, skewing the training objective towards forcing neurons to output multiple spikes, rather than letting it focus on solving the actual task. As such, we train our models by setting $\forall_{m>1}\, R_{hm} = 0$, which only penalizes neurons that do not spike at all (alternatively one might consider applying a smaller weight to subsequent spikes).

On another note, the cross-entropy loss as defined by (15) ignores the fact that each output layer neuron can produce multiple spikes. This means that for the last layer, either the refractory period could be set to $\tau_{\text{ref}} = \infty$, or that all $m > 1$ output spikes could simply be ignored (as they have no impact on the gradient propagation anyway). Both approaches lead to the same result.

The final risk function used to train the MIMO SNN is parameterized by the set of weights $\{w_c\}$, and is obtained by substituting the regularization term in (17) with the spike regularization term computed over the set of valid inputs according to (19).

2.4. Comparison with existing SNN models and the MIMO SNN limitations. Let us briefly comment on the computational burden of the MIMO SNN. The proposed algorithm slightly increases the complexity of the single-spike version proposed by Mostafa (2017). Note that it is possible to compute the first output spike of all neurons in a given layer in a single computation pass (see the work of Zhou *et al.* (2021) for a batched-input version of this algorithm). While the proposed MIMO SNN approach introduces an unavoidable feedback loop resulting from the spike causality principle, it does not prevent the model from training with modern deep learning software frameworks. During such an iterative search over output spikes, the SNN is no different from a nonspiking RNN.

Furthermore, the proposed algorithm iterates over the space of discrete output spike events rather than discretizing time at a predefined time-resolution. This event space is finite as each neuron cannot generate more events than it observes across all synapses. This leads to significantly fewer iterations when computing the full output spike train of each neuron than the alternative requiring a discretized time simulation. Additionally, this allows MIMO SNN to compute arbitrarily late events w.r.t. reference time, whereas discretized-time algorithms are limited in scope by the simulation window.

Nevertheless, the MIMO SNN adversely scales with the number of events observed by presynaptic neurons in all layers. This might be problematic for the input layer, in which the number of virtual (time-flattened) channels can be extremely large when the network has numerous input neurons, each observing lengthy spike trains. This limits the applicability of our approach to multichannel data streams without too many events (although it is difficult to estimate what this upper limit actually is), unless the dataset is preprocessed to contain fewer events. Furthermore, the average processing time increases with the number of spikes generated by the layer as the feedback loop must be unrolled over time. However, note that each successive spike is less likely

to be generated (because the causal set shrinks with each iteration), making the computational complexity of subsequent iterations smaller than the preceding one. Fortunately, the network spiking activity can be controlled by the τ_{ref} hyperparameter.

3. Experimental setup

3.1. Dataset description. The RT_{BUST} study reported by Mazza *et al*. (2019) used Twitter Premium Search API to compile a list of all Italian retweets shared between 18 June 2018 and 1 July 2018. In this 2-week period there were almost 10M retweets shared by 1.4M distinct users. The compiled dataset[2] consists of records of retweet timestamps associated with some original tweets. These records can be aggregated based on user id such that each user is characterized by a different number of tweet-retweet pairs. Importantly, the dataset contains no information on the content of shared tweets, thus the goal is to classify legitimate and bot users based solely on retweet timestamps. To supplement this vast collection of unlabeled records, about 1000 accounts were manually annotated based on published annotation guidelines for datasets containing social bots (Cresci *et al*., 2017).

We limit the scope of our analysis to the labeled portion of this dataset and, in contrast to the original study, apply a supervised learning algorithm on individual data points. Let us introduce a *retweet delay* as a difference between retweet and the origin-tweet timestamps, expressed in minutes. This quantity can be treated as describing an observed event, hence we denote it as t in order to avoid introducing additional notation. We filter the records so that only those with both tweet and retweet timestamps occurring from 18 June 2018 to 1 July 2018 (exclusive) are kept, meaning that the retweet delay attribute is bounded from above to about $\tau_{\text{max}} = 2 \cdot 10^4$ minutes (2 weeks) [3]. Additionally, only records with a delay of at least $\tau_{\text{min}} = 10^{-1}$ minutes were considered for further analysis. Records with a retweet delay below the selected τ_{min} can be considered a sign of either auto-retweeting bots, or legitimate users that were refreshing their feed as the tweet was posted and decided to retweet immediately. Either way, in our opinion, this does constitute a typical user behavior. As a result of this filtering step, 11.73% of all records were removed.

Finally, the records were grouped by the retweeting user and by label. This allowed us to construct a set of data points $\boldsymbol{X}$ with elements $\boldsymbol{x} = [t_1, \ldots, t_U; U]$ such that $\tau_{\text{min}} < t_1 < \ldots < t_U < \tau_{\text{max}}$ are the retweet delays bounded by observation time τ_{max} and $U = U(\tau_{\text{max}})$ is the length of the sequence. We obtain a nearly balanced data subset with 366 examples of legitimate users and 389 examples of bot users. These user tweet-retweet sequences have, on average, about 113 events with a maximum of 698. The remaining 46,883 cases are unlabeled and were not used in this study.

3.2. Signal preprocessing. The tweet-retweet sequences obtained in the previous Section are characterized by two peculiar traits that make designing an SNN-based classifier difficult. First, we note that all events occur in only one channel. If all neurons in the network have the same value of parameters τ_{syn}, τ_{ref} and V_{thr}, then this scenario imposes additional constraints on the weights during training. First of all, the synaptic weight of each connection must be positive, otherwise the postsynaptic neuron is unable to produce any spike. This also means that each postsynaptic neuron eventually produces a spike as the IF neurons are unable to lose charge if all presynaptic weights are positive. Additionally, if some weights are too large, then it is possible that the associated postsynaptic neurons will produce nearly identical spike trains, with only slight variations in spike frequency and their timing (illustrated in Fig. 2). While it is certainly not impossible to train a network with such constraints on the input layer, we can expect this to be more difficult. In fact, in Section 4.2 we empirically show that training a model with a single input neuron is indeed more challenging than the alternative.

For these reasons we explored the possibility of transforming input spike trains into multiple sequences of fewer events. This is analogous to conducting feature engineering instead of relying on trainable feature extractors in artificial neural networks. Loosely inspired by the technique of binning used in neuroscientific studies, we identify a collection of *bins* that divide the spike train aggregated over all examples into sub-sequences with approximately the same number of events in each bin.[4] Importantly, the step that identifies binning thresholds is computed over the training examples. Each sub-sequence can then be shifted so that it starts at $t = 0$ by subtracting the corresponding binning threshold.

We note that binning is a valid strategy for this specific problem because it is assumed that events aggregated into a tweet-retweet sequence of each user are independent of one another (Mazza *et al*., 2019). While this is a reasonable assumption given the data, in reality it seems plausible that graph community structure and programmed bot behavior play an important role in what gets retweeted, and when. Additionally, the IF neuron exhibits the memoryless property where the internal state of the neuron (membrane voltage) is preserved regardless of how recent was the previous event. Finally, as a side

[2]The dataset is available at `https://doi.org/10.5281/zenodo.2653137`.

[3]Throughout the rest of this paper, we express all units of time in terms of minutes, same as the original RT_{BUST} study.

[4]By contrast, in neuroscience binning produces the number (or an average number of) events that had occurred in a given time interval over multiple repetitions of the experiment.

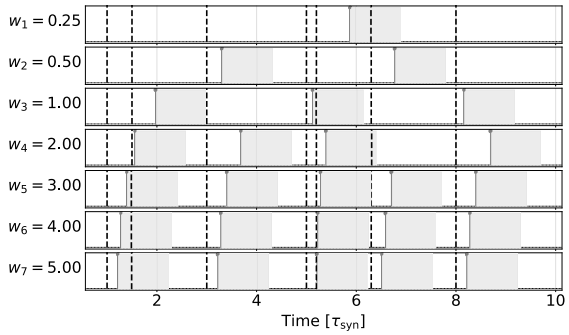


Fig. 2. Simulated spike trains from a simple network with one input neuron and seven postsynaptic neurons. Dashed black lines: input spike train (the same in all rows), solid gray lines: spikes generated by postsynaptic neurons. The shaded area denotes the refractory period after generating a spike. All output neurons have the same values of parameters τ_{syn}, τ_{ref} and V_{thr} with the only difference between them being the synaptic weight w. Note that, if the weight it too large (in this case $w \geq 3$), the neuron elicits a spike in response to every input event, unless it occurs during the refractory period, effectively repeating the input sequence. This means that a group of postsynaptic neurons is redundant because they produce almost identical spike trains. In such scenario, output sequence variability could be improved by adjusting the values of τ_{syn}, τ_{ref} and V_{thr} for each neuron individually.

note, taking the binning approach to the extreme would result in a multitude of thin bins such that there is at most one event per bin for each example. We did not consider this to be worth pursuing as it would greatly increase the number of neurons needed to represent the signal in the input layer of the network.

The second point that needs to be addressed is that events occur at timescales differing by almost five orders of magnitude. At such scales the transformed events in (5) easily surpass the limits of double-precision floating-point data format. This problem still persists even after binning the sequences and shifting each sub-sequence so that it has a common starting time of $t = 0$. For bins with a higher index the range of event times, while reduced, is still relatively large for the network. And so, we can either increase the number of bins (but as discussed previously this does not seem to be a suitable approach), or transform each sequence to reduce the range of observed values. Therefore, we apply a log-transform to binned sub-sequences with events occurring in range $t \in [T_{c-1}, T_c)$ such that

$$\forall_{t\in[T_{c-1},T_c)} \;\; g(t; b_c, c) = \log_{b_c}(t - T_{c-1} + 1)\,, \quad (21)$$

for $b_c > 1$, where $c \in \{1, 2, \ldots\}$ is the index of the bin, T_c is the upper boundary of the c-th bin (i.e., the threshold), and $T_0 = 0$. Each bin corresponds to a single network input channel, hence we reuse the index c to make this explicit and avoid introducing additional notation. Note that each transformed sequence is shifted to start at $t = 0$. This transform is controlled by a single parameter b_c, the base of the logarithm. Note that $g(t; b_c, c)$ is a decreasing function of b_c.

In general, the transform in (21) allows setting a different b_c for each bin (channel). In our experimental scope we consider two strategies for selecting b_c:

1. Setting the same base $b > 1$ for all bins

$$\forall_{t\in[T_{c-1},T_c)} \;\; g_1(t; b, c) = \log_b(t - T_{c-1} + 1)\,. \quad (22)$$

2. Adjusting the base of the logarithm for each bin separately so that all bins have the same range of values after transform

$$g_2(t; \kappa, c) = \frac{\kappa}{g_1(T_c; b, c)} g_1(t; b, c)\,, \quad (23)$$

where κ is the time-instant assigned to the threshold T_c after transform, i.e., $\kappa = g_2(T_c; \kappa, c)$. Note that in this strategy the value of the parameter b does not matter as the actual logarithm base for a given bin c is controlled by the boundaries T_{c-1}, T_c and the parameter κ. For simplicity, we use the same κ for all bins.

The difference between these two approaches lies in the range of values produced by the transform. Strategy $g_1(t; b, c)$ is unbounded from above and so it might be difficult to select a single b that works at both short and long timescales. Conversely, the function $g_2(t; \kappa, c)$ squeezes all values to lie in the range $[0, \kappa]$, which might make it easier for the network to learn the relationship between events at different timescales.

Figure 3 summarizes the outlined preprocessing steps. We found that applying the b-parameterized log-transform on binned sub-sequences sufficiently addressed our concerns for training the proposed spiking neural network. Note that while the log-transform (21) has the unfortunate effect of significantly increasing the latency of the model, we are only interested in the final classification prediction of the model and not its real-time performance.

4. Results and discussion

In all of our experiments we used the same C-12-24-48-2 architecture, where C is the number of bins used to preprocess the spike train, the network has 2 output neurons, and other digits represent the number of neurons in the hidden layers. This produces a relatively small network of about $1536 + 12C$ parameters. However, a MIMO SNN model complexity is not only determined

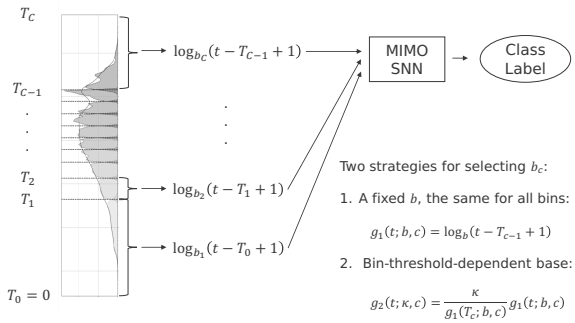


Fig. 3. Diagram of the preprocessing steps to obtain signals used to train a MIMO SNN on Twitter dataset. The empirical distribution on the left is divided into C bins.

by the number of parameters, but also by the refractory period τ_{ref} (see Eqn. (10)) which controls how often each neuron in the network is able to spike. Based on the results of preliminary study on the impact of the refractory period on the MIMO SNN signal propagation (w.r.t. training time and network spiking activity), we settle on $\tau_{\text{ref}} = 0.1$ in our further experiments.

4.1. Bot detection: Binary classifier performance. In our experiments on the classification problem, having fixed the value of τ_{ref}, we explored the impact of proposed preprocessing on model performance. We constructed a grid over preprocessing parameter space, selecting the number of bins from the set $C \in \{10, 20, 30, 40, 50\}$, whereas the log-transform parameter was set to either $b_c(\kappa)$ for $\kappa \in \{1, 2, 3\}$, or $b \in \{10, 30\}$. For every pair of parameters on this grid, several models were trained and evaluated with 5-fold cross validation. Thus, in total 125 models were trained. All models were trained for 50 epochs, each with 100 training steps over 64 class-balanced training examples. The learning rate was set to 10^{-3} in all steps. The synaptic regularization term factor γ varied substantially depending on the current training epoch – initially set to 10^5 in order to guide the model towards a state in which it is able to propagate spikes throughout all layers. After 10 epochs, we set $\gamma = 10^{-2}$ so that the model could focus on minimizing the task-specific loss term (15). This schedule for γ was determined empirically in preliminary experiments.

Given the small dataset size (366 legitimate users and 389 bots), we opt to use data augmentation in order to increase the effective training split size. Two types of augmentation were used:

- drop events with probability 0.1,
- randomly shift each event in time by t_δ uniformly distributed in $(-0.05, 0.05)$, independently with probability 0.3.

The latter augmentation type is applied only after the sequence has been preprocessed, making sure that the shift-augmented sub-sequence is still composed of events occurring at nonnegative time.

The classification accuracy achieved by all models trained over the preprocessing grid is summarized in Table 1. The results obtained during the hyperparameter search on the preprocessing grid do not suggest an existence of some pattern that holds across different number of bins or the transform choice. Notably, however, the setting denoted by the parameter $b = 30$ seems to be more robust compared to other settings in that it is the only one that allowed the model to consistently reach an accuracy over 70% regardless of the number of bins.

Overall, the wide range of performance classification scores reported by different models suggest that it is imperative to perform hyperparameter tuning when using the proposed MIMO network for this specific scenario of extremely long-range dependencies between events. Nevertheless, it must be stressed that the hyperparameters associated with the neural model itself (V_{thr}, τ_{syn}, τ_{ref}) can be selected heuristically. Firstly, note that V_{thr} only influences the scale of trained weights and has no impact on training dynamics, making the choice of V_{thr} arbitrary. On the other hand, the time constants τ_{syn} and τ_{ref} can be selected according to the input event distribution. The post-synaptic potential constant τ_{syn} needs to be longer than the relevant temporal patterns present in the input data (Bohte *et al.*, 2002). Lastly, setting $\tau_{\text{ref}} < \tau_{\text{syn}}$ prevents the scenario in which the network rarely generates events.

Table 2 compares our best model with the results obtained in the original study by Mazza *et al.* (2019) in terms of accuracy, recall, precision, F_1-score and Matthews correlation coefficient (MCC). Our MIMO SNN model outperforms all of the presented supervised approaches (Botometer, Social fingerprinting), as well as most of the unsupervised methods (HoloScope; PCA- and TICA-based RT_{BUST}). Importantly, the latter group of methods relied on fitting the model on the entire unlabeled dataset of 63,762 accounts and evaluating on the labeled portion, whereas we focus only on the annotated subset (as outlined in Section 3.1), composed of about 755 labeled cases in total. We note that the Variational Autoencoder (VAE) variant of the RT_{BUST} model performed better than our approach. As this model also leveraged the unlabeled portion of the dataset, it stands to reason that there is a clear benefit to clustering-based methods, in which the presence of a suspicious behavior emerges only when analyzing user in groups, rather then as individuals. Furthermore, we note that the examples examined by Mazza *et al.* (2019) were manually labeled and thus any incorrectly labeled cases could have had a much bigger impact on the supervised

Table 1. Classification accuracy for the MIMO SNN models trained on the Twitter tweet-retweet dataset, depending on the chosen preprocessing parameters. The star ($\star$) denotes the best result for a row, whereas the diamond ($\diamond$) denotes the best result in a column.

Stratified 5-fold cross validation		Log-transform-related parameter				
		$b = 10$	$b = 30$	$b_c(\kappa = 1)$	$b_c(\kappa = 2)$	$b_c(\kappa = 3)$
Number of bins	$C = 10$	$^{\star\diamond}73.25 \pm 3.71$	70.33 ± 3.83	68.74 ± 3.09	68.08 ± 4.20	68.61 ± 10.46
	$C = 20$	70.20 ± 3.72	$^{\star}70.60 \pm 1.65$	67.95 ± 2.70	70.33 ± 3.80	68.21 ± 5.70
	$C = 30$	67.42 ± 2.03	$^{\star\diamond}71.52 \pm 4.74$	69.40 ± 4.76	70.86 ± 1.83	69.14 ± 3.09
	$C = 40$	69.27 ± 2.67	70.60 ± 4.40	66.75 ± 4.50	69.40 ± 4.42	$^{\star\diamond}71.13 \pm 3.49$
	$C = 50$	67.68 ± 4.20	71.26 ± 6.22	$^{\star\diamond}72.19 \pm 3.82$	$^{\diamond}70.99 \pm 4.26$	68.08 ± 3.61

Table 2. Comparison of model performance on the bot detection task between different techniques.

Reference	Technique	Accuracy	Recall	Precision	$\mathbf{F_1}$-score	MCC
Mazza *et al*., 2019	Botometer	58.30	30.98	69.51	42.86	0.2051
	HoloScope	49.08	0.49	28.57	0.96	-0.0410
	Social fingerprinting	71.14	89.78	65.62	75.82	0.4536
	$\mathrm{RT}_{\mathrm{BUST}}$ (PCA)	51.54	95.12	51.11	66.49	0.0446
	$\mathrm{RT}_{\mathrm{BUST}}$ (TICA)	53.64	95.12	52.28	67.47	0.1168
	$\mathrm{RT}_{\mathrm{BUST}}$ (VAE)	87.55	81.46	93.04	86.87	0.7572
Ours	MIMO SNN	73.25 ± 3.71	69.56 ± 8.93	76.39 ± 3.34	72.46 ± 5.08	0.47 ± 0.07

Table 3. Model performance in the ablation study, given the specified scenario.

Scenario	Accuracy	Recall	Precision	$\mathbf{F_1}$-score	MCC
Baseline	73.25 ± 3.71	69.56 ± 8.93	76.39 ± 3.34	72.46 ± 5.08	0.47 ± 0.07
No data augmentation	66.89 ± 1.45	58.09 ± 10.61	73.77 ± 6.50	63.80 ± 5.60	0.36 ± 0.03
Infinite refractory period	67.28 ± 2.64	56.30 ± 4.52	74.44 ± 5.63	63.89 ± 3.06	0.36 ± 0.06
No channel-splitting	63.18 ± 4.60	69.67 ± 14.22	62.61 ± 2.74	65.42 ± 7.49	0.27 ± 0.10
No log transform	47.68 ± 1.18	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	-0.06 ± 0.07

model trained with significantly fewer examples. Lastly, it is important to note that the original study lacks technical details related to the LSTM-VAE network architecture, which prevents us from making a comparison between the two methods that is adjusted for model complexity as expressed by the number of trainable parameters.

4.2. Model ablation study. For the best-performing model, we ran an ablation study experiment in order to determine which components of the proposed approach have a significant impact on model performance. We tested four scenarios:

(i) no data augmentation,

(ii) setting $\tau_{\text{ref}} = \infty$, i.e., preventing the neurons in the network from spiking more than once in response to a given input signal,

(iii) no splitting of channels into separate bins,

(iv) no log transform.

Each one represents a single change to the experimental protocol outlined in Section 4.1. Clearly, with the exception of the first scenario, these changes have a major impact on how the signal is propagated through the network or on training dynamics.

As evidenced in Table 3, removing any of the components causes a reduction in model performance.

This shows that the proposed data augmentation scheme was effective in mitigating the problem of overfitting. Furthermore, the drop in performance for the $\tau_{\text{ref}} = \infty$ scenario suggests that some of the model's capacity to process information is tied to the ability to process it over time. Unsurprisingly, not splitting the input spike train into multiple bins makes it more difficult for the model to learn long range dependencies, which is an effect that was anticipated while designing this preprocessing step. Lastly, the sharpest drop in performance is observed when the data is passed through the network without any transform that squashes the range of values at its input. Note that the obtained average accuracy of 47.68% is quite close to the ratio of the number of legitimate users to all users in the stratified evaluation data split (48.48%). This means that the model was unable to learn anything, most likely because using raw event times as large as $2 \cdot 10^4$ minutes transformed according to (5) surpasses the limits of double-precision floating-point data format, making training impossible.

5. Conclusion

This study introduces a MIMO SNN model which extends existing time-coding single-spike time-to-first-spike SNN models by defining the rules for spike propagation throughout the layers when simulating the network on conventional hardware. Once trained, the model can be realized on existing neuromorphic devices that implement the IF neuron computation. In contrast to other works, the proposed MIMO SNN expresses the entire algorithm in terms of iteratively calculating successive spikes, which stands in contrast to other works simulating the state of the entire network over a finite time window with a fixed time step. The proposed MIMO SNN is suited towards datasets with spike trains composed of relatively infrequent events occurring at different timescales, such as the Twitter user activity dataset. Furthermore, the algorithm can be implemented in modern deep learning frameworks, making it scalable to large volumes of data.

The model was applied to a labeled subset of Twitter user activity data in order to determine whether each analyzed user is legitimate or not. Choosing a dataset with a time horizon of a 2-week-period allowed us to show that the model is able to process time series composed of events occurring at timescales differing by almost five orders of magnitude. Our best model was unable to outperform the RT_{BUST} model from the original study. However, we note that the latter is an unsupervised learning algorithm and is therefore able to infer different non-overlapping patterns of activity of distinct groups of users not present in the class-label-aggregated data. Furthermore, our model relies on about 755 labeled example compared to the 63,762 unlabeled cases used to train RT_{BUST}. As the data was manually labeled by the authors of the study, any incorrectly labeled examples could have had a much bigger impact on the supervised model trained with significantly fewer examples. We have shown that the proposed MIMO SNN operates directly in an event-domain, and so there is no need to encode the time series in any way for it to be processed by the model. In addition to the classification model feasibility study, this work showcased novel signal preprocessing steps, exemplary spike train data augmentation techniques, and the heuristic of modifying regularization scale factor during training to tackle this challenging dataset. We found that these concepts were critical at preventing overfitting and stabilizing the training procedure, as evidenced by the results of the ablation study. We hope to explore these insights in future works, particularly regarding the choice of the regularization scale factor value.

Author contributions

Mateusz Pabian: conceptualization, methodology, software, investigation, writing (original draft). Dominik Rzepka: conceptualization, methodology, writing (original draft). Mirosław Pawlak: supervision, writing (review and editing).

References

Balaanand, M., Karthikeyan, N., Karthik, S., Varatharajan, R., Manogaran, G. and Sivaparthipan, C.B. (2019). An enhanced graph-based semi-supervised learning algorithm to detect fake users on Twitter, *Journal of Supercomputing* **75**(9): 6085–6105.

Bohte, S.M., Kok, J.N. and La Poutre, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons, *Neurocomputing* **48**(1–4): 17–37.

Cao, Q., Sirivianos, M., Yang, X. and Pregueiro, T. (2012). Aiding the detection of fake accounts in large scale social online services, *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), San Jose, USA*, pp. 197–210.

Cresci, S., Di Pietro, R., Petrocchi, M., Spognardi, A. and Tesconi, M. (2017). The paradigm-shift of social spambots: Evidence, theories, and tools for the arms race, *Proceedings of the 26th International Conference on World Wide Web Companion, Perth, Australia*, pp. 963–972.

Davis, C.A., Varol, O., Ferrara, E., Flammini, A. and Menczer, F. (2016). BotOrNot: A system to evaluate social bots, *Proceedings of the 25th International Conference Companion on World Wide Web, Quebec, Canada*, pp. 273–274.

Dutta, H.S., Chetan, A., Joshi, B. and Chakraborty, T. (2018). Retweet us, we will retweet you: Spotting collusive retweeters involved in blackmarket services, *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), Barcelona, Spain*, pp. 242–249.

Eshraghian, J.K., Ward, M., Neftci, E.O., Wang, X., Lenz, G., Dwivedi, G., Bennamoun, M., Jeong, D.S. and Lu, W.D. (2023). Training spiking neural networks using lessons from deep learning, *Proceedings of the IEEE* **111**(9): 1016–1054.

Falez, P., Tirilly, P., Bilasco, I.M., Devienne, P. and Boulet, P. (2019). Unsupervised visual feature learning with spike-timing-dependent plasticity: How far are we from traditional feature learning approaches?, *Pattern Recognition* **93**: 418–429.

Fang, W., Yu, Z., Chen, Y., Huang, T., Masquelier, T. and Tian, Y. (2021). Deep residual learning in spiking neural networks, *Advances in Neural Information Processing Systems* **34**: 21056–21069.

Ferrara, E., Varol, O., Davis, C., Menczer, F. and Flammini, A. (2016). The rise of social bots, *Communications of the ACM* **59**(7): 96–104.

Huh, D. and Sejnowski, T.J. (2018). Gradient descent for spiking neural networks, *Advances in Neural Information Processing Systems* **31**: 1440–1450.

Javanshir, A., Nguyen, T.T., Mahmud, M.A.P. and Kouzani, A.Z. (2022). Advancements in algorithms and neuromorphic hardware for spiking neural networks, *Neural Computation* **34**(6): 1289–1328.

Kheradpisheh, S.R. and Masquelier, T. (2020). Temporal backpropagation for spiking neural networks with one spike per neuron, *International Journal of Neural Systems* **30**(06): 2050027.

Kim, Y., Chough, J. and Panda, P. (2022). Beyond classification: Directly training spiking neural networks for semantic segmentation, *Neuromorphic Computing and Engineering* **2**(4): 044015.

Lee, J.H., Delbruck, T. and Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation, *Frontiers in Neuroscience* **10**: 508.

Mazza, M., Cresci, S., Avvenuti, M., Quattrociocchi, W. and Tesconi, M. (2019). RTbust: Exploiting temporal patterns for botnet detection on Twitter, *Proceedings of the 10th ACM Conference on Web Science, Boston, USA*, pp. 183–192.

Midya, R., Wang, Z., Asapu, S., Joshi, S., Li, Y., Zhuo, Y., Song, W., Jiang, H., Upadhay, N., Rao, M., Lin, P., Li, C., Xia, Q. and Yang, J.J. (2019). Artificial neural network (ANN) to spiking neural network (SNN) converters based on diffusive memristors, *Advanced Electronic Materials* **5**(9): 1900060.

Minnich, A., Chavoshi, N., Koutra, D. and Mueen, A. (2017). BotWalk: Efficient adaptive exploration of Twitter bot networks, *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, Sydney, Australia*, pp. 467–474.

Mostafa, H. (2017). Supervised learning based on temporal coding in spiking neural networks, *IEEE Transactions on Neural Networks and Learning Systems* **29**(7): 3227–3235.

Mozafari, M., Ganjtabesh, M., Nowzari-Dalini, A., Thorpe, S.J. and Masquelier, T. (2019). Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks, *Pattern Recognition* **94**: 87–95.

Neftci, E.O., Mostafa, H. and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks, *IEEE Signal Processing Magazine* **36**(6): 51–63.

Nunes, J.D., Carvalho, M., Carneiro, D. and Cardoso, J.S. (2022). Spiking neural networks: A survey, *IEEE Access* **10**: 60738–60764.

Pabian, M., Rzepka, D. and Pawlak, M. (2022). Supervised training of siamese spiking neural networks with earth mover's distance, *ICASSP 22022 IEEE International Conference on Acoustics, Speech and Signal Processing, Singapore*, pp. 4233–4237.

Pan, J., Liu, Y., Liu, X. and Hu, H. (2016). Discriminating bot accounts based solely on temporal features of microblog behavior, *Physica A: Statistical Mechanics and its Applications* **450**(C): 193–204.

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

Rasmussen, D. (2019). NengoDL: Combining deep learning and neuromorphic modelling methods, *Neuroinformatics* **17**(4): 611–628.

Rodríguez-Ruiz, J., Mata-Sánchez, J.I., Monroy, R., Loyola-Gonzalez, O. and López-Cuevas, A. (2020). A one-class classification approach for bot detection on Twitter, *Computers & Security* **91**: 101715.

Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M. and Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification, *Frontiers in Neuroscience* **11**: 682.

Shrestha, S.B. and Orchard, G. (2018). SLAYER: Spike layer error reassignment in time, *Advances in Neural Information Processing Systems* **31**: 1419–1428.

Stöckl, C. and Maass, W. (2021). Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes, *Nature Machine Intelligence* **3**(3): 230–238.

Thorpe, S. and Gautrais, J. (1998). Rank order coding, *6th Annual Conference on Computational Neuroscience: Trends in Research, 1998, Big Sky, USA*, pp. 113–118.

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

Wu, Y., Deng, L., Li, G., Zhu, J. and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks, *Frontiers in Neuroscience* **12**: 331.

Wunderlich, T.C. and Pehle, C. (2021). Event-based backpropagation can compute exact gradients for spiking neural networks, *Scientific Reports* **11**(1): 12829.

Yin, B., Corradi, F. and Bohté, S.M. (2021). Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks, *Nature Machine Intelligence* **3**(10): 905–913.

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

Zhang, W. and Li, P. (2020). Temporal spike sequence learning via backpropagation for deep spiking neural networks, *Advances in Neural Information Processing Systems* **33**: 12022–12033.

Zheng, H., Wu, Y., Deng, L., Hu, Y. and Li, G. (2021). Going deeper with directly-trained larger spiking neural networks, *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 11062–11070, (virtual event).

Zhou, S., Li, X., Chen, Y., Chandrasekaran, S.T. and Sanyal, A. (2021). Temporal-coded deep spiking neural network with easy training and robust performance, *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 11143–11151.

Zhu, Y., Yu, Z., Fang, W., Xie, X., Huang, T. and Masquelier, T. (2022). Training spiking neural networks with event-driven backpropagation, *Advances in Neural Information Processing Systems* **35**: 30528–30541.

Mateusz Pabian received his MSc degree in biomedical engineering from the AGH University of Science and Technology, Kraków, Poland, in 2019, where he is currently pursuing the PhD degree at the Department of Measurement and Electronics. His research interests include machine learning and event-based systems, particularly spiking neural networks.

Dominik Rzepka holds a PhD degree in electrical engineering from the AGH University of Science and Technology, Kraków, Poland. His research work focuses on event-based and neuromorphic signal processing and machine learning. He is also involved in development of industrial applications of ML and DSP, including biomedical diagnostics, wireless communication and computer vision in industrial inspection.

Mirosław Pawlak holds a PhD and DSc degrees in computer engineering from Wroclaw University of Technology, Poland. He is currently a professor with the Department of Electrical and Computer Engineering, University of Manitoba, Canada. His research interests include statistical signal processing, machine learning, and nonparametric modeling.

Received: 29 September 2024
Revised: 22 March 2025
Accepted: 14 April 2025