Multivariate Time Series Classification Using Spiking Neural Networks

Haowen Fang, Amar Shrestha, Qinru Qiu
Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, New York
Email: {hfang02,amshrest,qiqiu}@syr.edu

***Abstract*—There is an increasing demand to process streams of temporal data in energy-limited scenarios such as embedded devices, driven by the advancement and expansion of Internet of Things (IoT) and Cyber-Physical Systems (CPS). Spiking neural network has drawn attention as it enables low power consumption by encoding and processing information as sparse spike events, which can be exploited for event-driven computation. Recent works also show SNNs' capability to process spatial temporal information. Such advantages can be exploited by power-limited devices to process real-time sensor data. However, most existing SNN training algorithms focus on vision tasks and temporal credit assignment is not addressed. Furthermore, widely adopted rate encoding ignores temporal information, hence it's not suitable for representing time series. In this work, we present an encoding scheme to convert time series into sparse spatial temporal spike patterns. A training algorithm to classify spatial temporal patterns is also proposed. Proposed approach is evaluated on multiple time series datasets in the UCR repository and achieved performance comparable to deep neural networks.**

***Index Terms*—Spiking neural network, neuromorphic computing**

I. Introduction

The function and behavior of Spiking Neural Networks (SNN) are derived from its inspiration, i.e. the brain, which is capable of performing numerous cognitive tasks with minimal energy requirements. The potential of SNN has drawn various research interests including emerging device, algorithm and applications [17], [23], [29], [30]. In general, the brain's capability comes from the complex dynamics of the networks of spiking neurons and the plastic synapses connecting them. These dynamics can capture complex spatial temporal features of input encoded as sparse temporal spiking activity. Despite the biological inspiration, majority of the existing models of SNNs are unable to replicate such dynamics to encode, learn and decode temporal information.

The limitations of existing SNNs are multifold. Firstly, most SNN models and training algorithms consider only the statistics of spike activities. A numerical value is represented by spike counts in a time window [9]. Though, this type of SNN has demonstrated state-of-art performance in various tasks [18], it suffers from high spike activities [16]. Thus, it cannot fully benefit from event-driven computation. Secondly, directly adapting backpropagation is not feasible because spiking neuron's output is a Dirac delta function. One approach to address the problem is to train an artificial neuron network (ANN) such as multi-layer perceptron (MLP) and map the weights to SNN. However, it suffers form accuracy degradation, additional fine-tuning of weights and thresholds is required to minimize the performance penalty [4]. Recently gradient surrogate is proposed to approximate the gradient of the spiking function, enabling backpropagation [6], [8], [15], [24]–[26]. [6] derived a cumulative error function as gradient surrogate. [26] derived a simplified model from Leaky Integrate and Fire neuron (LIF), and proposed four functions as gradient surrogates. Other approaches include replacing hard threshold function by a differentiable soft spike [20] [14]. However, it compromises SNN's most distinct feature, binary spike.

While most SNN models and training algorithm use spike counts to resemble numerical value, it is observed that in biological neural networks, temporal structure of spike train also conveys information [3]. Two spike trains of same spike rates can have distinct temporal patterns, hence the represented information is different. Such temporal encoding can efficiently encode information using extremely sparse spikes-events [16]. There are some existing works to train neurons to detect temporal spike patterns. Tempotron [13] trains individual neuron to perform binary classifications for different spatial temporal input spike patterns. Neuron generate at least one spike for positive pattern, and remain inactive for other patterns. Based on Tempotron, [12] proposed an algorithm to adjust synaptic weights such that neuron can generate desired number of spikes given a specific input pattern. SPAN [19] trains an individual neuron to associate a spatial temporal input pattern with a specific output spike pattern. However, these works aim at training individual neurons, cannot be extended to multiple layers, therefore the performance is limited. There are also recent works utilize backpropagation through time (BPTT) to address the temporal dependency problems. [25] proposed a training rule to reassign errors in time. [11] proposed a novel loss function and derived an iterative model from Tempotron. Based on the iterative model, network can be unrolled hence BPTT is possible. [31] captures the temporal dependency on membrane potentials and use membrane potential as objective function to learn temporal patterns.

Existing works have achieved comparable performance with DNN in vision tasks such as static image or event-based data classification, however few SNN models address the time series classification tasks. The first challenge is the limitation of rate coding, since it treats spikes statistically, spike coding cannot represent temporal information. Though it is possible to

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

flatten the time series into a 1-D array and then represent it by rate coding, this increases the input size. In addition, for real time applications, flattening input requires buffering, which increases computation latency. Secondly, unlike images, such as MNIST images, where all value's range, precision and scale are identical, multivariate time series may be collected from different sensors, therefore their precision and range may vary. Rate coding has to guarantee the precision for the most high-resolution input. Such "design for worst case" coding scheme lacks flexibility and hence is not efficient. New coding methods to represent time series are required to exploit the potential of SNNs.

In this work, our contributions are summarized as follows:

- We present a coding method that can convert time series into sparse spatial temporal spike patterns.
- We derive an iterative SNN model from Spike Response Model, such that the Backpropagation Through Time is possible. An event-based updating algorithm is also proposed to reduce computation overhead for inference.
- We formulate a backpropagation rule for the iterative SNN model and propose a training algorithm to train the model on spatial temporal patterns.
- We evaluate the proposed method on multiple datasets, and achieve comparable accuracy with DNN. To the best of our knowledge, this is the first work applying SNN for multivariate time series classification.

II. SNN Model

Without loss of generality, we adopt the a widely used Leaky Integrate and Fire (LIF) neuron defined by Spike Response Model [10], [13]. Each input spike induces a charge in the neuron's membrane potential, which is called a postsynaptic potential (PSP):

$$PSP(t) = \sum_{t_i}^{t_i<t} K(t - t_i)w \tag{1}$$

where t_i denotes the arrival time of i_{th} input spike. $K(t)$ is synapse kernel. Neuron accumulate all input PSPs, such that the membrane potential $v(t)$ is defined as:

$$v(t) = \sum_{i}^{N} w_i PSP_i(t) - V_{th} \sum_{t_s^j<t} e^{-\frac{t-t_s^j}{\tau}} \tag{2}$$

where N is the number of input synapse. t_j^i is the arrival time of j_{th} spike at i_{th} input synapse. τ is a time constant of neuron. w_i is the weight associated with each input synapse. $t_s^j < t$ is the time when the neuron generates an output spike and the rightmost term can be interpreted as a negative voltage applied to the neuron itself such that the membrane potential is decreased by a factor of the threshold voltage V_{th}. This serves as the reset mechanism at the time of spike. Thus, the neuron's potential is the summation of all weighted input PSP plus the negative voltage given by rightmost term [10]. The neuron model with N inputs is illustrated in figure 1. PSP kernel $K(t)$ is defined as:

$$K(t) = V_0(e^{-\frac{t}{\tau_m}} - e^{\frac{-t}{\tau_s}}) \tag{3}$$

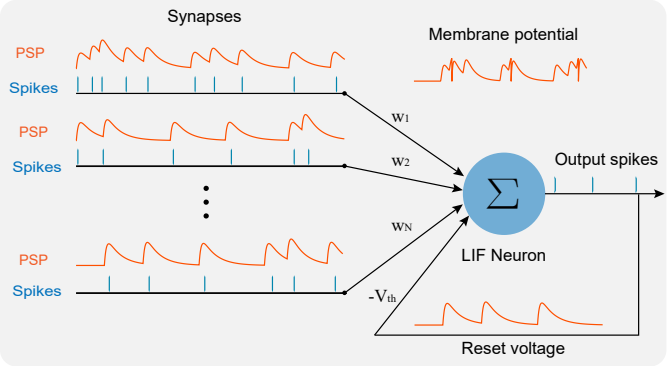


Fig. 1: Spiking neuron model

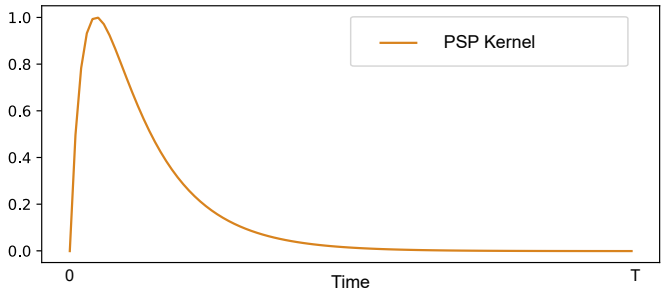


Fig. 2: PSP kernel

where τ_m and τ_s are two time constants. $V_0 = \eta^{\frac{\eta}{\eta-1}}$ is a normalization factor which scales the maximum value of $K(t)$ to 1, and $\eta = \frac{\tau_m}{\tau_s}$ [12]. The shape of the PSP kernel is shown in figure 2.

At time t, PSP and membrane potential are determined by all previous inputs. However, it is not feasible to directly implement the SNN model defined by Equation 2. At any given time t, $v(t)$ has to be computed by recursively convolving input spike trains with $K(t)$, thus, incurring significant computation overhead. To address this issue, an incremental way to update the PSP can be derived from the SRM model in discrete time domain.

More formally, input spike train $S[t]$ can be defined as a sequence of time shifted Dirac Delta functions:

$$S[t] = \sum_{n}^{t} x[n]\delta[t - n] \tag{4}$$

where $x_i[t] = 1$ denotes a spike received at time t, otherwise $x_i[t] = 0$. Similarly, output spike train $O[t]$ can defined as:

$$O[t] = \sum_{n}^{t} y[n]\delta[t - n] \tag{5}$$

where $y[t]$ satisfies $(v[t] < V_{th} \rightarrow y[t] = 0) \cap (v[t] > V_{th} \rightarrow y[t] > 0)$. To derive the incremental model, We define $M[t] = \sum_n e^{-\frac{n}{\tau_m}} S[t - n]$, $H[t] = \sum_n e^{-\frac{n}{\tau_s}} S[t - n]$, such that the PSP can be expressed as the combination of $M[t]$ and $H[t]$ [28]:

$$PSP[t] = V_0(M[t] - H[t]) \tag{6}$$

$M[t]$ and $H[t]$ can be computed incrementally:

$$M(H)[t] = e^{\frac{-1}{\tau_{m(s)}}} M(H)[t-1] + S[t] \tag{7}$$

Similarly, we can compute reset voltage $R[t]$:

$$R[t] = e^{\frac{-1}{\tau_s}} R[t-1] + O[t-1] \tag{8}$$

Such that the SNN defined in Equation 2 can be equivalently expressed as:

$$V_i^l[t] = I_i^l[t] - V_{th} R_i^l[t] \tag{9a}$$

$$I_i^l[t] = V_0 \sum_j^{M_{l-1}} w_{i,j}^l (M_i^l[t] - H_i^l[t]) \tag{9b}$$

$$M_j^l[t] = \alpha N_j^l[t-1] + O_j^{l-1}[t] \tag{9c}$$

$$H_j^l[t] = \beta H_j^l[t-1] + O_j^{l-1}[t] \tag{9d}$$

$$R_i^l[t] = \gamma R_i^l[t-1] + O_i^l[t-1] \tag{9e}$$

$$O_i^L[t] = U(V_i^l[t] - V_{th}) \tag{9f}$$

where indexes l, i, j denote layer index, neuron index and input index respectively. N_l denotes the number of neurons in l_{th} layer. $I_i^l[t]$ is input current, $R[t]$ is reset voltage, and $O_i^l[t]$ is neuron output. $\alpha = e^{\frac{-1}{\tau_m}}$, $\beta = e^{\frac{-1}{\tau_s}}$, $\gamma = e^{\frac{-1}{\tau}}$ are three decay factors. More specifically, $l = 0$ denotes the encoding layer, which will be discussed in section III, L is the number of layers in the network and $l = L$ denotes output layer. $U(x)$ is a Heaviside step function:

$$U(x) = 0, \text{if } x < 0, \text{ otherwise } 1 \tag{10}$$

In the above model, the temporal dependency can be clearly seen in Equation 9a - 9f. At each time t, the PSP, membrane potential and output can be computed based on time $t-1$, hence by unfolding the network, Backpropagation Through Time (BPTT) can be used to train the network. Note that the gradient of $U(x)$ is a Dirac Delta function, therefore backpropagation cannot be directly applied. Its approximation will be discussed in IV.

A. Event-driven Inference

The above model provides an explicit approach to update SNN's states based on step-wise computation, and is suitable for training. In inference, the model can be simulated in an event-driven manner, i.e. computation is only necessary when there is a spike event, hence significantly reducing the computation overhead.

Suppose at time t, the value of $M[t]$ or $H[t]$ is known, without input spike, after Δt unit time later, i.e. at time $t' = t + \Delta t$, the $M[t]$ and $H[t]$ can be computed as:

$$\begin{aligned} M(H)[t'] &= \sum_{t_i}^{t_i<t} e^{-\frac{t+\Delta t-t_i}{\tau_{m(s)}}} \\ &= \sum_{t_i<t'} e^{-\frac{t-t_i}{\tau_{m(s)}}} \cdot e^{-\frac{\Delta t}{\tau_{m(s)}}} \\ &= M(H)[t] e^{\frac{-\Delta t}{\tau_{m(s)}}} \end{aligned} \tag{11}$$

Algorithm 1: Event-driven inference

```
Input spike buffer: Q_spike ← ∅
Elapsed time since last input spike: D_in[:] ← 0
Elapsed time since last output spike: D_out[:] ← 0
Time t ← 0
M[:] ← 0
H[:] ← 0
R[:] ← 0
for t < T do
    if Q_spike ≠ ∅ then
        foreach synapse j do
            V ← 0
            if j in Q_syn then
                M[j] ← M[j] · α^{D_in[j]} + 1
                H[j] ← H[j] · β^{D_in[j]} + 1
                D_in[j] ← 0
                V ← V + V_0 · w_j · (M[j] − H[j])
            else
                D_in[i] ← D_in[i] + 1
        if V > V_th then
            D_out ← 0
            R ← R · γ^{D_out[j]} + V_th
            V ← V − V_th
        else
            D_out ← D_out + 1
    else
        D_in ← D_in + 1
        D_out ← D_out + 1
    t ← t + 1
```

When there is an input spike at time $t' = t + \Delta t$. There is an instantaneous unit charge on $M[t']$ and $H[t']$:

$$M(H)(t') = M(H) e^{\frac{-\Delta t}{\tau_{m(s)}}} + 1 \tag{12}$$

Similar update rule applies for $R[t]$:

$$R(t + \Delta t) = R[t] e^{\frac{-\Delta t}{\tau_m}} \tag{13}$$

The event-driven computation algorithm is shown in Algorithm 1. By tracking the elapsed time Δt, computation is only necessary when there is an input or output spike. In addition, the kernel decays over time, and becomes effectively 0 after a period. Therefore the decay factor for different Δt can be pre-computed and stored in a look-up table, so that the expensive exponential function is avoided.

III. Spatial Temporal Population Encoding

Rate coding represents a numerical value by the activity of an individual neuron that fires at a particular rate. For example, in vision tasks, to encode one pixel, a spike train's spike count C in a given time window T is proportional to the pixel value. There are several drawbacks of rate coding. 1) the precision is limited because the value represented by rate coding is quantized by bin size $1/T$. Though higher precision can be obtained by increasing T, the computational latency increases as well; 2) it is unable to represent temporal

information as it treats spike activity statistically. Time series have to be flattened and then converted to spike trains. In real time scenarios, it requires data stream to be buffered, which causes additional latency; 3) Individual neuron is too noisy due to stochastic nature, thus it introduces additional noise; 4) it causes high spiking activity, as larger value has to be represented by more spikes, which deprives the SNN energy efficiency. 5) It is incapable of representing negative values, which are common in sensor inputs.

To address the above issues, we employ a coding method suitable for encoding time series by combining population coding and temporal coding [7]. In population coding the information is represented by the activity of a group of neurons. Inside a population, each neuron has its favorable input, i.e. each neuron responds to a particular input and remains relatively inactive for other inputs. In temporal coding, the spike train patterns also convey information.

We utilize a population of Current-based Integrate and Fire(CUBA) neurons as encoder. A CUBA neuron is defined as a hybrid system [2]:

$$\begin{aligned} \frac{dV}{dt} &= -\frac{1}{\tau}V + g \cdot I_{ext}(t) \\ V &\leftarrow 0 \qquad \text{if } V > V_{th} \end{aligned} \tag{14}$$

where $I_{ext}(t)$ is the external time-varying input current, τ is the membrane time constant, which determines the decay speed of membrane potential, $V(t)$ is the neuron membrane potential, g is the gain. Neuron accumulates the input current and updates the membrane potential continuously. When the $V(t)$ exceeds V_{th}, a reset is triggered, membrane potential is forced to 0.

In practice, CUBA neuron model is simulated in discrete time, $V(t)$ is evaluated on a time grid and the interval of the grid is dt [21]. $I_{ext}(t)$ is also sampled at each time grid. Such that the discrete version of the model represented by Equation 14 is:

$$\begin{aligned} V[t+1] &= e^{-\frac{dt}{\tau}}V[t] + g \cdot I_{ext}[t] \\ V[t+1] &= 0 \qquad \text{if } V[t] > V_{th} \end{aligned} \tag{15}$$

With this coding scheme, a univariate time series, for example sensor data is treated as input current and connected to a population of E encoder neurons. Each neuron may have different time constant τ and gain g, so that each encoder responds to the input differently. In addition, by setting g to negative, neuron can also respond to negative input, which overcomes the drawback of rate coding. We utilize Neural Engineering Framework (NEF) [5] to pre-train the encoder. Using this approach, time varying signal is converted into time varying spike patterns. For multivariate time series of C channels, each channel can be encoded using the above approach. Unlike vision tasks in which all input dimensions have identical resolution and range, multivariate time series maybe collected from different sensors, therefore the resolution, precision and range may vary. By coding method, the population size and tuning curve can be adjusted to provide just enough precision for all the channels [7], [27].

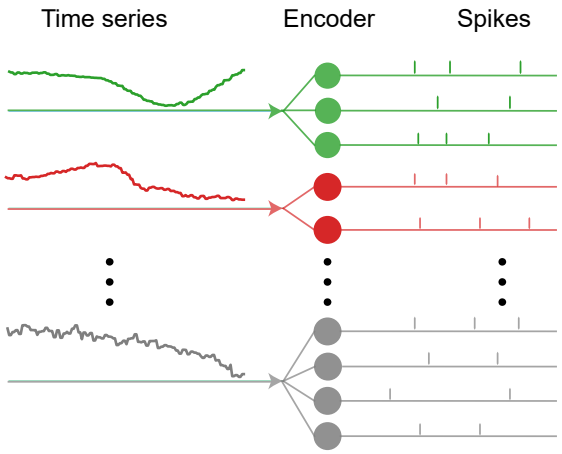


Fig. 3: Coding method

IV. Training Algorithm

For the classification task, the neuron in the output layer that fires most frequently represents the result, we use cross-entropy loss defined as:

$$E = -\sum_{i}^{N_L} y_i log(p_i) \tag{16}$$

where p_i is the probability of each class calculated by softmax, p_i given by:

$$p_i = \frac{\exp\left(\sum_t^T O_i^L[t]\right)}{\sum_{j=1}^{N_L} \exp\left(\sum_t^T O_j^L[t]\right)} \tag{17}$$

where y_i is the label, L is number of layers, $O_i^L[t]$ denotes output of last layer, and N_L is the neuron number of last layer.

Equation 9a - 9f provide an explicit way to update SNN states and outputs. By unfolding the network, BPTT can be used for training. First, we define $\delta_i^l[t] = \frac{\partial L}{\partial O_i^l[t]}$, $\epsilon_i^l[t] = \frac{\partial U(V_i^l[t]-V_{th})}{\partial V_i^l[t]}$ and $\kappa_i^l[t] = \frac{\partial O_i^l[t+1])}{\partial O_i^l[t]}$.

$\kappa_i^l[t]$ is given by:

$$\kappa_i^l[t] = -V_{th}\gamma\epsilon_i^l[t] \tag{18}$$

At last layer L, $\delta_i^L[t]$ can be computed as:

$$\begin{aligned} \delta_i^L[t] &= \frac{\partial E}{\partial O_i^L[k]} = \frac{\partial E}{\partial(\sum_{k=1}^T O_i^L[t])}\frac{\partial(\sum_{k=1}^T O_i^L[t])}{\partial O_i^L[t]} \\ &= (p_i - y_i)(\sum_{k=t}^{T}\frac{\partial O_i^L[k]}{\partial O_i^L[t]}) \end{aligned} \tag{19}$$

$\frac{\partial O_i^L[k]}{\partial O_i^L[t]}$ is computed as:

$$\frac{\partial O_i^L[k]}{\partial O_i^L[t]} = \prod_{n=0}^{k-t-1}\frac{\partial O_i^L[t+n+1]}{\partial O_i^L[t+n]} = \prod_{n=0}^{k-t-1}(-V_{th}\gamma\epsilon_i^L[t+n]) \tag{20}$$

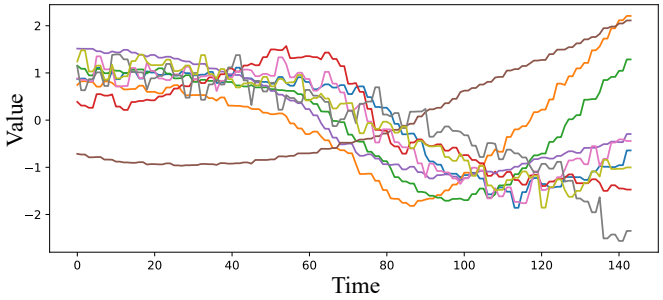


Fig. 4: Articulary Word Recognition dataset sample

For hidden layer $l < L$, $\delta_i^l[t]$ can be computed recursively from output layer L and time T to input layer and time 0:

$$\begin{aligned}\delta^{l,i}[t] &= \sum_j^{N_{l+1}} \frac{\partial E}{\partial O_j^{l+1}[t+1]} \frac{\partial O_j^{l+1}[t]}{\partial O_i^l[t]} + \frac{\partial E}{\partial O_i^l[t]} \frac{\partial O_i^l[t+1]}{\partial O_i^l[t]} \\ &= -V_{th}\delta_i^l[t+1]\epsilon_i^l[t+1] \\ &+ \sum_j^{N_{l+1}} w_j^i \delta_i^{l+1}[t+1]\epsilon_i^{l+1}[t+1](\alpha - \beta) \end{aligned} \tag{21}$$

Heaviside step function $U(x)$ is non-differentiable. We employ gradient surrogate [20] to address this issue. In forward path, the spike generation mechanism remains unchanged, while in the backward path, the derivative of $U(x)$ is replaced by the derivative of a smooth function. We use a sigmoid function proposed by [26] as the gradient surrogate in the backward path, such that the gradient of $U(x)$ is approximated as:

$$\frac{\partial U(v)}{\partial V} \approx \frac{e^{V_{th}-v}}{(1+e^{V_{th}-v})^2} \tag{22}$$

Based on above equations, the gradient of weight can be computed as:

$$\begin{aligned}\frac{\partial E}{\partial \mathbf{w}^l} &= \sum_{t=1}^{T} \frac{\partial E}{\partial \boldsymbol{O}^l[t]} \frac{\partial \boldsymbol{O}^l}{\partial \boldsymbol{V}^l[t]} \frac{\partial \boldsymbol{V}^l[t]}{\partial \boldsymbol{I}^l[t]} \frac{\partial \boldsymbol{I}^l[t]}{\partial \mathbf{w}^l} \\ &= \sum_{t=1}^{T} V_0 \cdot \boldsymbol{\delta}^l[t]\boldsymbol{\epsilon}^l[t](\boldsymbol{M}^l[t] - \boldsymbol{N}^l[t]) \end{aligned} \tag{23}$$

V. Experiments

The proposed network model and algorithm are implemented in PyTorch. We demonstrate the effectiveness of our work in two experiments. In the first experiment, we compared the coding efficiency of rate coding and temporal population coding in terms of spike rate and input size. In second experiments, the proposed network and algorithm is evaluated on various multivariate time series classification tasks.

A. Coding Efficiency

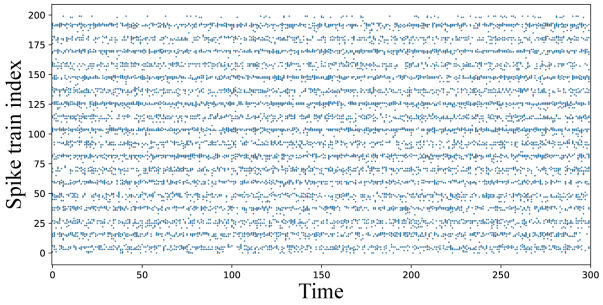
First, we study the efficiency of the proposed coding method by utilizing the Articulary Word Recognition dataset collected by UEA & UCR Time Series repository [1] as an example. This dataset consists of multivariate sensor data recorded by Electromagnetic Articulograph (EMA), which is a device to track the motion of speakers' tongue and lips. Each sample contains 9 variates of length 144. An example of this dataset is shown in figure 4, each line represents a time varying input. We use both rate coding and temporal population coding to convert the time series to spikes. The spike patterns are shown in figure 5. Each dot in figure 5a represents a spike, and in figure 5b a spike is represented by a vertical line. We use an input window of 300. Due to the incapability of rate coding to represent temporal information, the time series have to be flattened, resulting in 1296 spike trains. For clarity, only the first 100 spike trains are shown in figure. In temporal coding, for each variate, we use 5 neurons to encode. It is clearly seen that the temporal population coding is sparser. In addition, it is encoding the input with 45 spike trains, which is significantly smaller than the number of spike trains obtained by rate coding. This is beneficial to reduce the SNN model size.

We tested our coding method with rate coding on four multivariate datasets, details including average spike count, spike rate, input size are shown in table I. Temporal in table I refers to temporal population coding. The spike rates are significantly lower than rate-based coding. As can be seen in last column, the input size of our coding method is also significant less. Particularly for long time series, such as Atrial Fibrillation dataset. It consists of two variates, and the length is 640, therefore flattening operation resulting a large number of

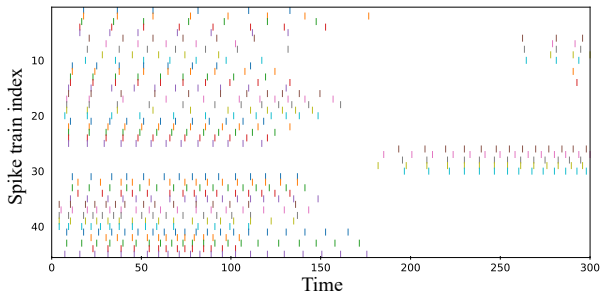
Algorithm 2: Training process of one iteration

Input: Time-varying input $\boldsymbol{I}_{ext}[t]$
Output: Optimized weights $\boldsymbol{W}^l$
`// Forward`
for $t = 1$ *to* T **do**
  `// Encoding`
  $\boldsymbol{V}^0 \leftarrow \boldsymbol{e}^{\frac{-1}{\tau}} \cdot \boldsymbol{V}^0 + \boldsymbol{I}_{ext}[t]$
  if $\boldsymbol{V}^0 > V_{th}$ **then**
    $\boldsymbol{V}^0 \leftarrow 0$ `// Encoding`
    $\boldsymbol{O}^0 \leftarrow 1$ `// Generate spike`
  else
    $\boldsymbol{O}^0 \leftarrow 0$
  for $l = 1$ *to* L **do**
    `// Update states Eq.9a -9e`
    $(\boldsymbol{M}^l, \boldsymbol{H}^l, \boldsymbol{R}^l, \boldsymbol{V}^l, \boldsymbol{I}^l) \leftarrow$
      Update$(\boldsymbol{M}^l, \boldsymbol{H}^l, \boldsymbol{R}^l, \boldsymbol{O}^{l-1}, \boldsymbol{O}^l)$
      $\boldsymbol{O}^l \leftarrow$ SpikeFunction$(\boldsymbol{V}^l)$ `// Eq.9f`
  `// Calculate loss`
  $E =$ Loss$(\boldsymbol{O}^L[1], ..., \boldsymbol{O}^L[T])$ `// Eq.16-17`
  `// Backward`
for $t = T$ *to* 1 **do**
  $(\boldsymbol{\delta}^L[t-1], \boldsymbol{\kappa}^L[t-1]) \leftarrow$ BackProp$(\boldsymbol{E}, \boldsymbol{\delta}^L[t], \boldsymbol{\kappa}^L[t])$
    `// Eq.19-20`
  for $l = L$ *to* 1 **do**
    $(\boldsymbol{\delta}^{l-1}[t-1], \boldsymbol{\kappa}^{l-1}[t-1]) \leftarrow$
      BackProp$(\boldsymbol{\delta}^l[t], \boldsymbol{\kappa}^l[t])$ `// Eq.18,21`

inputs. Buffering such long time series also causes significant latency in real time applications. While temporal population coding can convert input on the fly, not only input size is reduced, buffering is also no longer required.



(a) Rate coding



(b) Temporal population coding

Fig. 5: Coding comparison

TABLE I: Coding Efficiency

Dataset	**Coding Method**	**Spike Count**	**Spike Rate**	**Input Size**
Articulary Word Recognition	Rate	65653.8	16.90	1296
	Temporal	518.7	3.84	45
Basic Motions	Rate	15061.7	8.36	600
	Temporal	122.4	1.13	30
Finger Movements	Rate	1069.0	25.45	1400
	Temporal	471.3	1.12	140
Atrial Fibrillation	Rate	23041.4	10.77	1280
	Temporal	164.5	4.76	10

B. Computation Overhead

To evaluate the computation overhead of proposed coding method and event driven inference algorithm, we build a SNN to classify Articulary Word Recognition dataset. A vanilla 2 layer stacked LSTM and RNN of unit size 300 are also built as reference. The network structure, number of network parameters, and accuracy are shown in table II. Our network achieved comparable accuracy with 11 % number of parameters of LSTM. In addition, the length of this time series is 144, LSTM and RNN have to perform computation step by step, this introduces significant amount of operations. Our model can benefit from the event driven nature, computations are only necessary when there are spike events. Given the average number of input spike is 518.7, the computation overhead is minimal compared with LSTM/RNN.

TABLE II: Model comparison

Model	**Network structure**	**Parameter number**	**Accuracy**
LSTM	9-300-300-25	1103125	98.31
RNN	9-300-300-25	281425	98.20
SNN	45-300-300-25	125880	98.27

C. Time Series Classification

TABLE III: Accuracy

Dataset	ED [1]	DTW [1]	TapNet [32]	WEASEL [22]	This work
Articulary Word Recognition	0.97	0.98	0.987	-	0.98
FaceDetection	0.519	0.513	0.556	-	0.57
BasicMotions	0.675	1	1	-	1
Heartbeat	0.62	0.659	0.751	-	0.72
Spoken Arabic Digits	0.967	0.96	0.983	0.992	0.98
JapaneseVowels	0.924	0.959	0.965	0.976	0.97
RacketSports	0.868	0.842	0.868	0.934	0.87

Our algorithm is evaluated on 7 multivariate time series datasets. We build a network of size 500-500-500-X, X indicates the size of last layer, which varies according to different dataset class numbers. Adam optimizer is used and the learning rate is 0.0001. The result is shown in table III. ED and DTW refer to 1-Nearest Neighbor with Euclidean Distance and Dynamic Time Warping respectively [1]. TapNet is a DNN-based approach for time series classification [32]. No accuracy in SNN domain is listed, to our best knowledge, there is no previous work that comprehensively focusing on time series classification with SNN.

In all the 7 datasets, our method outperforms the 1-Nearest Neighbor classifier, which is a standard classifier for time series classification. In Spoken Arabic Digits dataset and Racket Sport dataset, our method achieves higher accuracy than DNN based approach. In Articulary Word Recognition dataset, Heart Beat dataset, thought TapNet achieves better accuracy, however the advantage is insignificant: 0.987 v.s. 0.98, 0.751 v.s. 0.72.

VI. Conclusion

In this work, we presented an iterative SNN model and training algorithm for spatial temporal spike pattern classification. A coding method to convert continuous time series to discrete spikes is also proposed. Our coding method is able to represent information by sparse spike patterns, such that the computation overhead can be significantly reduced. We evaluate our algorithm and coding method on various multivariate time series dataset, and outperform the standard 1-Nearest Neighbor classifiers and also show competitive performance with DNN based approaches.

References

[1] A. Bagnall, H. A. Dau, J. Lines, M. Flynn, J. Large, A. Bostrom, P. Southam, and E. Keogh. The uea multivariate time series classification archive, 2018. *arXiv preprint arXiv:1811.00075*, 2018.

[2] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. M. Bower, M. Diesmann, A. Morrison, P. H. Goodman, F. C. Harris, et al. Simulation of networks of spiking neurons: a review of tools and strategies. *Journal of computational neuroscience*, 23(3):349–398, 2007.

[3] D. A. Butts, C. Weng, J. Jin, C.-I. Yeh, N. A. Lesica, J.-M. Alonso, and G. B. Stanley. Temporal precision in the neural code and the timescales of natural vision. *Nature*, 449(7158):92, 2007.

[4] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2015.

[5] C. Eliasmith and C. H. Anderson. *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT press, 2004.

[6] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha. Backpropagation for energy-efficient neuromorphic computing. In *Advances in Neural Information Processing Systems*, pages 1117–1125, 2015.

[7] H. Fang, A. Shrestha, Z. Zhao, Y. Li, and Q. Qiu. An event-driven neuromorphic system with biologically plausible temporal dynamics. In *38th IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2019*, page 8942083. Institute of Electrical and Electronics Engineers Inc., 2019.

[8] H. Fang, A. Shrestha, Z. Zhao, and Q. Qiu. Exploiting neuron and synapse filter dynamics in spatial temporal learning of deep spiking neural network. *arXiv preprint arXiv:2003.02944*, 2020.

[9] H. Fang, A. Shrestha, Z. Zhao, Y. Wang, and Q. Qiu. A general framework to map neural networks onto neuromorphic processor. In *20th International Symposium on Quality Electronic Design (ISQED)*, pages 20–25. IEEE, 2019.

[10] W. Gerstner and W. M. Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[11] P. Gu, R. Xiao, G. Pan, and H. Tang. Stca: spatio-temporal credit assignment with delayed feedback in deep spiking neural networks. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 1366–1372. AAAI Press, 2019.

[12] R. Gütig. Spiking neurons can discover predictive features by aggregate-label learning. *Science*, 351(6277):aab4113, 2016.

[13] R. Gütig and H. Sompolinsky. The tempotron: a neuron that learns spike timing–based decisions. *Nature neuroscience*, 9(3):420, 2006.

[14] D. Huh and T. J. Sejnowski. Gradient descent for spiking neural networks. In *Advances in Neural Information Processing Systems*, pages 1433–1443, 2018.

[15] J. H. Lee, T. Delbruck, and M. Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in neuroscience*, 10:508, 2016.

[16] T. Liu, Z. Liu, F. Lin, Y. Jin, G. Quan, and W. Wen. Mt-spike: A multilayer time-based spiking neuromorphic architecture with temporal error backpropagation. In *Proceedings of the 36th International Conference on Computer-Aided Design*, pages 450–457. IEEE Press, 2017.

[17] X. Ma, G. Yuan, S. Lin, C. Ding, F. Yu, T. Liu, W. Wen, X. Chen, and Y. Wang. Tiny but accurate: A pruned, quantized and optimized memristor crossbar framework for ultra efficient dnn implementation. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 301–306, 2020.

[18] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[19] A. Mohemmed, S. Schliebs, S. Matsuda, and N. Kasabov. Span: Spike pattern association neuron for learning spatio-temporal spike patterns. *International journal of neural systems*, 22(04):1250012, 2012.

[20] E. O. Neftci, H. Mostafa, and F. Zenke. Surrogate gradient learning in spiking neural networks. *arXiv preprint arXiv:1901.09948*, 2019.

[21] S. Rotter and M. Diesmann. Exact digital simulation of time-invariant linear systems with applications to neuronal modeling. *Biological cybernetics*, 81(5-6):381–402, 1999.

[22] P. Schäfer and U. Leser. Multivariate time series classification with weasel+ muse. *arXiv preprint arXiv:1711.11343*, 2017.

[23] A. Shrestha, K. Ahmed, Y. Wang, D. P. Widemann, A. T. Moody, B. C. Van Essen, and Q. Qiu. A spike-based long short-term memory on a neurosynaptic processor. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 631–637. IEEE, 2017.

[24] A. Shrestha, H. Fang, Q. Wu, and Q. Qiu. Approximating back-propagation for a biologically plausible local learning rule in spiking neural networks. In *Proceedings of the International Conference on Neuromorphic Systems*, page 10. ACM, 2019.

[25] S. B. Shrestha and G. Orchard. Slayer: Spike layer error reassignment in time. In *Advances in Neural Information Processing Systems*, pages 1412–1421, 2018.

[26] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in neuroscience*, 12, 2018.

[27] S. Yarrow and P. Seriès. The influence of population size, noise strength and behavioral task on best-encoded stimulus for neurons with unimodal or monotonic tuning curves. *Frontiers in computational neuroscience*, 9:18, 2015.

[28] Q. Yu, H. Li, and K. C. Tan. Spike timing or rate? neurons learn to make decisions for both through threshold-driven plasticity. *IEEE transactions on cybernetics*, 49(6):2178–2189, 2018.

[29] G. Yuan, C. Ding, R. Cai, X. Ma, Z. Zhao, A. Ren, B. Yuan, and Y. Wang. Memristor crossbar-based ultra-efficient next-generation baseband processors. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1121–1124. IEEE, 2017.

[30] G. Yuan, X. Ma, C. Ding, S. Lin, T. Zhang, Z. S. Jalali, Y. Zhao, L. Jiang, S. Soundarajan, and Y. Wang. An ultra-efficient memristor-based dnn framework with structured weight pruning and quantization using admm. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2019.

[31] F. Zenke and S. Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation*, 30(6):1514–1541, 2018.

[32] X. Zhang, Y. Gao, J. Lin, and C.-T. Lu. Tapnet: Multivariate time series classification with attentional prototypical network. 2020.