IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 20, 2020, accepted April 22, 2020, date of publication May 14, 2020, date of current version June 5, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2994360

Efficient Spiking Neural Networks With Logarithmic Temporal Coding

MING ZHANG[1], ZONGHUA GU[2], (Senior Member, IEEE), NENGGAN ZHENG[3], DE MA[1,4], AND GANG PAN[1,4]

[1]College of Computer Science, Zhejiang University, Hangzhou 310027, China
[2]Department of Applied Physics and Electronics, Umeå University, 901 87 Umeå, Sweden
[3]Qiushi Academy for Advanced Studies, Zhejiang University, Hangzhou 310027, China
[4]Zhejiang Lab, Hangzhou 311121, China

Corresponding author: De Ma (made@zju.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFB1002503, in part by the Key Research and Development Program of Zhejiang Province under Grant 2020C03004, in part by NSFC under Grant U1909202, Grant 61672454, Grant 61572433, and Grant 61972347, in part by the Zhejiang Provincial Natural Science Foundation under Grant LR19F020005, and in part by the Zhejiang Lab under Grant 2019KC0AD02.

ABSTRACT A Spiking Neural Network (SNN) can be trained indirectly by first training an Artificial Neural Network (ANN) with the conventional backpropagation algorithm, then converting it into an equivalent SNN. To reduce the computational cost of the resulting SNN as measured by the number of spikes, we present Logarithmic Temporal Coding (LTC), where the number of spikes used to encode an activation grows logarithmically with the activation value; and the accompanying Exponentiate-and-Fire (EF) neuron model, which only involves efficient bit-shift and addition operations. Moreover, we improve the training process of ANN to compensate for approximation errors due to LTC. Experimental results indicate that the resulting SNN achieves competitive performance in terms of classification accuracy at significantly lower computational cost than related work.

INDEX TERMS Spiking neural networks, temporal coding, neuromorphic computing.

I. INTRODUCTION

Deep learning based on multi-layer *Artificial Neural Networks (ANNs)*, especially Convolutional Neural Networks (CNNs), have achieved tremendous success in many application domains in recent years. A typical ANN as used in Deep Learning have many stacked layers, including an input layer that reads in the input, e.g., an image, multiple hidden layers and an output layer that outputs the result, e.g., the classification result for the input image. Despite their successes, the substantial computational cost of training and running DNNs are significant due to the large number of neurons and edges in a DNN, which lead to high costs for both computation and memory accesses. In this paper, we focus on the *inference* stage, consisting of a single forward pass through the DNN to produce a classification result, instead of the *training* stage, consisting of many forward and backward passes through the DNN to update the edge weight parameters with backpropogration. The inference stage is more important for embedded/edge devices, since inference is performed on resource and power-constrained embedded devices, whereas training can be performed in the cloud with powerful server machines.

In an ANN, each layer's output is calculated by first computing the dot product of its input vector and weight matrix of its input edges, and then applying a non-linearity, e.g., *Sigmoid* or *Rectified Linear Unit (ReLU)*. The computational cost of one forward inference pass is a constant value determined by the neural network topology, i.e., the number of layers, number of neurons in each layer, and the connectivity patterns between layers. In contrast, *Spiking Neural Networks* (SNNs) [1] rely on *spikes* for event-driven computation and communication, inspired by biological neurons, and the computational cost of an SNN is proportional to the number of spikes. An SNN neuron is only active when it has incoming spikes. Thanks to its event-driven execution model, hardware implementation of an SNN can potentially achieve significantly higher computational efficiency and lower power consumption compared to its equivalent ANN. Neuromorphic hardware accelerators for SNNs, e.g., the Darwin Neural

The associate editor coordinating the review of this manuscript and approving it for publication was Nizam Uddin Ahamed.

Processing Unit [2], [3], can be much more power-efficient compared to hardware accelerators for ANNs, and has the potential to contribute to the long-term goal of integrating artificial intelligence and biological intelligence to form *cyborg intelligence* [4], [5].

In this paper, we present an efficient ANN-to-SNN conversion algorithm. The resulting SNN has significantly reduced number of spikes for each forward inference, yet achieves comparable classification performance as related work. Our main contributions are as follows:

1) We propose *Logarithmic Temporal Coding (LTC)*, where an activation value is approximately encoded with a binary string with a predefined length. The number of spikes needed to encode an activation value grows logarithmically, rather than linearly, with the activation value.
2) We propose the *Exponentiate-and-Fire (EF) neuron model* used in conjunction with LTC, which performs equivalent computation to that of an ANN neuron with Rectified Linear Unit (ReLU) nonlinearity, the most commonly-used neuron model in Deep Learning.
3) We present enhanced ANN training algorithm to reduce the performance gap between ANN and the converted SNN.

II. RELATED WORK

A. THE LEAKY INTEGRATE-AND-FIRE (LIF) MODEL

Most literature on SNNs adopt the *Leaky Integrate-and-Fire* (LIF) neuron model. Fig. 1 [1] shows the dynamics of membrane potentials of an LIF neuron with multiple input spikes. Every time the neuron receives an input spike, a current is injected to cause an instantaneously increase in its membrane potential V_m, called the Excitatory Post-Synaptic Potential (EPSP). If the time interval between two consecutive input spikes is long, e.g., the time interval from 1 ms to 23 ms, the membrane potential decreases gradually due to the leakage current. Without subsequent input spikes, the membrane potential would eventually decrease to the equilibrium potential E_l. (Note that the numbers shown in the figures are not biologically-realistic, since the equilibrium potential of biological neurons is typically -70 mV instead of 0 mV.) If enough input spikes arrive within a short time interval, e.g., the three spikes arriving during the time interval from 44 ms to 50 ms, the membrane potential rises rapidly. When it is high enough to reach the firing threshold V_{th}, the neuron fires an output spike. Immediately after firing an output spike, the membrane potential is reset to the equilibrium potential E_l, and the neuron enters a *refractory period*, during which the neuron is not responsive to any input spikes. The LIF model captures the essential property of SNN that the response of a neuron is closely related to the temporal structure of the input. A neuron acts like a coincidence detector: Enough spikes arriving at the neuron within a short time interval can elicit an output spike from the neuron. However, an equal number of spikes scattered across a long time interval may not be able to trigger any spike from the neuron due to the leakage current.

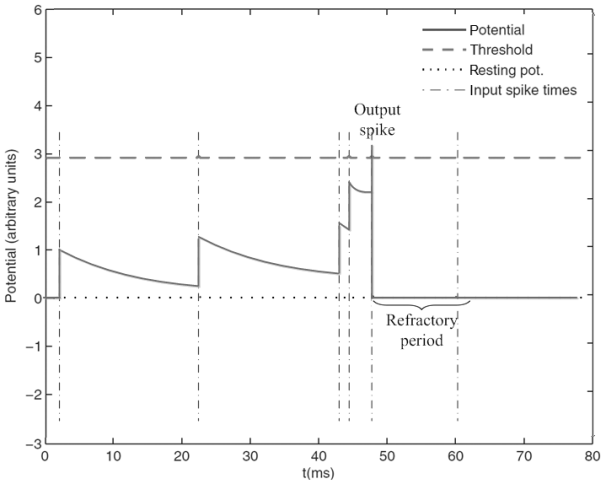


FIGURE 1. Dynamics of the membrane potential of a Leaky Integrate-and-Fire (LIF) neuron with input spikes.

In contrast to most related work, we present a novel Exponentiate-and-Fire (EF) model instead of the more commonly-used LIF model. The EF model is not as biologically-realistic as LIF, but it is used in conjunction with LTC to achieve efficient encoding of activation values in an SNN.

There are two approaches to SNN training: direct training of an SNN, or training of an ANN followed by ANN-to-SNN conversion into an equivalent SNN. We review them briefly in the following subsections.

B. DIRECT TRAINING OF SNNS

Direct training of SNNs include the following approaches:

1) Unsupervised learning based on Hebb's rule, stated informally as: "Neurons that fire together, wire together." More formally, an increase in synaptic strength arises from the pre-synaptic neuron's repeated and persistent stimulation of the post-synaptic neuron. *Spike Timing Dependent Plasticity (STDP)* is a Hebbian learning rule for SNNs: for two connected neurons, if the pre-synaptic neuron A always fires within a small time window before the post-synaptic neuron B fires, which means firing of B is correlated with firing of A, then the strength of the synapse between A and B is increased; on the contrary, if the pre-synaptic neuron A always fires within a small time window after the post-synaptic neuron B fires, the strength of the synapse between them is decreased. As one recent work on this topic, Liu *et al.* [6] presented unsupervised training of SNN based on STDP for object recognition from Address Event Representation (AER) input generated by an Event Camera.
2) Supervised learning. Examples include: SpikeProp [7], Tempotron [8], Remote Supervised Method

(ReSuMe) [9], Chronotron [10], Spike Pattern Association Neuron (SPAN) [11], Precise-Spike-Driven Synaptic Plasticity (PSD) [12], etc. Supervised learning algorithms are not biologically-realistic, in contrast to STDP.

In general, SNN training algorithms are less mature than training algorithms for ANNs based on back-propagation for deep learning, and the performance, as measured by classification accuracy, of the resulting SNN is typically worse than an ANN with the same network topology.

C. ANN-TO-SNN CONVERSION

A promising alternative is to train an ANN with the conventional backpropagation algorithm and then convert it into an equivalent SNN, thus avoiding the difficulty of direct training of an SNN. In [13], an ANN with Rectified Linear Unit (ReLU) nonlinearity was trained using backpropagation, and the weights were then directly mapped to an SNN of Integrate-and-Fire (IF) neurons with the same topology. In a similar way, an ANN with Softplus [14] or Noisy Softplus [15] nonlinearity could be converted to an SNN of more biologically plausible Leaky Integrate-and-Fire (LIF) neurons. The performance gap between the resulting SNN and the original ANN can be narrowed by weight normalization [16] and resetting membrane potential by subtracting the firing threshold [17]. Zambrano *et al.* [18] proposed Adaptive Spiking Neuron (ASN) based on synchronous Pulsed Sigma-Delta coding. When driven by a strong stimulus, an ASN adaptively raises its dynamic firing threshold every time it fires a spike, thus reducing its firing rate.

Most of the current ANN-to-SNN conversion methods are based on *rate coding*, where an activation value in the ANN is approximated by the firing rates of a Poisson spike train in the SNN, and the number of spikes for encoding an activation value grow linearly with the activation value. This implies that the SNN needs to fire a large number of spikes in order to achieve comparable performance as its corresponding ANN, which implies high computational cost. In contrast to rate-coding, our work is based on *temporal coding* [19], which relies on the precise timing of the spikes to encode information instead of average firing rates. Temporal coding has a biological basis, and it helps to implement pattern recognition based on temporally coded spike trains, which is vital to biological sensory systems. Thanks to temporal coding, the human vision system can achieve very fast processing (in millisecond scale) of the stimulus image.

Some authors [20], [21] proposed logarithmic-scaled quantization for efficient encoding of edge weight parameters to reduce the memory size requirements of an ANN. In this paper, we apply logarithmic-scaled quantization to neuron *activation*, the output value of each neuron, instead of edge weights, in order to achieve efficient ANN-to-SNN conversion.

III. LOGARITHMIC TEMPORAL CODING (LTC) AND EXPONENTIATE-AND-FIRE (EF) NEURON MODEL

A. INTUITIVE DESCRIPTION OF LTC AND EF NEURON MODEL

To encode an activation value into a spike train, the activation value is encoded as a binary number with a predefined number of bits. For multi-spike LTC, one spike is generated for each 1 bit in the binary number, and no spike is generated for the 0 bits; for single-spike LTC, only one spike is generated for the most significant 1 bit.

Before delving into the formal descriptions, we use a small example to illustrate the basic idea. Suppose the edge in the ANN connecting a neuron N_i to one of its downstream neurons N_j has weight W_{ij}. During execution, neuron N_i emits an output value of 5 to its downstream neurons, and the *pre-activation* of neuron N_j due to the output of N_i should be $5 \times W_{ij}$. Now we would like to convert the ANN into an (approximately) equivalent SNN.

- With rate coding: every spike causes an equal amount of increase to neuron N_j's membrane potential. For example, 5 spikes may be used to encode value 5, and each spike causes an equal increase of W_{ij}, with total increase of $5 \times W_{ij}$ to neuron N_j's membrane potential.
- With *multi-spike LTC*: the decimal value of 5 is 101 in binary, so we use 2 spikes emitted at time steps t and $t + 2$ to encode the decimal value 5 (t may be 0 or some other value, depending on the pre-defined exponent range). The first spike encodes the higher-order bit of 1, and the second spike encodes the lower-order bit of 1. A spike's contribution to receiver neuron N_j's membrane potential increases exponentially with time (multiplies by 2 at every time step), as defined by the EF neuron model. Therefore, at the last time step, a spike emitted earlier will cause exponentially larger increase in N_j's membrane potential. At time $t+2$, the first spike will cause an increase of $2^2 \times W_{ij}$, and the second spike will cause an increase of $2^0 \times W_{ij}$, with total increase of $(2^2 + 2^0) \times W_{ij} = 5 \times W_{ij}$ to neuron N_j's membrane potential.
- With *single-spike LTC*: we use one spike emitted at time step t to encode the decimal value 4, as an approximation to value 5. At time $t+2$, this spike will cause an increase of $2^2 \times W_{ij} = 4 \times W_{ij}$ to neuron N_j's membrane potential.

This example shows that the number of spikes is reduced from 5 for rate coding, to 2 and 1 respectively, for multi-spike LTC and for single-spike LTC. Furthermore, single-spike LTC incurs a smaller number of spikes than multi-spike LTC at the cost of reduced precision. We present a more complete toy example in Section III-D.

We emphasize computational efficiency over biological realism in the design of LTC and EF neuron model, hence our model may not be biologically realistic, e.g., the membrane potential dynamics of a biological neuron do not conform to LTC and EF, and we do not model the refractory period during which the neuron is not responsive to input spikes.

Next, we present more formal descriptions of LTC and the EF neuron model.

B. LOGARITHMIC TEMPORAL CODING (LTC)

We approximate a non-negative real-valued activation a by the sum of a finite series of powers of two $\tilde{a} = \sum_{e=e_{min}}^{e_{max}} b_e \cdot 2^e$, where the sum runs over the elements of a predefined *exponent range* $[e_{min}, e_{max}]$, i.e., a set of consecutive integers from e_{min} to e_{max}, and b_e is the bit at position e of the binary number of a. We refer to this approximation as *Multi-spike Logarithmic Approximation (Multi-spike LA)*. The value of $\tilde{a}$ is:

$$\tilde{a} = \begin{cases} 0 & \text{if } a < 2^{e_{min}}, \\ \lfloor a/2^{e_{min}} \rfloor \cdot 2^{e_{min}} & \text{if } 2^{e_{min}} \le a < 2^{e_{max}+1}, \\ 2^{e_{max}+1} - 2^{e_{min}} & \text{if } a \ge 2^{e_{max}+1}. \end{cases} \tag{1}$$

where $\lfloor \cdot \rfloor$ denotes the floor function.

We can obtain the *Single-spike Logarithmic Approximation (Single-spike LA)* by only keeping the Most Significant Bit (MSB):

$$\tilde{a} = \begin{cases} 0 & \text{if } a < 2^{e_{min}}, \\ 2^{\lfloor \log_2 a \rfloor} & \text{if } 2^{e_{min}} \le a < 2^{e_{max}+1}, \\ 2^{e_{max}} & \text{if } a \ge 2^{e_{max}+1}. \end{cases} \tag{2}$$

We refer to multi-spike LA and single-spike LA collectively as *Logarithmic Approximation (LA)*.

Correspondingly, we can obtain two variants of LTC: *Multi-spike LTC* and *Single-spike LTC*. For multi-spike LTC, an LTC spike train is generated from the logarithmic approximation $\tilde{a}$ of the activation value a in the time window $[0, T-1]$, where $T = e_{max} - e_{min} + 1$ is the time window size in terms of the number of discrete time steps in the time window. If a power of two 2^e term is present in the series of powers of two of $\tilde{a}$, i.e., $e \in [e_{min}, e_{max}]$ and $b_e = 1$, then a spike is generated at time step $t = e_{max} - e$. We derive an upper bound of the number of spikes used to encode an activation value in Proposition 1. Similarly, single-spike LTC encodes an activation value into an LTC spike train consisting of at most one single spike.

Proposition 1: Suppose multi-spike LTC encodes a real value a into a spike train with n_s spikes. If $a < 2^{e_{min}}$, then $n_s = 0$; if $2^{e_{min}} \le a < 2^{e_{max}+1}$, then $n_s \le \lfloor \log_2 a \rfloor - e_{min} + 1$; if $a \ge 2^{e_{max}+1}$, then $n_s = e_{max} - e_{min} + 1$.

We prove Proposition 1 in Section A of the supplementary material.

The exponent ranges are hyperparameters that need to be determined/tuned by the designer, to perform trade-offs between computation accuracy and computational cost as measured by the number of spikes. In a feed-forward SNN, the input exponent range of one layer is the same as the output exponent range of its preceding layer. To minimize the number of hyperparameters, we use the same output exponent range for all hidden layers.

C. EXPONENTIATE-AND-FIRE (EF) NEURON MODEL

Fig. 2 illustrates the EF neuron model, in the form of the Spike Response Model (SRM) [22]. An EF neuron integrates input LTC spike trains using an exponentially growing postsynaptic potential (PSP) kernel, and generates output spikes using an exponentially growing afterhyperpolarizing potential (AHP) kernel. With the exponentially growing kernels, an EF neuron is able to perform computation that is equivalent to the computation of an ANN neuron with ReLU nonlinearity.

Assuming discrete time, we now describe the EF neuron model. If the neuron fires an output spike at the time step $t \in \mathbb{Z}$ and then resets its membrane potential instantly, its membrane potential may have two values at the time step. To distinguish between the two values, we define the *pre-reset membrane potential* $V_m^-(t)$ at time step t, i.e., the membrane potential immediately before the reset, as:

$$V_m^-(t) = \sum_{i \in \Gamma} w_i \cdot h_i(t) + \sum_{t^{out} \in \mathcal{F}^{out}} \eta(t - t^{out}) \cdot \mathbb{1}(t > t^{out}) \tag{3}$$

where Γ is the set of synapses; w_i is the weight of the synapse i; h_i is the total PSP elicited by the input spike train at the synapse i; $\mathcal{F}^{out} \subseteq \mathbb{Z}$ is the set of output spike times of the neuron; $\eta(t - t^{out})$ is the AHP elicited by the output spike at time t^{out}; $\mathbb{1}(\cdot)$ evaluates to one if and only if the condition enclosed within the round brackets is true. The *post-reset membrane potential* $V_m(t)$ immediately after the reset is given by:

$$V_m(t) = V_m^-(t) + \eta(0) \cdot \mathbb{1}(t \in \mathcal{F}^{out}) \tag{4}$$

If the neuron does not fire any output spike at time step t, then $V_m^-(t) = V_m(t)$.

1) INPUT INTEGRATION

Input spike trains of an EF neuron are generated according to LTC using the input exponent range $[e_{min}^{in}, e_{max}^{in}]$, and presented to the neuron during its input time window $[0, T^{in}-1]$, where $T^{in} = e_{max}^{in} - e_{min}^{in} + 1$ is the input time window size. The exponentially growing PSP kernel $\epsilon(t - t^{in})$ is used to integrate input spikes:

$$\epsilon(t - t^{in}) = 2^{e_{min}^{in}} \cdot 2^{t - t^{in}} \cdot \mathbb{1}(t \ge t^{in}) \tag{5}$$

where $t \in \mathbb{Z}$ is the current time; $t^{in} \in \mathbb{Z}$ is time of the input spike. With this PSP kernel, the PSP elicited by an input spike is equal to $2^{e_{min}^{in}}$ at the input spike time $t = t^{in}$. The EF neuron's membrane potential grows exponentially, i.e., is doubled every time step, until it reaches the firing threshold V_{th} to trigger an output spike.

The total PSP elicited by an input spike train at the synapse i is the superposition of PSPs elicited by all spikes in the spike train:

$$h_i(t) = \sum_{t^{in} \in \mathcal{F}_i^{in}} \epsilon(t - t^{in}) \tag{6}$$

where $\mathcal{F}_i^{in}$ is the set of spike times of the input spike train.

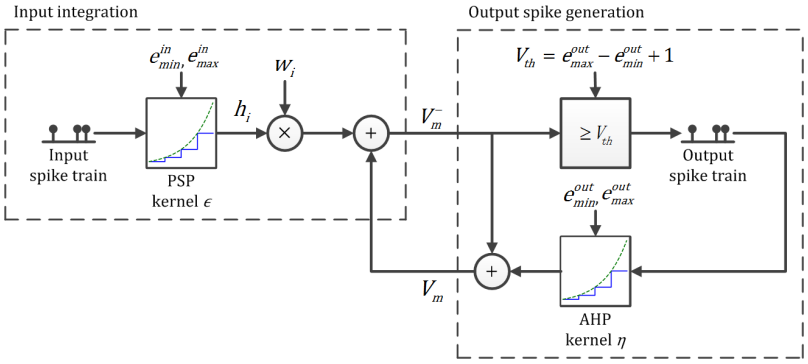


FIGURE 2. Computation graph of an Exponentiate-and-Fire (EF) neuron.

If the EF neuron does not fire any output spike before $t = T^{in} - 1$, then no output spike would interfere with input integration, and the EF neuron computes a weighted sum of its input spike trains, as Lemma 1 states.

Lemma 1: The pre-reset membrane potential of an EF neuron $V^-_m(T^{in} - 1) = \sum_{i\in\Gamma} w_i \cdot \tilde{a}_i$, *if the EF neuron does not fire any output spike during the time interval* $[0, T^{in} - 2]$, *where* $\tilde{a}_i$ *is the LA of the i-th input LTC spike train.*

We prove Lemma 1 in Section B of the supplementary material.

The EF neuron requires input spike trains to be processed in a time-triggered fashion, which makes it ideally suitable for digital hardware implementation.

2) OUTPUT SPIKE GENERATION

Suppose an EF neuron with $[e^{out}_{min}, e^{out}_{max}]$ being its output exponent range fires an output spike at time step t^{out}, when its membrane potential reaches its firing threshold $V_{th} = 2^{e^{out}_{max}}$. With multi-spike LTC, the AHP kernel $\eta(t - t^{out})$ of an *multi-spike EF neuron* is defined as:

$$\eta(t - t^{out}) = -V_{th} \cdot 2^{t - t^{out}} \tag{7}$$

Immediately after firing the output spike at time t^{out}, the membrane potential is reset by subtracting the firing threshold from it (by substituting Eqn. (7) into Eqn. (4) with $t = t^{out}$):

$$V_m(t) = V^-_m(t) - V_{th} \tag{8}$$

With single-spike LTC, the AHP kernel $\eta(t - t^{out})$ of an *single-spike EF neuron* is defined as:

$$\eta(t - t^{out}) = -V^-_m(t^{out}) \cdot 2^{t - t^{out}} \tag{9}$$

Immediately after the output spike is fired at time t^{out}, the membrane potential is reset to zero (by substituting Eqn. (9) into Eqn. (4) with $t = t^{out}$):

$$V_m(t) = 0 \tag{10}$$

After firing an output spike, with multi-spike LTC, the neuron may fire subsequent output spikes, since its membrane potential may still be positive, and the membrane potential is always doubled at every time step; with single-spike LTC, no more subsequent output spikes will be fired, since its membrane potential has been reset to 0, and 0 doubled is still 0.

As a result of the firing rules defined above, an EF neuron is able to generate an output LTC spike train that encodes $\max(V^-_m(T^{in} - 1), 0)$ using its output exponent range $[e^{out}_{min}, e^{out}_{max}]$, and emit the spike train within its output time window $[T^{in} - 1, T^{in} + T^{out} - 2]$, which starts at the last time step $T^{in} - 1$ of the input time window, and lasts for $T^{out} = e^{out}_{max} - e^{out}_{min} + 1$ time steps.

If the EF neuron receives all input spikes within its input time window, then no input spike would interfere with output spike generation during its output time window, and the EF neuron generates the desired output LTC spike train within its output time window, as Lemma 2 states.

Lemma 2: An EF neuron generates an output LTC spike train that encodes $\max(V^-_m(T^{in} - 1), 0)$ *using its output exponent range and presents the spike train within its output time window, if the EF neuron does not receive any input spike after the end of its input time window.*

We prove Lemma 2 in Section C of the supplementary material.

Theorem 1: An EF neuron performs equivalent computation to the computation of an ANN neuron with ReLU nonlinearity, and encodes the result into its output LTC spike train, if the following conditions hold:

1) *All input spikes are received within its input time window, and*
2) *No output spikes are fired before the beginning of its output time window.*

Proof: Theorem 1 follows from Lemmas 1 and 2. □

However, with the spike generation mechanism alone, an EF neuron may fire undesired output spikes outside its output time window. An undesired early output spike before the output time window interferes with input integration of the neuron. In addition, the output time window of a layer l of EF neurons is the input time window of the next layer $l + 1$. An undesired late output spike after the output

time window interferes with output spike generation of the downstream neurons. Undesired output spikes break the equivalence between EF neurons and ANN neurons with ReLU nonlinearity, which in turn degrades the performance of the SNN.

In order to prevent undesired output spikes of an EF neuron from affecting the downstream neurons, we allow output spikes within the output time window to travel to the downstream neurons, and discard undesired output spikes outside the output time window. Furthermore, we reduce the chance for an EF neuron to fire an undesired early output spike by suppressing excessively large activations of the corresponding ANN neuron, as detailed in Section III-E.

Algorithm 1 shows operations an EF neuron performs at every time step. First, the membrane potential V_m is doubled (Eqns. (5), (7) and (9)). Then, the input current I is calculated by summing up weights w_i of the synapses that receive an input spike at the current time step (Eqn. (3)). The input current is scaled by the resistance $2^{e^{in}_{min}}$ (Eqn. (5)) and the result is added to the membrane potential (Eqn. (3)). If the membrane potential is greater than or equal to the firing threshold V_{th}, an output spike is fired, and the membrane potential is reset accordingly (Eqns. (7) and (9)).

From Algorithm 1, it can be seen that the EF neuron model can be efficiently implemented with digital circuit. If V_m is implemented as a fixed-point number, it can be doubled by a bit shift; if V_m is implemented as a floating-point number, it can be doubled by an addition to its exponent component, which is an integer. The multiplication by $2^{e^{in}_{min}}$ can be avoided by pre-computing $w^i \cdot 2^{e^{in}_{min}}$ for every synaptic weight w_i and using the scaled synaptic weights at run-time. In summary, an EF neuron performs:

1) one integer/floating-point addition for every input spike,
2) one bit shift/integer addition to double its membrane potential at every time step,
3) one comparison to determine whether it should fire an output spike at every time step, and
4) one integer/floating-point subtraction or assignment to reset its membrane potential for every output spike.

D. A TOY EXAMPLE OF LTC AND THE EF NEURON MODEL

Fig. 3 illustrates the operations of an EF neuron according to Algorithm 1. The neuron's input and output exponent ranges are chosen as $[e^{in}_{min}, e^{in}_{max}] = [0, 3]$ and $[e^{out}_{min}, e^{out}_{max}] = [1, 4]$, respectively. Given these exponent ranges, its firing threshold $V_{th} = 2^{e^{out}_{max}} = 16$; its input and output time windows are $[0, 3]$ and $[3, 6]$, respectively. Let's assume multi-spike LTC. Input activation values are encoded into LTC spike trains: $a^{in}_1 = 8$ (1000 in binary) is encoded with a single spike at time step 0; and $a^{in}_2 = 5$ (0101 in binary) is encoded with 2 spikes at time steps 1 and 3. (In general the encoding of an activation value with spikes is approximate due to limited exponent ranges, but it happens to be exact in this case.) Suppose the initial membrane potential $V_m = 0$. The neuron

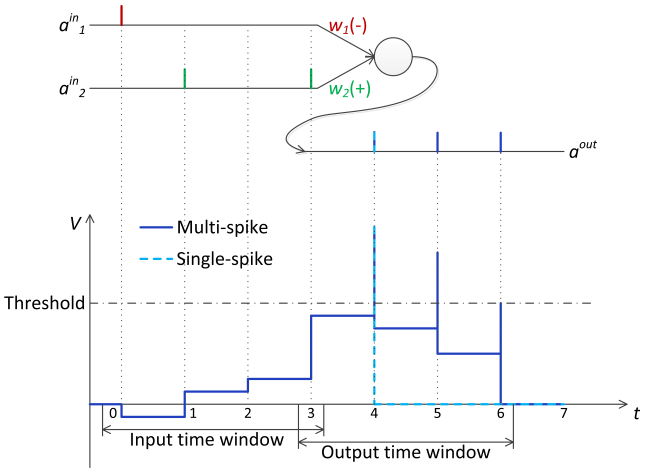


FIGURE 3. A toy example to illustrate the operation of Logarithmic Temporal Coding (LTC) and an Exponentiate-Fire (EF) neuron. Input activation values $a^{in}_1 = 8$, $a^{in}_2 = 5$; input edge weights $w_1 = -2$, $w_2 = 6$.

Algorithm 1 Operations Performed by an EF Neuron at Every Time Step

```
Input integration:
    V_m := V_m × 2
    I := Σ_{i:synapse i receives a spike} w_i
    V_m := V_m + I × 2^{e^in_min}
Output spike generation:
    if V_m ≥ V_th then
        Fire an output spike
        if the neuron is a multi-spike EF neuron then
            V_m := V_m − V_th
        end if
        if the neuron is a single-spike EF neuron then
            V_m := 0
        end if
    end if
```

operates as follows according to Algorithm 1 (The membrane potential V_m is always doubled at every time step, hence we omit this statement from the following description):

- $t = 0$: The input time window starts. Upon receiving the input spike from the higher edge, V_m is updated as $V_m := V_m \times 2 + w_1 \times 2^{e^{in}_{min}} = 0 + (-2) \times 1 = -2$.
- $t = 1$: Upon receiving the first input spike from the lower edge, V_m is updated as $V_m := V_m \times 2 + w_2 \times 2^{e^{in}_{min}} = -2 \times 2 + 6 \times 1 = 2$.
- $t = 2$: No input spike. V_m is updated as $V_m := V_m \times 2 = 4$.
- $t = 3$: The input time window ends and the output time window starts. Upon receiving the second input spike from the lower edge, V_m is updated as $V_m := V_m \times 2 + w_2 \times 2^{e^{in}_{min}} = 4 \times 2 + 6 \times 1 = 14$.
- $t = 4$: V_m is updated as $V_m := V_m \times 2 = 28 \geq V_{th} = 16$. Since 28 exceeds the firing threshold $V_{th} = 16$, the first

output spike is emitted, and V_m is reset: $V_m := V_m - V_{th} = 28 - 16 = 12$.
- $t = 5$: V_m is updated as $V_m := V_m \times 2 = 24 \geq V_{th} = 16$. Since 24 exceeds the firing threshold $V_{th} = 16$, the second output spike is emitted, and V_m is reset again: $V_m := V_m - V_{th} = 24 - 16 = 8$.
- $t = 6$: V_m is updated as $V_m := V_m \times 2 = 16 \geq V_{th} = 16$. The third output spike is emitted, and V_m is reset again to $V_m - V_{th} = 16 - 16 = 0$. The output time window ends, and no more spikes will be emitted.

We can see that the EF neuron performs equivalent computation as an ANN neuron with ReLU activation.

- The input activation value on the higher edge $a_1^{in} = 8 = 2^3$ is encoded with a single spike at time step 0, corresponding to binary number 1000.
- The input activation value on the lower edge $a_2^{in} = 5 = 2^2 + 2^0$ is encoded with 2 spikes at time steps 1 and 3, corresponding to binary number 0101.
- The output activation value after ReLU operation $a^{out} = \max(0, w_1\, a_1^{in} + w_2\, a_2^{in}) = \max(0, (-2) \times 8 + 6 \times 5) = 14$ is encoded with 3 spikes at time steps 4, 5 and 6 (the first 3 time steps in the output time window), corresponding to binary number 1110.

If single-spike LTC is adopted for the output activation, then V_m is reset to 0 after emitting the first output spike at time step 4 (shown by the dashed line), which encodes the value 8, corresponding to binary number 1000. Hence single-spike LTC incurs a smaller number of spikes at the expense of increased approximation error.

E. ENHANCED ANN TRAINING ALGORITHM

If we take a trained ANN and convert it directly to SNN with LTC and EF neuron model described earlier, then there may be a relatively large performance gap between the ANN and the corresponding SNN, due to both LTC approximation errors and undesired early output spikes. This performance gap is especially large for single-spike LTC due to the large approximation errors for each activation value. To reduce the performance gap, we introduce the *Logarithmic Approximation (LA)* into the ANN before converting it to SNN. For each ANN neuron, we apply LA to its non-negative outputs, so that the downstream neurons receive the approximate activation value instead of the original accurate activation value. The variant of LA corresponds to the variant of LTC (multi- or single-spike) used to generate the corresponding spike train in the SNN. Negative pre-activations of the output layer are not approximated using LA and remain unchanged.

Eqns. (1) and (2) show that the derivative of the LA $\tilde{a}$ w.r.t. the activation value a is zero almost everywhere, which prevents backpropagation from updating parameters of the bottom layers of the ANN. To allow the gradients to pass through LA, for both variants of LA, we define the derivative of $\tilde{a}$ w.r.t. a as

$$\frac{\mathrm{d}\tilde{a}}{\mathrm{d}a} = \begin{cases} 1 & \text{if } a < 2^{e_{max}+1}, \\ 0 & \text{if } a \geq 2^{e_{max}+1}. \end{cases} \tag{11}$$

In addition, we add a regularization term, called the *Excess Loss* L_{excess}, to the normal loss function L of the ANN, which is to be minimized by the training process.

$$L = L(x; \theta) + \lambda L_{excess} \tag{12}$$

where $L(x; \theta)$ is the loss function on training data x given parameters θ; $\lambda > 0$ is a hyperparameter that controls the strength of the excess loss. L_{excess} is defined as:

$$L_{excess} = \frac{1}{2} \sum_m \sum_l \sum_j (\max(a_{m,j}^{(l)} - (2^{e_{max}^{out,(l)}+1} - 2^{e_{min}^{out,(l)}}), 0))^2 \tag{13}$$

where the outer sum runs across training examples m, the middle sum runs across all layers l of the ANN, the inner sum runs across all neurons j of the layer l, and $a_{m,j}^{(l)}$ is the activation of the j-th neuron of the layer l for the m-th training example. The excess loss punishes large positive activations of every layer l that are greater than $2^{e_{max}^{out,(l)}+1} - 2^{e_{min}^{out,(l)}}$. The regularization term helps to suppress excessively large activations beyond the scope defined by the exponent range, and also reduces the chance of undesired early output spikes. With L_{excess}, the performance of an SNN with LTC is very close to the performance of the corresponding ANN with LA.

IV. PERFORMANCE EVALUATION

We conduct our experiments on a PC with an NVIDIA GeForce GTX 1060 GPU. We use the TensorFlow framework for training and testing ANNs for deep learning. We also implement an SNN simulator with TensorFlow. For each SNN, we build a computation graph with operations performed by the SNN at every time step, in which each spiking neuron outputs either one or zero to indicate whether it fires an output spike or not.

We use the classic MNIST dataset of handwritten digits [23], which consists of 70000 28x28-pixel greyscale images of handwritten digits, divided into a training set of 60000 images and a test set of 10000 images. For hyperparameter tuning, we divide the original training set into a training set of 55000 images and a validation set of 5000 images. The test set is only used to test ANNs and SNNs after all hyperparameters are fixed. The pixel intensities are linearly normalized to the range [0, 1] to produce input values of ANNs. The normalized pixel intensities are further encoded into input spike trains of SNNs according to multi-spike LTC.

We use two CNN architectures in our experiments:

- The *CNN-small* architecture (12C5@28x28-P2-64C5 @12x12-P2-F10) [16], which consists of the following layers: Convolutional Layer #1: Applies 12 5x5 filters (extracting 5x5-pixel subregions) to input size of 28x28; Pooling Layer #1: Performs average pooling with a 2x2 filter and stride of 2 (which specifies that pooled regions do not overlap); Convolutional Layer #2: Applies 64 5x5 filters to input size of 12x12; Pooling Layer #2: Performs average pooling with a 2x2 filter and

stride of 2; Dense Layer #1 (Logits Layer): 10 neurons, one for each digit target class (0–9). There is a total of 13.77k neurons and 29.74k edge weight parameters.

- The *CNN-large* architecture (32C5@28x28-P2-64C5@14x14-P2-F1024-F10) from the online TensorFlow tutorial [24], which consists of the following layers: Convolutional Layer #1: Applies 32 5x5 filters (extracting 5x5-pixel subregions) to input size of 28x28; Pooling Layer #1: Performs max pooling with a 2x2 filter and stride of 2; Convolutional Layer #2: Applies 64 5x5 filters to input size of 14x14; Pooling Layer #2: Performs max pooling with a 2x2 filter and stride of 2; Dense Layer #1: 1,024 neurons; Dense Layer #2 (Logits Layer): 10 neurons, one for each digit target class (0–9). There is a total of 48.07k neurons and 3.27M edge weight parameters. Since it is difficult to implement max pooling for multi-spike LTC, we replace max pooling with average pooling in the CNN-large architecture for all SNN and CNN types (detailed below).

We consider five SNN types:

1) *SNN-rate-IF-rst-zero*: SNNs based on rate coding and Integrate-and-Fire (IF) neurons that use the reset-to-zero mechanism [16]. These SNNs are converted from CNNs of the *CNN-ReLU* type, with ReLU nonlinearity and 0 biases.
2) *SNN-rate-IF-rst-subtract*: Same as SNN-rate-IF-rst-zero, except that the neurons use the reset-by-subtraction mechanism [17].
3) *SNN-ASN*: SNNs based on adaptive spike-time coding and Adaptive Spiking Neurons (ASNs) [18]. These SNNs are converted from CNNs of the *CNN-TF* type, which is obtained by replacing the ReLU nonlinearity in CNN-ReLU with the rectified half sigmoid-like transfer function in [18].
4) *SNN-multi-spike-LTC*: SNNs based on multi-spike LTC and EF neurons.
5) *SNN-single-spike-LTC*: Same as SNN-multi-spike-LTC, except that the EF neurons in the hidden layers generate single-spike LTC spike trains.

We refer to SNN-rate-IF-rst-zero and SNN-rate-IF-rst-subtract collectively as *SNN-rate-IF*, and SNN-multi-spike-LTC and SNN-single-spike-LTC collectively as *SNN-LTC*.

We have a total of 10 SNN models by instantiating each SNN type with either CNN-small or CNN-large architecture. In addition, for each of the 10 combinations, we obtain 5 different concrete SNNs, by converting from ANNs trained separately with the same hyperparameter setting (such as learning rate for all SNN types, as well as those in Table 1 for SNN-LTC), but with different random initial weights. Therefore, we consider a total of 50 concrete SNNs for performance evaluation.

For SNN-rate-IF, the maximum input rate for generating an Poisson spike train from the input image is one spike per time step, since this maximum input rate was shown to achieve the best performance [16]. For SNN-ASN (converted from CNN-TF), we adopt the hyperparameter settings for the transfer function and ASN in [18]. The resting threshold θ_0 and the multiplicative parameter m_f are set to a large value 0.1 to decrease neuron firing rates. Table 1 shows hyperparameter settings used during the ANN training process for SNN-LTC, including output exponent ranges and strength of excess loss λ as described in Section III-E. Recall that the output exponent range for each layer of the SNN is also the input exponent range for the next layer. For example, for the input layer, the pixel values range from 0 to 255, hence we set the exponent range to be $[-7, 0]$, and the input layer's output spike train spans 8 time steps.

TABLE 1. Output exponent ranges and strength of excess loss λ for ANNs of SNN-LTC types.

CNN architecture	Output exponent ranges			λ
	Input layer	Hidden layers	Output layer	
CNN-small	$[-7, 0]$	$[-3, 0]$	$[-3, 4]$	0.1
CNN-large		$[-7, -4]$		0.01

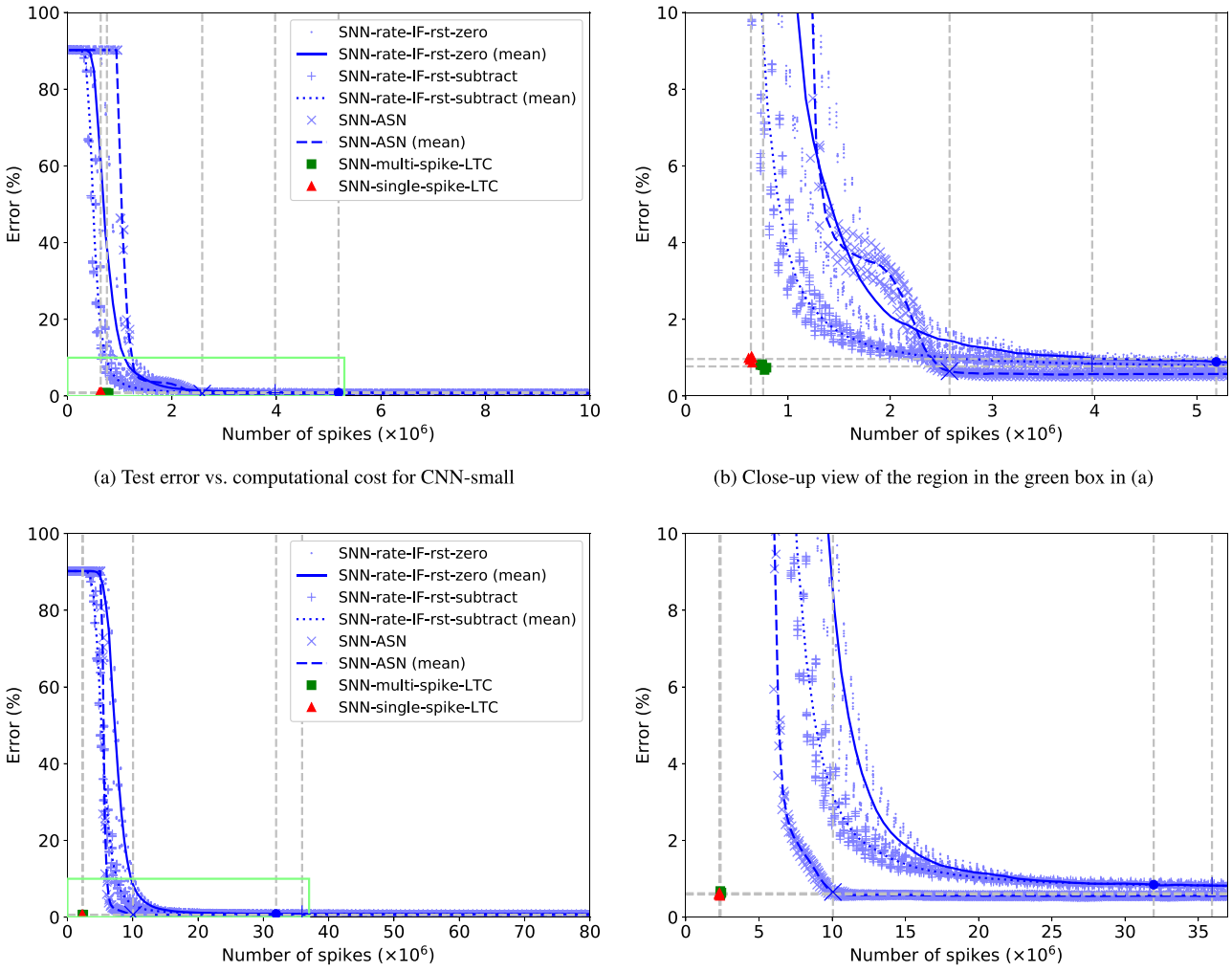
In this section, we compare performance and computational cost of our SNN-LTC types with related work [16]–[18]. We use classification error rates to measure the performance, and the number of spikes during each forward inference pass for an input image to measure the computational cost.

In the ANN, the computational cost of one forward inference pass is a constant. For rate-coding SNN types, including SNN-rate-IF-rst-zero, SNN-rate-IF-rst-subtract, and SNN-ASN, the image is presented to the input layer for a certain simulation duration, and the network outputs a classification result at every time step. The classification error rate of the SNN decreases gradually with increasing simulation time and number of spikes. The time it takes for the output to converge depends on the specific input and the SNN, and is difficult to determine *a priori*. The *stable* computational cost of a rate-coding SNN type is defined as the computational cost at the earliest time point when the test error of the rate-coding SNN type converges to a stable value. More precisely, we consider the test error of the rate-coding SNN type to have converged if it remains within the $\pm 0.1\%$ range around the final test error. In contrast, in our work (SNN-LTC), the SNN outputs a classification result at the end of a deterministic, fixed-length delay, which is determined by the exponent range sizes of the layers. There is no gradual test error reduction process as for the rate-coding SNNs. Hence Table 2 compares the *final* test errors and the *final* computational costs for the SNN-LTC types with the *stable* test errors and the *stable* computational costs for the rate-coding SNN types.

Fig. 4 shows how the test errors decrease with increasing computational costs for different SNN types. For each rate-coding SNN type, each point denotes the test error vs. computational cost at every time step during a test run of each of the 5 concrete SNNs, and the curves denote the test errors and computational costs averaged over the 5 concrete SNNs.

TABLE 2. Final/stable test errors (%) and computational costs (# spikes) for CNN-small and CNN-large.

SNN type	CNN-small			CNN-large		
	ANN test error	SNN test error	Comp. cost	ANN test error	SNN test error	Comp. cost
SNN-rate-IF-rst-zero [16]	0.75%	0.89%	5.19M	0.73%	0.85%	31.95M
SNN-rate-IF-rst-subtract [17]	0.75%	0.84%	3.98M	0.73%	0.82%	35.94M
SNN-ASN [18]	0.57%	0.65%	2.58M	0.55%	0.66%	10.05M
SNN-multi-spike-LTC [this work]	0.77%	0.77%	0.76M	0.62%	0.62%	2.37M
SNN-single-spike-LTC [this work]	0.97%	0.97%	0.64M	0.59%	0.59%	2.29M



(a) Test error vs. computational cost for CNN-small

(b) Close-up view of the region in the green box in (a)

(c) Test error vs. computational cost for CNN-large

(d) Close-up view of the region in the green box in (c)

FIGURE 4. Test errors vs. computational costs of SNNs for either the CNN-small or the CNN-large architecture.

For the SNN-LTC types, each point (triangle or square) denotes the final test error and computational cost of each of the 5 concrete SNNs. Grey vertical lines mark the *stable* computational costs of rate-coding SNN types and the *final* computational costs of the SNN-LTC types, averaged over 5 concrete SNNs.

We can draw the following conclusions from Fig. 4 and Table 2:

- The SNN-LTC types are able to achieve comparable classification accuracy with significantly reduced computational cost as measured by the number of spikes during each forward inference, compared to related work. The differences in classification accuracy are small and insignificant, and the reduced computational cost is the more important issue in this context, which may translate into reduced power consumption or hardware cost in the implementation.
- The SNN-LTC types have more deterministic timing delay than rate-coding SNN types, hence more suitable for real-time systems with strict deadline constraints.

V. CONCLUSIONS

In this paper, we propose an efficient ANN-to-SNN conversion method based on novel Logarithmic Temporal Coding, and the Exponentiate-and-Fire neuron model. Experimental results indicate that the resulting SNN achieves competitive performance in terms of classification accuracy at significantly lower computational cost than related work.

SUPPLEMENTARY MATERIAL

A. LOGARITHMIC GROWTH OF THE NUMBER OF SPIKES TO ENCODE AN ACTIVATION FOR MULTI-SPIKE LTC

In this section, we prove Proposition 1 in Section III-B.

Proof: Let $\tilde{a}$ be the multi-spike LA of a. Any power 2^e with an integer exponent $e \geq \lfloor \log_2 a \rfloor + 1$ cannot contribute to $\tilde{a}$, because $2^e > a \geq \tilde{a}$. For a power 2^e with an integer exponent e to contribute to $\tilde{a}$, $e \in (-\infty, \lfloor \log_2 a \rfloor] \cap [e_{min}, e_{max}]$.

If $a < 2^{e_{min}}$, $(-\infty, \lfloor \log_2 a \rfloor] \cap [e_{min}, e_{max}] = \emptyset$. Hence, no power of two contributes to the multi-spike LA of a. According to LTC, the spike train contains no spike, hence $n_s = 0$.

If $2^{e_{min}} \leq a < 2^{e_{max}+1}$, $(-\infty, \lfloor \log_2 a \rfloor] \cap [e_{min}, e_{max}] = [e_{min}, \lfloor \log_2 a \rfloor]$. In the worst-case, every 2^e with an integer exponent in the set $[e_{min}, \lfloor \log_2 a \rfloor]$ contributes to $\tilde{a}$. Thus, $n_s \leq \lfloor \log_2 a \rfloor - e_{min} + 1$.

If $a \geq 2^{e_{max}+1}$, $(-\infty, \lfloor \log_2 a \rfloor] \cap [e_{min}, e_{max}] = [e_{min}, e_{max}]$. Every 2^e with an integer exponent $e \in [e_{min}, e_{max}]$ contributes to $\tilde{a}$, hence $n_s = e_{max} - e_{min} + 1$. □

B. AN EF NEURON COMPUTES A WEIGHTED SUM OF THE LAS OF ITS INPUT SPIKE TRAINS

In this section, we prove Lemma 1 in Section III-C.1.

Proof: According to LA, $\tilde{a}_i = \sum_k 2^{e^{in}_{i,k}}$. According to LTC, the spike time corresponding to the power $2^{e^{in}_{i,k}}$ is $t^{in}_{i,k} = e^{in}_{max} - e^{in}_{i,k}$. The total PSP elicited by the i-th input LTC spike train at $t = T^{in} - 1$ is given by

$$h_i(T^{in} - 1) = \sum_{t^{in}_{i,k} \in \mathcal{F}^{in}_i} \epsilon(T^{in} - 1 - t^{in}_{i,k}) = \sum_k 2^{e^{in}_{i,k}} = \tilde{a}_i \tag{S1}$$

Since the EF neuron does not fire any output spike before $t = T^{in} - 1$, $V^-_m(T^{in} - 1)$ reduces to a weighted sum of PSPs elicited by the input spike trains:

$$V^-_m(T^{in} - 1) = \sum_{i \in \Gamma} w_i \cdot h_i(T^{in} - 1) = \sum_{i \in \Gamma} w_i \cdot \tilde{a}_i \tag{S2}$$

completing the proof. □

C. AN EF NEURON GENERATES AN LTC SPIKE TRAIN

In this section, we prove Lemma 2 in Section III-C.2.

We observe that an EF neuron doubles its membrane potential every time step if the neuron does not receive or fire any spike, as Lemma 3 states.

Lemma 3: Let $t_0, t \in \mathbb{Z}$ be two time steps, where $t_0 < t$. The pre-reset membrane potential of an EF neuron $V^-_m(t) = V_m(t_0) \cdot 2^{t-t_0}$ if the following conditions hold:

1) *The neuron does not receive any input spike during the time interval $[t_0 + 1, t]$, and*
2) *The neuron does not fire any output spike during the time interval $[t_0 + 1, t - 1]$.*

Proof: Lemma 3 follows from the definitions of the pre- and post-reset membrane potentials (Eqns. (3) and (4)) and the exponentially growing postsynaptic potential and afterhyperpolarizing potential kernels (Eqns. (5), (7) and (9)). □

With Lemma 3, we prove Lemma 2 below.

Proof: Depending on the value of $V^-_m(T^{in} - 1)$, there are four cases: $V^-_m(T^{in} - 1) \leq 0$, $0 < V^-_m(T^{in} - 1) < 2^{e^{out}_{min}}$, $2^{e^{out}_{min}} \leq V^-_m(T^{in} - 1) < 2^{e^{out}_{max}+1}$, and $V^-_m(T^{in} - 1) \geq 2^{e^{out}_{max}+1}$.

If $V^-_m(T^{in} - 1) \leq 0$, the logarithmic approximation of $\max(V^-_m(T^{in} - 1), 0)$ is zero, and the desired LTC spike train contains no spikes. For the EF neuron, by Lemma 3, $V^-_m(t)$ remains zero or negative during the output time window, and the neuron does not fire any output spike during this time interval. Hence, the neuron generates the desired LTC spike train within its output time window.

If $0 < V^-_m(T^{in} - 1) < 2^{e^{out}_{min}}$, with exponent range $[e^{out}_{min}, e^{out}_{max}]$, the logarithmic approximation of $\max(V^-_m(T^{in} - 1), 0)$ is zero, and the desired LTC spike train contains no spikes.

For the EF neuron, by Lemma 3, the first output spike time t^{out}_0 satisfies the following condition:

$$V^-_m(t^{out}_0) = V^-_m(T^{in} - 1) \cdot 2^{t^{out}_0 - (T^{in} - 1)} \geq V_{th} \tag{S3}$$

Solving the inequality for the minimum integer value for t^{out}_0, we have

$$t^{out}_0 = e^{out}_{max} - \lfloor \log_2 V^-_m(T^{in} - 1) \rfloor + (T^{in} - 1) \tag{S4}$$

Since $V^-_m(T^{in} - 1) < 2^{e^{out}_{min}}$, $t^{out}_0 > T^{in} + T^{out} - 2$. In other words, the neuron fires the first output spike after the end of its output time window. Hence, the neuron generates the desired LTC spike train within its output time window.

For the remaining cases, we derive the spike times of the desired LTC spike train and the output spike times of the EF neuron, and show that the output spike train of the EF neuron is consistent with the desired LTC spike train at the end of this proof.

If $2^{e^{out}_{min}} \leq V^-_m(T^{in} - 1) < 2^{e^{out}_{max}+1}$, $2^{e^{out}_{min}} \leq a = \max(V^-_m(T^{in} - 1), 0) < 2^{e^{out}_{max}+1}$. In this case, Eqn. (1) can be formulated as

$$\tilde{a} = \sum_k 2^{e_k} \tag{S5}$$

$$e_k = \begin{cases} \lfloor \log_2 a \rfloor & \text{if } k = 0, \\ \lfloor \log_2 (a - \sum_{k'=0}^{k-1} 2^{e_{k'}}) \rfloor & \text{if } k > 0 \end{cases} \tag{S6}$$

$$\forall k, e_k \geq e_{min} \tag{S7}$$

where the sum in Eqn. (S6) runs across exponents $\{e_0, e_1, \ldots\}$ from $\lfloor \log_2 a \rfloor$ to the smallest $e_k \geq e_{min}$. Note that 2^{e_0} gives

the single-spike LA of a. By substituting $e_k^{out} = e_{max}^{out} - t_k^{out}$ and $a = V_m^-(T^{in}-1)$ into Eqn. (S7), we derive the spike times of the desired LTC spike train:

$$t_k^{out} = e_{max}^{out} - \lfloor \log_2 (V_m^-(T^{in}-1) - \sum_{k'=0}^{k-1} 2^{e_{max}^{out} - t_{k'}^{out}}) \rfloor \tag{S8}$$

where t_k^{out} is the $(k+1)$-th output spike time. For both multi-spike LTC and single-spike LTC, Eqn. (S8) gives the first spike time t_0^{out}. For multi-spike LTC, Eqn. (S8) also gives subsequent spike times. By substituting $e_k^{out} = e_{max}^{out} - t_k^{out}$ and $e_{min}^{out} = e_{max}^{out} - (T^{out}-1)$ into Inequality S7, we derive the following constraint on the spike times:

$$\forall k, t_k^{out} \le T^{out} - 1 \tag{S9}$$

For the EF neuron, every output spike time t_k^{out} within the output time window satisfies the following conditions:

$$V_m^-(t_k^{out}) \ge V_{th} \tag{S10}$$

$$t_k^{out} \le T^{in} + T^{out} - 2 \tag{S11}$$

The first output spike may fired either at time $t_0^{out} = T^{in} - 1$, if $V_m^-(T^{in}-1) \ge 2^{e_{max}^{out}}$; or at time $t_0^{out} > T^{in} - 1$ by Lemma 3, if $V_m^-(T^{in}-1) < 2^{e_{max}^{out}}$. In both cases, the first output spike time t_0^{out} satisfies Conditions S3 and S4.

In the case of single-spike LTC, the EF neuron fires a single output spike at time t_0^{out}. In the case of multi-spike LTC, the EF neuron may fire subsequent output spikes. Consider every two consecutive output spike times t_{k-1}^{out} and t_k^{out}, where $t_{k-1}^{out} < t_k^{out}$. By Lemma 3, the pre-reset membrane potentials $V_m^-(t_k^{out})$ can be formulated as

$$V_m^-(t_k^{out}) = (V_m^-(t_{k-1}^{out}) - 2^{e_{max}^{out}}) \cdot 2^{t_k^{out} - t_{k-1}^{out}} \tag{S12}$$

$$V_m^-(t_0^{out}) = V_m^-(T^{in}-1) \cdot 2^{t_0^{out} - (T^{in}-1)} \tag{S13}$$

Solving the recurrence relation above, we have

$$V_m^-(t_k^{out}) = 2^{t_k^{out}} \cdot (V_m^-(T^{in}-1) \cdot 2^{-(T^{in}-1)} - \sum_{k'=0}^{k-1} 2^{e_{max}^{out} - t_{k'}^{out}}) \tag{S14}$$

By substituting Eqn. (S14) into Eqn. (S11) and solving the resulting inequality for the minimum integer value for t_k^{out}, we have

$$t_k^{out} - (T^{in}-1) = e_{max}^{out} - \lfloor \log_2 \left(V_m^-(T^{in}-1) - \sum_{k'=0}^{k-1} 2^{e_{max}^{out} - (t_{k'}^{out} - (T^{in}-1))} \right) \rfloor \tag{S15}$$

If $V_m^-(T^{in}-1) \ge 2^{e_{max}^{out}+1}$, $a = \max(V_m^-(T^{in}-1), 0) \ge 2^{e_{max}^{out}+1}$. In this case, Eqn. (1) can be formulated as

$$\tilde{a} = \sum_{k=0}^{e_{max}-e_{min}} 2^{e_k} \tag{S16}$$

$$e_k = e_{max} - k \tag{S17}$$

Note that 2^{e_0} gives the single-spike LA of a. By substituting $e_k^{out} = e_{max}^{out} - t_k^{out}$ into Eqn. (S17), we derive the spike times of the desired LTC spike train:

$$\forall k \in [0, T^{out}-1], \quad t_k^{out} = k \tag{S18}$$

For both multi-spike LTC and single-spike LTC, Eqn. (S18) gives the first spike time $t_0^{out} = 0$. For multi-spike LTC, Eqn. (S18) also gives subsequent spike times.

For the EF neuron, since $V_m^-(T^{in}-1) \ge 2^{e_{max}^{out}+1} > V_{th}$, the first output spike time is

$$t_0^{out} = T^{in} - 1 \tag{S19}$$

In the case of single-spike LTC, the EF neuron fires only a single output spike at time t_0^{out}. In the case of multi-spike LTC, suppose the EF neuron fires an output spike at time t_k^{out}. By Lemma 3,

$$V_m^-(t_k^{out}+1) = 2(V_m^-(t_k^{out}) - 2^{e_{max}^{out}}) \tag{S20}$$

It is easy to see that, if $V_m^-(t_k^{out}) \ge 2^{e_{max}^{out}+1}$, then $V_m^-(t_k^{out}+1) \ge 2^{e_{max}^{out}+1} > V_{th}$, and the next output spike will be fired at time $t_{k+1}^{out} = t_k^{out} + 1$. Since $V_m^-(t_0^{out}) = V_m^-(T^{in}-1) \ge 2^{e_{max}^{out}+1}$, the EF neuron fires an output spike at every time step within its output time window. Hence,

$$\forall k \in [0, T^{out}-1], \quad t_k^{out} = T^{in} - 1 + k \tag{S21}$$

By comparing Eqns. (S4) and (S15) with Eqn. (S8), Eqn. (S11) with Eqn. (S9), and Eqn. (S19) and Eqn. (S21) with Eqn. (S18), it can be seen that the output spike train of the EF neuron within its output time window is consistent with the desired LTC spike train, except that every output spike time of the EF neuron is $T^{in}-1$ larger than the corresponding spike time of the desired LTC spike train. The difference is due to the fact that the output time window of the EF neuron starts at the time step $T^{in}-1$.

Therefore, in all cases, the EF neuron generates an LTC spike train that encodes $\max(V_m^-(T^{in}-1), 0)$ within its output time window, completing the proof. □

REFERENCES

[1] M. Zhang, Z. Gu, and G. Pan, "A survey of neuromorphic computing based on spiking neural networks," *Chin. J. Electron.*, vol. 27, no. 4, pp. 667–674, Jul. 2018.
[2] D. Ma, J. Shen, Z. Gu, M. Zhang, X. Zhu, X. Xu, Q. Xu, Y. Shen, and G. Pan, "Darwin: A neuromorphic hardware co-processor based on spiking neural networks," *J. Syst. Archit.*, vol. 77, pp. 43–51, Jun. 2017.
[3] J. Shen, D. Ma, Z. Gu, M. Zhang, X. Zhu, X. Xu, Q. Xu, Y. Shen, and G. Pan, "Darwin: A neuromorphic hardware co-processor based on spiking neural networks," *Sci. China Inf. Sci.*, vol. 59, no. 2, pp. 1–5, Feb. 2016.
[4] Z. Wu, R. Reddy, G. Pan, N. Zheng, P. F. Verschure, Q. Zhang, X. Zheng, J. C. Principe, A. Kreilinger, and M. Rohm, "The convergence of machine and biological intelligence," *IEEE Intelligent Systems*, vol. 28, no. 5, pp. 28–43, Sep. 2013.
[5] Z. Wu, N. Zheng, S. Zhang, X. Zheng, L. Gao, and L. Su, "Maze learning by a hybrid brain-computer system," *Sci. Rep.*, vol. 6, no. 1, pp. 1–12, Sep. 2016.
[6] Q. Liu, G. Pan, H. Ruan, D. Xing, Q. Xu, and H. Tang, "Unsupervised AER object recognition based on multiscale spatio-temporal features and spiking neurons," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Feb. 10, 2020, doi: 10.1109/TNNLS.2020.2966058.
[7] S. M. Bohte, J. N. Kok, and J. A. La Poutré, "SpikeProp: Backpropagation for networks of spiking neurons," in *Proc. ESANN*, 2000, pp. 419–424.

[8] R. Gütig and H. Sompolinsky, "The tempotron: A neuron that learns spike timing–based decisions," *Nature Neurosci.*, vol. 9, no. 3, pp. 420–428, 2006.
[9] F. Ponulak and A. Kasiński, "Supervised learning in spiking neural networks with ReSuMe: Sequence learning, classification, and spike shifting," *Neural Comput.*, vol. 22, no. 2, pp. 467–510, Feb. 2010.
[10] R. V. Florian, "The chronotron: A neuron that learns to fire temporally precise spike patterns," *PLoS ONE*, vol. 7, no. 8, Aug. 2012, Art. no. e40233.
[11] A. Mohemmed, S. Schliebs, S. Matsuda, and N. Kasabov, "Span: Spike pattern association neuron for learning spatio-temporal spike patterns," *Int. J. Neural Syst.*, vol. 22, no. 4, Aug. 2012, Art. no. 1250012.
[12] Q. Yu, H. Tang, K. C. Tan, and H. Li, "Precise-spike-driven synaptic plasticity: Learning hetero-association of spatiotemporal spike patterns," *PLoS ONE*, vol. 8, no. 11, Nov. 2013, Art. no. e78318.
[13] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *Int. J. Comput. Vis.*, vol. 113, no. 1, pp. 54–66, May 2015.
[14] E. Hunsberger and C. Eliasmith, "Spiking deep networks with LIF neurons," *CoRR*, vol. abs/1510.08829, 2015. [Online]. Available: http://arxiv.org/abs/1510.08829
[15] Q. Liu, Y. Chen, and S. B. Furber, "Noisy Softplus: An activation function that enables SNNs to be trained as ANNs," *CoRR*, vol. abs/1706.03609, 2017. [Online]. Available: http://arxiv.org/abs/1706.03609
[16] 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 *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2015, pp. 1–8.
[17] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers Neurosci.*, vol. 11, p. 682, Dec. 2017.
[18] D. Zambrano, R. Nusselder, H. S. Scholte, and S. M. Bohte, "Efficient computation in adaptive artificial spiking neural networks," *CoRR*, vol. abs/1710.04838, 2017. [Online]. Available: http://arxiv.org/abs/1710.04838
[19] N. Sengupta and N. Kasabov, "Spike-time encoding as a data compression technique for pattern recognition of temporal data," *Inf. Sci.*, vols. 406–407, pp. 133–145, Sep. 2017.
[20] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," *CoRR*, vol. abs/1603.01025, 2016. [Online]. Available: http://arxiv.org/abs/1603.01025
[21] S. Vogel, M. Liang, A. Guntoro, W. Stechele, and G. Ascheid, "Efficient hardware acceleration of CNNs using logarithmic data representation with arbitrary log-base," in *Proc. Int. Conf. Comput.-Aided Design*, Nov. 2018, p. 9.
[22] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge, U.K.: Cambridge Univ. Press, 2002.
[23] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[24] TensorFlow. (2020). *A Guide to TF Layers: Building a Convolutional Neural Network*. [Online]. Available: https://tensorflow.google.cn/tutorials