Spike Accumulation Forwarding for Effective Training of Spiking Neural Networks

Anonymous authors
Paper under double-blind review

Abstract

In this article, we propose a new paradigm for training spiking neural networks (SNNs), *spike accumulation forwarding* (*SAF*). It is known that SNNs are energy-efficient but difficult to train. Consequently, many researchers have proposed various methods to solve this problem, among which online training through time (OTTT) is a method that allows inferring at each time step while suppressing the memory cost. However, to compute efficiently on GPUs, OTTT requires operations with spike trains and weighted summation of spike trains during forwarding. In addition, OTTT has shown a relationship with the Spike Representation, an alternative training method, though theoretical agreement with Spike Representation has not to be proven. Our proposed method can solve these problems; namely, SAF can halve the number of operations during the forward process, and it can be theoretically proven that SAF is consistent with the Spike Representation and OTTT, respectively. Furthermore, we confirmed the above contents through experiments and showed that it is possible to reduce memory and training time while maintaining accuracy.

1 Introduction

Due to the carbon emission reduction problem, energy-efficient spiking neural networks (SNNs) are attracting attention (Luo et al., 2023). SNNs are known to be more bio-plausible models than artificial neural networks (ANNs) and can replace the multiply-accumulate (MAC) operations with additive operations. This characteristic comes from propagating the spike train (belonging to $\{0,1\}^T$, where T is the number of time steps) and is energy-efficient on neuromorphic chips (Akopyan et al., 2015; Davies et al., 2018).

Despite the usefulness of SNNs for CO_2 reduction, their neurons are non-differentiable, which makes them difficult to train. Solving this problem is in the mainstream of SNN research, and back-propagation through time (BPTT) with surrogate gradient (SG) (Zheng et al., 2021; Xiao et al., 2022) is one of the main methods to achieve high performance. In particular, the recently proposed Online Training Through Time (OTTT) can train SNNs at each time step, just like our brains, and achieve high performance with few time steps (Xiao et al., 2022).

To enable online training, OTTT uses different information for the forward and backward processes. For forwarding, the spike train is used; for backwarding, the weighted summation of spike trains (which we refer to as *spike accumulation*) is used. Therefore, efficient computation on GPUs using the Autograd of libraries such as PyTorch (Paszke et al., 2019) requires operations with spike train and spike accumulation during the forward process (see Fig. 1). Meanwhile, OTTT has the theoretical guarantee that the gradient descent direction is similar to that of Spike Representations by the weighted firing rate coding by summing up the gradients of each time step (Xiao et al., 2021; Meng et al., 2022). However, these gradients are not shown to be perfectly consistent. To accurately bridge them, it is essential to develop a method that guarantees the gradient can be consistent with each of above two gradients.

In this article, we propose *Spike Accumulation Forwarding* (*SAF*) as a new paradigm for training SNNs. Unlike OTTT, SAF propagates not only backward but also forward processes by spike accumulation (see

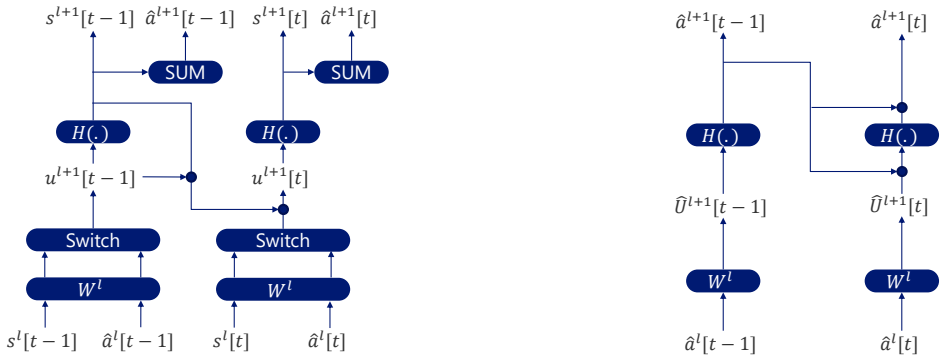


Figure 1: Overview of OTTT and SAF training. OTTT requires operations with spike $\boldsymbol{s}^l[t]$ and spike accumulation $\widehat{\boldsymbol{a}}^l[t]$ during the forward process, while SAF requires operations with $\widehat{\boldsymbol{a}}^l[t]$. Also, $\boldsymbol{u}^l[t]$ represents the membrane potential, and $\widehat{\boldsymbol{U}}^{l+1}[t]$ represents the potential accumulation (see Sec. 4). The SUM layer computes the spike accumulation, and the Switch layer propagates $\boldsymbol{W}^l\boldsymbol{s}$ and $\boldsymbol{W}^l\widehat{\boldsymbol{a}}$ in forwarding and backwarding, respectively. Note that, unlike membrane potentials, the potential accumulation does not require the retention of past information.

Fig. 1). By using this process, we can halve the number of operations during the forward process. In addition, because SAF does not require retaining the information of membrane potentials as in Zhou et al. (2021), we can reduce memory usage during training compared to OTTT. Furthermore, this propagation strategy is only executed during training, and during inference, we can replace the propagation with the spike train without approximation error. We demonstrate this by proving that the neurons for spike accumulation are identical to the Leaky-Integrate-and-Fire (LIF) (Stein, 1965) neuron, which is a generalization of the Integrate-and-Fire (IF) (Lapique, 1907) neuron, which are commonly used as SNN neurons. This result indicates that the SNN composed of LIF neurons can achieve the same accuracy using the trained parameters of SAF (i.e., SAF is capable of inference by the SNN composed of LIF neurons). Furthermore, we prove that the gradient of the SAF is consistent with that of the OTTT, which trains at each time step, and that by summing up the gradients at each time step, SAF is also consistent with the gradient of the Spike Representation. This shows that SAF can accurately bridge the gap between Spike Representation and OTTT.

Main Contributions

(A) We propose SAF, which trains SNNs by only spike accumulation, halving the number of operations in the forward process, reducing the memory cost, and enabling inference on SNNs composed of LIF neurons.

(B) We prove theoretically that the neurons for spike accumulation are absolutely identical to the LIF neuron.

(C) Our study also shows that the gradient of SAF is theoretically consistent with the gradient of the Spike Representation and also with that of OTTT, which trains each time step.

(D) We consider the situation that SNNs have a feedforward or feedback connection like brain and discuss the equivalence of SAF with OTTT and with Spike Representation.

(E) Brief experiments confirmed that for training at each time step, the training results were in close agreement with OTTT while reducing the training cost.

2 Related Work

Regarding SNN training, there are two research directions: conversion from ANN to SNN and direct training. The conversion approach reuses the parameters of the ANN while converting the activation function for the spiking function (Diehl et al., 2015; Deng and Gu, 2020; Han et al., 2020). This approach can be employed by a wide range of many trained deep-learning models, and there are use cases for tasks other than recognition (Kim et al., 2020; Qu et al., 2023). However, because the accuracy tends to be proportional to the number of time steps and although several improvement methods have been proposed (Chowdhury et al., 2021; Wu et al., 2021), high-precision inference is still difficult for small time steps. Meanwhile, direct training does not use the parameters of the trained ANNs. Thus, the non-differentiable SNNs are trained by some approximation techniques. One of the most significant techniques is to utilize the surrogate gradient (SG). SGs enables backpropagation by approximating the gradient of non-differentiable activation functions, and various types of SGs (e.g., rectangular or derivatives of sigmoid functions) have been proposed (Shrestha and Orchard, 2018; Wu et al., 2018; Lian et al., 2023; Suetake et al., 2023). Other recently proposed methods include those based on the clamp function (Meng et al., 2022) or implicit differentiation on the equilibrium state (Xiao et al., 2021). These employ Spike representation, which propagates information such as firing rates, not spike trains, and have the advantage of being able to train SNNs like ANNs (Thiele et al., 2019; Zhou et al., 2021). However, because these methods assume $T \to \infty$, T must be large to achieve high accuracy. In addition, these are only differences in how to approximate; the basic approach is similar to SGs. Although there are bio-inspired training methods, such as Hebbian learning rule (Hebb, 2005; Frémaux and Gerstner, 2016) and spike timing dependent plasticity (STBP) (Bi and Poo, 1998; Bengio et al., 2015). In particular, learning rules based on eligibility traces, such as SuperSpike (Zenke and Ganguli, 2018), a method for improving SpikeProp (Bohte et al., 2002), are associated with three-factor plasticity rules (Neftci et al., 2019). OTTT, the main focus of our study, has also been mentioned as being associated with the three-factor plasticity rule. However, training multi-layered SNNs with SuperSpike is challenging, and it is necessary to introduce local errors in each layer (Kaiser et al., 2020). Nonetheless, in Kaiser et al. (2020), fewer than ten layers were trained, and training deeper SNNs has still been challenging. On the other hand, OTTT can train deeper networks such as VGG, which is useful for many applications.

In the following, we discuss the OTTT (Xiao et al., 2022) most relevant to our study. Because OTTT is a variant of BPTT with SG, it allows for low-latency training. In addition, it is sufficient for OTTT to maintain the computational graph only for the current time step during training, different from the standard BPTT with SG. Thus, training can be performed with constant memory usage even as time steps increase. However, OTTT requires additional information for propagating the spike accumulation as well as the spike train for the forward process, which can increase training time. In addition, because OTTT is based on the LIF neuron, it must retain the membrane potential, which can increase memory usage. Furthermore, it is important to note that while OTTT and Spike Representation have similar gradient directions, their gradients do not necessarily match exactly (i.e., the inner product between their gradients is positive).

3 Preliminaries

3.1 Typical Neuron Model

In this subsection, we explain the LIF neuron, which is widely used in SNNs. The LIF neuron is a neuron model that considers the leakage of the membrane potential, and its discrete computational form is as follows:

$$\begin{cases} \boldsymbol{u}^{l+1}[t] = \lambda(\boldsymbol{u}^{l+1}[t-1] - V_{\mathrm{th}}\,\boldsymbol{s}^{l+1}[t-1]) + \boldsymbol{W}^l \boldsymbol{s}^l[t] + \boldsymbol{b}^{l+1}, \\ \boldsymbol{s}^{l+1}[t] = H(\boldsymbol{u}^{l+1}[t] - V_{\mathrm{th}}), \end{cases} \tag{1}$$

where $\boldsymbol{s}^l[t]$, $\boldsymbol{u}^l[t]$, $\boldsymbol{W}^l$ and $\boldsymbol{b}^l$ are the spike train, membrane potential, weight and bias of l-th layer, respectively. $\lambda \leq 1$ is the leaky term, and λ is set to 1 if we use the IF neuron, which is a special case of the LIF neuron. Also, H is the element-wise Heaviside step function, that is, $H = 1$ when the membrane potential $\boldsymbol{u}[t]$ exceeds

the threshold V_{th}. From this equation, the membrane potentials $\boldsymbol{u}^l[t]$ are computed sequentially and must retain the previous membrane potential $\boldsymbol{u}^l[t-1]$. This is the same in the case of OTTT, which uses the LIF neuron as described below.

3.2 Training methods for SNNs

This subsection introduces two training methods that are closely related to our method: Spike Representation and OTTT.

Spike Representation

Spike Representation is a method of training SNNs by propagating information differently to the spike trains (Xiao et al., 2021; Meng et al., 2022). In this article, we consider the weighted firing rate $\boldsymbol{a}[t] = \sum_{\tau=0}^{t} \lambda^{t-\tau} \boldsymbol{s}[\tau] / \sum_{\tau=0}^{t} \lambda^{t-\tau}$ as in Xiao et al. (2021); Meng et al. (2022); Xiao et al. (2022). Likewise, we define the weighted average input $\boldsymbol{m}[t] = \sum_{\tau=0}^{t} \lambda^{t-\tau} \boldsymbol{x}[\tau] / \sum_{\tau=0}^{t} \lambda^{t-\tau}$, where x is the value of the input data. Then, given a convergent sequence $\boldsymbol{m}[t] \to \boldsymbol{m}^*$ $(t \to \infty)$, it is known that $\boldsymbol{a}[t] \to \sigma\left(\boldsymbol{m}^* / V_{\mathrm{th}}\right)$ $(t \to \infty)$ holds (Xiao et al., 2021), where σ is the element-wise clamp function $\sigma(x) = \min(\max(0, x), 1)$. Using this convergence and under the assumption that the time step T is sufficiently large, the weighted firing rate in the $(l+1)$-th layer is approximated as $\boldsymbol{a}^{l+1}[T] \approx \sigma(\boldsymbol{W}^l \boldsymbol{a}^l[T] + \boldsymbol{b}^{l+1}/V_{\mathrm{th}})$. We consider the loss L as $L[t] = \mathcal{L}(\boldsymbol{a}[T], y)$ (where $\mathcal{L}$ is a convex function like cross-entropy and y is the label). Then the gradient of L with respect to $\boldsymbol{W}^l$ is computed as follows:

$$\left(\frac{\partial L}{\partial \boldsymbol{W}^l}\right)_{\mathrm{SR}} = \frac{\partial L}{\partial \boldsymbol{a}^N[T]} \left(\prod_{i=N-1}^{l+1} \frac{\partial \boldsymbol{a}^{i+1}[T]}{\partial \boldsymbol{a}^i[T]}\right) \frac{\partial \boldsymbol{a}^{l+1}[T]}{\partial \boldsymbol{W}^l}, \tag{2}$$

where N represents the number of layers.

Spike Representation can include a feedforward or feedback connections, which have been frequently used in recent years (Xiao et al., 2021; 2022; 2023), where notice that, feedforward and feedback connections do not refer to the weights between adjacent layers but to additional weight matrices connecting any layers l and l'. When there is a feedforward connection, the gradient (2) holds. However, in the case of feedback connection, it does not hold because we need to calculate implicit differentiation. See Appendix B.6 for a detailed explanation.

Online Training Through Time

OTTT (Xiao et al., 2022) is a training method based on BPTT with the surrogate gradient (SG). BPTT with SG enables low-latency training; however, during training, it requires the computational graph to be maintained at each time step, resulting in substantial memory usage when a large number of time steps are involved. OTTT solves this problem and allows for training with minimal memory consumption.

In OTTT, for the forward process, (weighted) spike accumulation $\widehat{\boldsymbol{a}}[t] = \sum_{\tau=0}^{t} \lambda^{t-\tau} \boldsymbol{s}[\tau]$ is propagated in addition to spike trains $\boldsymbol{s}[t]$ computed by the LIF neuron. Defining the loss at each time step as $L[t] = \mathcal{L}(\boldsymbol{s}[t], y)/T$, OTTT computes the gradient at time t as follows:

$$\left(\frac{\partial L[t]}{\partial \boldsymbol{W}^l}\right)_{\mathrm{OT}} = \widehat{\boldsymbol{a}}^l[t] \frac{\partial L[t]}{\partial \boldsymbol{s}^N[t]} \left(\prod_{i=N-1}^{l+1} \frac{\partial \boldsymbol{s}^{i+1}[t]}{\partial \boldsymbol{s}^i[t]}\right) \frac{\partial \boldsymbol{s}^{l+1}[t]}{\partial \boldsymbol{u}^{l+1}[t]}. \tag{3}$$

Note that since this equation uses the accumulated information of spikes up to $\widehat{a}^l[t]$ at the current time t, OTTT implicitly trains using information up to t, not just the current time. This corresponds to the fact that membrane potentials accumulate information from the past. Note also that the term $\partial \boldsymbol{s}/\partial \boldsymbol{u}$ is non-differentiable at $u = V_{\mathrm{th}}$; thus, we approximate it with the SG; for example, since $\partial \boldsymbol{s}^{i+1}[t]/\partial \boldsymbol{s}^i[t]$ can decomposed into $(\partial \boldsymbol{s}^{i+1}[t]/\partial \boldsymbol{u}^{i+1}[t])(\boldsymbol{u}^{i+1}[t]/\partial \boldsymbol{s}^i[t])$, we also replace $\partial \boldsymbol{s}^{i+1}[t]/\partial \boldsymbol{u}^{i+1}[t]$ with the SG. Like the

case of Spike Representation, OTTT can include a feedforward or feedback connection, and the gradients are almost the same as (3).

Xiao et al. (2022) proposed two types of training approaches: $\mathrm{OTTT_O}$, where parameters are updated at each time step using $\partial L[t]/\partial \boldsymbol{W}^l$, and $\mathrm{OTTT_A}$, where parameters are updated collectively by summing $\partial L[t]/\partial \boldsymbol{W}^l$ up to T.

In particular, they proved that the gradient descent directions in $\mathrm{OTTT_A}$ and Spike Representation are similar, i.e., the inner product between their gradients is positive.

4 Spike Accumulation Forwarding

In this section, we introduce our proposed method, SAF, which only propagates (weighted) spike accumulation $\widehat{\boldsymbol{a}}[t]$. We first explain the forward and backward processes. Then, we prove that SAF can be consistent with OTTT and Spike Representation. We also show that the feedback connection can be added to SAF, as in Xiao et al. (2022), and furthermore, feedforward connections can also be incorporated into the SAF. For summaries of the main formulas, see App. A.

4.1 Details of SAF

Forward process

As mentioned earlier, for the forward processes of conventional SNNs, the spike trains $\boldsymbol{s}[t]$ are propagated. In other words, the firing state of the spike for each neuron at each time step is retained. In SAF, as in OTTT and SR, instead of the spike trains, it propagates (weighted) spike accumulation $\widehat{\boldsymbol{a}}[t] = \sum_{\tau=0}^{t} \lambda^{t-\tau} \boldsymbol{s}[\tau]$, meaning that it retains the (weighted) count of the fired spikes up to the current time for each neuron. Additionally, although in conventional SNNs, the spike firing is determined with the membrane potential $\boldsymbol{u}[t]$, in SAF, it is determined with (weighted) potential accumulation $\widehat{\boldsymbol{U}}[t]$ defined by $\widehat{\boldsymbol{U}}^{l+1}[t] = \lambda \widehat{\boldsymbol{U}}^{l+1}[t-1] + \boldsymbol{W}^l(\widehat{\boldsymbol{a}}^l[t] - \lambda \widehat{\boldsymbol{a}}^l[t-1]) + \boldsymbol{b}^{l+1}$, which corresponds to the membrane potential in the relation (1). With these considerations, SAF is updated as follows:

$$\begin{cases} \widehat{\boldsymbol{U}}^{l+1}[t] = \boldsymbol{W}^l \widehat{\boldsymbol{a}}^l[t] + \boldsymbol{b}^{l+1} \sum_{\tau=0}^{t-1} \lambda^{t-\tau} + \lambda^t \widehat{\boldsymbol{U}}^{l+1}[0], \\ \widehat{\boldsymbol{a}}^{l+1}[t] = \lambda \widehat{\boldsymbol{a}}^{l+1}[t-1] + H(\widehat{\boldsymbol{U}}^{l+1}[t] - V_{\mathrm{th}}(\lambda \widehat{\boldsymbol{a}}^{l+1}[t-1] + 1)), \end{cases} \tag{4}$$

where $\widehat{\boldsymbol{U}}^{l+1}[0]$ is the initial value for potential accumulation, and here, we assume it to be the initial membrane potential $\boldsymbol{u}^{l+1}[0]$. Here, the membrane potential $\boldsymbol{u}^{l+1}[t]$ and spike trains $\boldsymbol{s}^{l+1}[t]$ in the LIF model can be expressed by $\widehat{\boldsymbol{U}}^{l+1}[t]$ and $\widehat{\boldsymbol{a}}^{l+1}[t-1]$ as follows:

$$\begin{cases} \boldsymbol{u}^{l+1}[t] = \widehat{\boldsymbol{U}}^{l+1}[t] - V_{\mathrm{th}}\, \lambda \widehat{\boldsymbol{a}}^{l+1}[t-1], \\ \boldsymbol{s}^{l+1}[t] = H(\widehat{\boldsymbol{U}}^{l+1}[t] - V_{\mathrm{th}}(\lambda \widehat{\boldsymbol{a}}^{l+1}[t-1] + 1)). \end{cases} \tag{5}$$

For derivations of (4) and (5), refer to Appendix B.1. Note that, as shown in (4), SAF does not need to retain the past potential accumulation $\widehat{\boldsymbol{U}}^{l+1}[t-1]$. Meanwhile, the various SNNs, including OTTT, require the LIF neurons used for training to retain the previous membrane potentials $\boldsymbol{u}^{l+1}[t-1]$, as described above. Therefore, SAF can reduce the memory usage for the forward process compared to OTTT.

As a result, it becomes possible to compute $\boldsymbol{u}^{l+1}[t]$ and $\boldsymbol{s}^{l+1}[t]$ during the process of obtaining $\widehat{\boldsymbol{U}}^{l+1}[t]$ and $\widehat{\boldsymbol{a}}^{l+1}[t]$. Because it is possible to compute $\widehat{\boldsymbol{U}}^{l+1}[t]$ and $\widehat{\boldsymbol{a}}^{l+1}[t]$ from $\boldsymbol{u}^{l+1}[t]$ and $\boldsymbol{s}^{l+1}[\tau]$ ($\tau = 1, \ldots, t$), the forward processes of SAF and SNN composed of LIF neurons are mutually convertible. Additionally, because the IF neuron is a special case of an LIF neuron (i.e., $\lambda = 1$), the forward processes of SAF, when $\lambda = 1$, and SNN composed of IF neurons are mutually convertible. Furthermore, in OTTT, both $\boldsymbol{s}[t]$ and $\widehat{\boldsymbol{a}}[t]$ need to be propagated during the forward process for efficient GPU computation, whereas in SAF, only $\widehat{\boldsymbol{a}}[t]$ needs to be propagated (see Fig. 1). Therefore, SAF can reduce the computation time during training.

Backward process

As with OTTT, SAF can be trained in two different ways. The first method updates the parameters by computing the gradient at each time step. We call this *SAF-E*. Let $L_E[t] = \mathcal{L}(\boldsymbol{s}^N[t], \boldsymbol{y})/T$ be the loss function. Assuming that $L_E[t]$ depends only on $\widehat{\boldsymbol{a}}^l[t]$ and $\widehat{\boldsymbol{U}}^l[t]$, i.e., not on anything up to $t-1$, we calculate the derivative based on the definition of forward propagation as

$$\frac{\partial L_E[t]}{\partial \boldsymbol{W}^l} = \widehat{\boldsymbol{a}}^l[t] \frac{\partial L_E[t]}{\partial \widehat{\boldsymbol{a}}^N[t]} \left(\prod_{i=N-1}^{l+1} \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[t]}{\partial \widehat{\boldsymbol{a}}^i[t]} \right) \frac{\partial \widehat{\boldsymbol{a}}^{l+1}[t]}{\partial \widehat{\boldsymbol{U}}^{l+1}[t]}. \tag{6}$$

Note that $\partial \widehat{\boldsymbol{a}} / \partial \widehat{\boldsymbol{U}}$ is non-differentiable; we approximate it with SG. Detailed calculations are given in Appendix B.2. Here, we set

$$\boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{l+1}[t] = \frac{\partial L_E[t]}{\partial \widehat{\boldsymbol{a}}^N[t]} \left(\prod_{i=N-1}^{l+1} \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[t]}{\partial \widehat{\boldsymbol{a}}^i[t]} \right) \frac{\partial \widehat{\boldsymbol{a}}^{l+1}[t]}{\partial \widehat{\boldsymbol{U}}^{l+1}[t]}.$$

Then, it holds that

$$\frac{\partial L_E[t]}{\partial \boldsymbol{W}^l} = \widehat{\boldsymbol{a}}^l[t] \, \boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{l+1}[t]. \tag{7}$$

The second method calculates the gradient only at the final time step and updates the parameters. We call this *SAF-F*. Let $L_F = \mathcal{L}(\sum_{t=0}^{T} \lambda^{T-t} \boldsymbol{s}^N[t] / \sum_{t=0}^{T} \lambda^{T-t}, \boldsymbol{y})$ be a loss function. As with SAF-E, suppose that L_F depends only on $\widehat{\boldsymbol{a}}^l[T]$ and $\widehat{\boldsymbol{U}}^l[T]$. Simply replacing t with T and L_E with L_F in the above calculation, we obtain

$$\frac{\partial L_F}{\partial \boldsymbol{W}^l} = \widehat{\boldsymbol{a}}^l[T] \, \boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{l+1}[T]. \tag{8}$$

4.2 Equivalence with OTTT_{O} and Spike Representation

In this subsection, we show that SAF-E is equivalent to OTTT_{O} and SAF-F is equivalent to Spike Representation, i.e., the forward and backward processes are consistent, respectively. This means that we can train SNNs by SAF and infer by LIF neurons.

Equivalence with OTTT_{O}

We will transform the gradient of SAF-E to be consistent with that of OTTT_{O} when the loss function is $L_E[t]$. Because $L_E[t]$ does not include any argument before t, we obtain

$$\frac{\partial L_E[t]}{\partial \widehat{\boldsymbol{a}}^N[t]} = \frac{\partial L_E[t]}{\partial \boldsymbol{s}^N[t]}. \tag{9}$$

The following two equations hold from the forward processes of SAF and OTTT:

$$\frac{\partial \widehat{\boldsymbol{a}}^{i+1}[t]}{\partial \widehat{\boldsymbol{a}}^i[t]} = \frac{\partial \boldsymbol{s}^{i+1}[t]}{\partial \boldsymbol{s}^i[t]}, \quad \frac{\partial \widehat{\boldsymbol{a}}^{l+1}[t]}{\partial \widehat{\boldsymbol{U}}^{l+1}[t]} = \frac{\partial \boldsymbol{s}^{l+1}[t]}{\partial \boldsymbol{u}^{l+1}[t]}. \tag{10}$$

By substituting (9), (10) for $\boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{l+1}$ in (7), we have

$$\boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{l+1}[t] = \frac{\partial L_E[t]}{\partial \boldsymbol{s}^N[t]} \left(\prod_{i=N-1}^{l+1} \frac{\partial \boldsymbol{s}^{i+1}[t]}{\partial \boldsymbol{s}^i[t]} \right) \frac{\partial \boldsymbol{s}^{l+1}[t]}{\partial \boldsymbol{u}^{l+1}[t]}.$$

Hence, the following theorem holds from (3).

Theorem 1. *The backward processes of SAF-E and OTTT$_{\mathrm{O}}$ are identical, that is,* $\dfrac{\partial L_E[t]}{\partial \boldsymbol{W}^l} = \left(\dfrac{\partial L_E[t]}{\partial \boldsymbol{W}^l}\right)_{\mathrm{OT}}$.

A detailed proof is given in Appendix B.3. Because we have already confirmed that the forward process is consistent, SAF-E and OTTT$_{\mathrm{O}}$ are equivalent.

Equivalence with Spike Representation

Now, we show that SAF-F is equivalent to Spike Representation. Setting the loss function of Spike Representation as L_F, from the expression (2), we have

$$\left(\frac{\partial L_F}{\partial \boldsymbol{W}^l}\right)_{\mathrm{SR}} = \frac{\widehat{\boldsymbol{a}}^l[T]}{V_{\mathrm{th}}}\left(\frac{\partial L}{\partial \widehat{\boldsymbol{a}}^N[T]}\left(\prod_{i=N-1}^{l+1}\frac{\partial \widehat{\boldsymbol{a}}^{i+1}[T]}{\partial \widehat{\boldsymbol{a}}^i[T]}\right)\odot \boldsymbol{d}^{l+1}[T]^\top\right),$$

where $\boldsymbol{d}^{l+1}[T] = \sigma'\left((\boldsymbol{W}^l\widehat{\boldsymbol{a}}^l[T]/\Lambda + \boldsymbol{b}^{l+1})/V_{\mathrm{th}}\right)$, $\Lambda = \sum_{\tau=0}^{T}\lambda^{T-\tau}$, and $\odot$ is the element-wise product. Now we assume that

$$\frac{\partial \boldsymbol{s}^{l+1}[T]}{\partial \boldsymbol{u}^{l+1}[T]} = \mathrm{diag}(\boldsymbol{d}^{l+1}[T]),$$

for any $l = 0, \ldots, N-1$, where $\mathrm{diag}(\boldsymbol{d}^{l+1}[T])$ is a diagonal matrix constructed from $\boldsymbol{d}^{l+1}[T]$. The reason why this assumption is valid discussed in Xiao et al. (2022). Then, we obtain

$$\left(\frac{\partial L_F}{\partial \boldsymbol{W}^l}\right)_{\mathrm{SR}} = \frac{1}{V_{\mathrm{th}}}\widehat{\boldsymbol{a}}^l[T]\,\boldsymbol{g}^{l+1}_{\widehat{\boldsymbol{U}}}[T].$$

Hence, the following theorem holds.

Theorem 2. *Suppose that $\boldsymbol{m}[t]$ converges when $t \to \infty$. Then, for sufficiently large T, the backward processes of SAF-F and Spike Representation are identical up to a scale factor, that is,* $\dfrac{\partial L_F}{\partial \boldsymbol{W}^l} = V_{\mathrm{th}}\left(\dfrac{\partial L_F}{\partial \boldsymbol{W}^l}\right)_{\mathrm{SR}}$.

See Appendix B.4 for the complete proof. Because we have already confirmed that the forward process is consistent, SAF-F and Spike Representation are equivalent.

4.3 Feedforward and Feedback Connection

In the brain, there are not only layer stacking, as in normal SNNs, but also feedforward and feedback connections (Semedo et al., 2022). Therefore, it is important to consider SAF-E/F including these connections. In this subsection, we discuss the relationship between OTTT$_{\mathrm{O}}$, SR, and SAF-E/F in the context of both feedforward and feedback connections.

Feedforward Connection

To begin, we consider the SNN composed of LIF neurons. The forward process of the $(q+1)$-th layer of the SNN with a feedforward connection from the p-th layer to the $(q+1)$-th layer (where $q \geq p$) with weight $\boldsymbol{W}_f$ is as follows:

$$\begin{cases} \boldsymbol{u}^{q+1}[t] = \lambda(\boldsymbol{u}^{q+1}[t-1] - V_{\mathrm{th}}\,\boldsymbol{s}^{q+1}[t-1]) + \boldsymbol{W}^q\boldsymbol{s}^q[t] + \boldsymbol{b}^{q+1} + \boldsymbol{W}_f\boldsymbol{s}^p[t], \\ \boldsymbol{s}^{q+1}[t] = H(\boldsymbol{u}^{q+1}[t] - V_{\mathrm{th}}). \end{cases} \tag{11}$$

Note that the layers other than the $(q+1)$-th layer are the same as in (1). Meanwhile, the forward process of the $(q+1)$-th layer of the SAF is as follows:

$$\begin{cases} \widehat{\boldsymbol{U}}^{q+1}[t] = \boldsymbol{W}^q\widehat{\boldsymbol{a}}^q[t] + \boldsymbol{b}^{q+1}(\sum_{\tau=0}^{t-1}\lambda^\tau) + \lambda^t\widehat{\boldsymbol{U}}^{q+1}[0] + \boldsymbol{W}_f\,\widehat{\boldsymbol{a}}^p[t], \\ \widehat{\boldsymbol{a}}^{q+1}[t] = \lambda\widehat{\boldsymbol{a}}^{q+1}[t-1] + H(\widehat{\boldsymbol{U}}^{q+1} - V_{\mathrm{th}}(\lambda\widehat{\boldsymbol{a}}^{q+1}[t-1]+1)). \end{cases} \tag{12}$$

The layers other than the $(q+1)$-th layer are the same as in (4). The forward processes of SAF and LIF with feedforward connection are mutually convertible.
Regarding the backward processes, the gradients for parameters other than $\boldsymbol{W}_f$ are the same as when there is no feedforward connection. The derivative with respect to $\boldsymbol{W}_f$ is calculated as

$$\frac{\partial L_E[t]}{\partial \boldsymbol{W}_f} = \widehat{\boldsymbol{a}}^p[t] \frac{\partial L_E[t]}{\partial \widehat{\boldsymbol{a}}^N[t]} \left(\prod_{i=N-1}^{q+1} \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[t]}{\partial \widehat{\boldsymbol{a}}^i[t]} \right) \frac{\partial \widehat{\boldsymbol{a}}^{q+1}[t]}{\partial \widehat{\boldsymbol{U}}^{q+1}[t]}.$$

Therefore, $\partial L_E[t]/\partial \boldsymbol{W}_f = \widehat{\boldsymbol{a}}^p[t]\, \boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{q+1}[t]$ for SAF-E and $\partial L_F/\partial \boldsymbol{W}_f = \widehat{\boldsymbol{a}}^p[T]\, \boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{q+1}[T]$ for SAF-F.

Feedback Connection

The forward process of the $(q+1)$-th layer of the SNN with a feedback connection from the p-th layer to the $(q+1)$-th layer (where $q < p$) with weight $\boldsymbol{W}_b$ is as follows:

$$\begin{cases} \boldsymbol{u}^{q+1}[t] = \lambda(\boldsymbol{u}^{q+1}[t-1] - V_{\mathrm{th}}\, \boldsymbol{s}^{q+1}[t-1]) + \boldsymbol{W}^q \boldsymbol{s}^q[t] + \boldsymbol{b}^{q+1} + \boldsymbol{W}_b\, \boldsymbol{s}^p[t-1], \\ \boldsymbol{s}^{q+1}[t] = H(\boldsymbol{u}^{q+1}[t] - V_{\mathrm{th}}). \end{cases} \tag{13}$$

Note that the layers other than the $(q+1)$-th layer are the same as in (1). Meanwhile, the forward process of the $(q+1)$-th layer of SAF is as follows:

$$\begin{cases} \widehat{\boldsymbol{U}}^{q+1}[t] = \boldsymbol{W}^q \widehat{\boldsymbol{a}}^q[t] + \boldsymbol{b}^{q+1}(\sum_{\tau=0}^{t-1} \lambda^\tau) + \lambda^t \widehat{\boldsymbol{U}}^{q+1}[0] + \boldsymbol{W}_b\, \widehat{\boldsymbol{a}}^p[t-1], \\ \widehat{\boldsymbol{a}}^{q+1}[t] = \lambda \widehat{\boldsymbol{a}}^{q+1}[t-1] + H(\widehat{\boldsymbol{U}}^{q+1} - V_{\mathrm{th}}(\lambda \widehat{\boldsymbol{a}}^{q+1}[t-1] + 1)). \end{cases} \tag{14}$$

The layers other than the $(q+1)$-th layer are the same as in (4). The forward processes of SAF and LIF with feedback connection are mutually convertible.
Regarding the backward processes, the gradients for parameters other than $\boldsymbol{W}_b$ are the same as when there is no feedback connection. The derivative with respect to $\boldsymbol{W}_b$ is calculated as

$$\frac{\partial L_E[t]}{\partial \boldsymbol{W}_b} = \widehat{\boldsymbol{a}}^p[t-1] \frac{\partial L_E[t]}{\partial \widehat{\boldsymbol{a}}^N[t]} \left(\prod_{i=N-1}^{q+1} \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[t]}{\partial \widehat{\boldsymbol{a}}^i[t]} \right) \frac{\partial \widehat{\boldsymbol{a}}^{q+1}[t]}{\partial \widehat{\boldsymbol{U}}^{q+1}[t]}.$$

Therefore, we obtain $\partial L_E[t]/\partial \boldsymbol{W}_b = \widehat{\boldsymbol{a}}^p[t-1]\, \boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{q+1}[t]$ for SAF-E and $\partial L_F/\partial \boldsymbol{W}_b = \widehat{\boldsymbol{a}}^p[T-1]\, \boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{q+1}[T]$ for SAF-F.

Equivalence with $\mathrm{OTTT_O}$ and Spike Representation

We can show the equivalence of SAF-E and $\mathrm{OTTT_O}$ with a feedforward connection, or with a feedback connection, as well as in Theorem 1. Moreover, as mentioned in Sec. 3.2, Spike Representation computes the gradients as (2), then the equivalence of SAF-F and Spike Representation also holds, even with a feedforward connection.

Corollary 3. *For SNN with a feedforward connection* (11)*, or a feedback connection* (13)*, the following hold.*

(i) *The backward processes of SAF-E and $\mathrm{OTTT_O}$ with a feedforward connection are identical.*

(ii) *Suppose that $\boldsymbol{m}[t]$ converges when $t \to \infty$. Then, for sufficiently large T, the backward processes of SAF-F and Spike Representation with a feedforward connection are identical up to a scale factor.*

(iii) *The backward processes of SAF-E and $\mathrm{OTTT_O}$ with a feedback connection are identical.*

The proof is stated in Appendix B.5.

Algorithm 1 One iteration of SAF training.

Require: Network parameters $\{\boldsymbol{W}^l\}$, $\{\boldsymbol{b}^{l+1}\}$; Time steps T; Number of layers N; Other hyperparameters; Input dataset
Ensure: Trained network parameters $\{\boldsymbol{W}^l\}$, $\{\boldsymbol{b}^{l+1}\}$

```
for t = 1, 2, ..., T do
    % Forward
    for l = 1, 2, ..., N do
        Update the (weighted) potential accumulation Û^l[t] and spike accumulation â^l[t] using (4).
    end for
    % Backward
    for l = N, N − 1, ..., 1 do
        if training option is SAF-E then
            Update parameters with ∂L_E[t]/∂W^l = â^l[t] g^{l+1}_Û[t] based on the gradient-based optimizer.
        else if training option is SAF-F and t = T then
            Update parameters with ∂L_F/∂W^l = â^l[T] g^{l+1}_Û[T] based on the gradient-based optimizer.
        end if
    end for
end for
```

Proximity to Spike Representation

Assume that the SNN have a feedback connection same as in (13). Then, the same assertion as Theorem 2 does not hold, that is, SAF-F is not equivalent to Spike Representation in general. However, we can show that the gradient descent directions in SAF-F and Spike Representation are close.

Theorem 4. *Suppose that $\boldsymbol{m}[t]$ converges when $t \to \infty$. Then, for sufficiently large T, the backward processes of SAF-F and Spike Representation with a feedback connection are similar, that is, $\left\langle \frac{\partial L_F}{\partial \boldsymbol{\theta}}, \left(\frac{\partial L_F}{\partial \boldsymbol{\theta}}\right)_{\mathrm{SR}} \right\rangle > 0$ for all parameters $\boldsymbol{\theta}$.*

See Appendix B.6 for the proof.

5 Experiments

In Sec. 4.2, we theoretically proved that SAF-E and $\mathrm{OTTT_O}$ as well as SAF-F and spike representation are equivalent. In this section, we experimentally compare these methods. As complex and large datasets make it difficult to analyze the results, we trained SAF on the CIFAR-10 and the CIFAR-100 datasets (Krizhevsky and Hinton, 2009) and inferred with SNN composed of LIF neurons. This experiment was performed five times with different initial parameters, and all approximation was executed by the sigmoid-like SG for fair comparison. We used the same experimental setup as (Xiao et al., 2022), including the choice of SG. The code was written in PyTorch (Paszke et al., 2019), and the experiments were executed using one GPU, an NVIDIA Tesla V100, 32GB. We show the pseudo-code of SAF-E and SAF-F in Algorithm 1 to better understand our methods. The implementation details are in Appendix C. The main objective here is to analyze whether there are any inconsistencies between theory and experiment rather than to achieve state-of-the-art performance.

5.1 Analysis of SAF-E

We experimentally analyze the performance of SAF-E. First, we compare accuracy. Table 1 shows the accuracy when we set $T = 6$ (the difference of gradients are shown in App. D). As shown in this table, SAF-E and $\mathrm{OTTT_O}$ have almost the same accuracy. The values in parentheses in Table 1 show the change in accuracy due to inference by an SNN composed of LIF neurons. From this table, we can see that the accuracy

change due to inference by SNN consisting of LIF neurons is almost negligible in the case of CIFAR-10. On the other hand, CIFAR-100 shows a minor difference in accuracy (compared to the difference in gradient). This could be attributed to the increased task complexity, resulting in a more intricate loss function and greater susceptibility to even minor numerical errors affecting final accuracy.

Figure 2 shows the accuracy and loss curves during the training of CIFAR-10. This indicates that the progress during training are comparable. Therefore, we confirmed experimentally that SAF-E and $\mathrm{OTTT_O}$ are numerically close. Similar results were obtained when there was a feedforward or a feedback connection (see Appendix E).

Next, we compare the training costs. From Table 1, it can be seen that SAF-E takes less time to train and uses less memory during training than $\mathrm{OTTT_O}$. However, $\mathrm{OTTT_O}$ can be executed with constant memory usage even as time steps increase. Therefore, we compared the training time and memory usage at different time steps. Figures 3 (A) and (B) show the training time and memory at different time steps. Note that the training time was measured in one batch. It can be seen that the memory usage of SAF-E does not increase even if the number of time steps increases, similar to $\mathrm{OTTT_O}$. Also, from Fig. 3 (B), we can see that SAF-E uses less memory than $\mathrm{OTTT_O}$. This result stems from the fact that SAF does not need to maintain the previous membrane potential.

Finally, we compare the firing rate. As shown in Table 1, the total firing rates of SAF-E and $\mathrm{OTTT_O}$ are close. Also, the amount of change due to inference with SNNs consisting of LIF neurons is also almost negligible, similar to the case for accuracy. Furthermore, from Fig. 3 (C), the firing rates of each layer are almost close too.

These results indicate that using SAF-E can reduce the training time and memory compared to $\mathrm{OTTT_O}$ while achieving close firing rate and accuracy. It was also shown that using the parameters trained with SAF-E to infer with the SNN consisting of LIF neurons is feasible.

5.2 Analysis of SAF-F

In this subsection, we experimentally analyze the performance of SAF-F. First, we compare accuracy. The results are shown in Table 2 and Fig. 4 (the difference of gradients are shown in App. D). Since the spike representation method is effective when T is large, there is no theoretical guarantee that SAF-F can infer well in a short time step. However, Table 2 shows that the accuracy of SAF-F is almost the same for $T = 6$ and $T = 32$. Note that Spike Representation methods, except for SAF-F, do not use SGs, which makes precise comparisons difficult. Therefore, we only compared SAF-F with $\mathrm{OTTT_A}$. As with the previous results, the accuracy change due to inference with SNNs consisting of LIF neurons is almost negligible. Meanwhile, the accuracies of SAF-F and $\mathrm{OTTT_A}$ are close, though from the perspective of standard deviation, there seems to be a difference. From Sec. 4.2, the gradient directions of Spike Representation and SAF-F are identical, but those of Spike Representation and $\mathrm{OTTT_A}$ are only similar. Therefore, the gradient directions of SAF-F and $\mathrm{OTTT_A}$ are also only similar. This is thought to be the cause of the differences in accuracy and loss.

Next, we compare the training costs. From Table 2, it can be seen that SAF-F requires less time for training and uses less memory than $\mathrm{OTTT_A}$. This trend is also similar when the time step is varied (see Figs. 5 (A) and (B)).

Finally, we compare firing rate. As shown in Table 2 as for accuracy, the change of the total firing rate by inferring with SNNs consisting of LIF neurons is almost negligible. Meanwhile, the total firing rate of SAF-F is smaller than of $\mathrm{OTTT_A}$. In addition, from Fig. 5 (C), it can be seen that the firing rate of each layer (especially the first layer) is smaller in SAF-F than $\mathrm{OTTT_A}$. These differences also indicate that SAF-F and $\mathrm{OTTT_A}$ are generally not identical.

From the above analysis, we can say that SAF-F is a better choice than $\mathrm{OTTT_A}$ in terms of training time, memory usage, and firing rate. Also, as with SAF-E, inference can be performed by the standard SNN using the parameters trained by SAF-F.

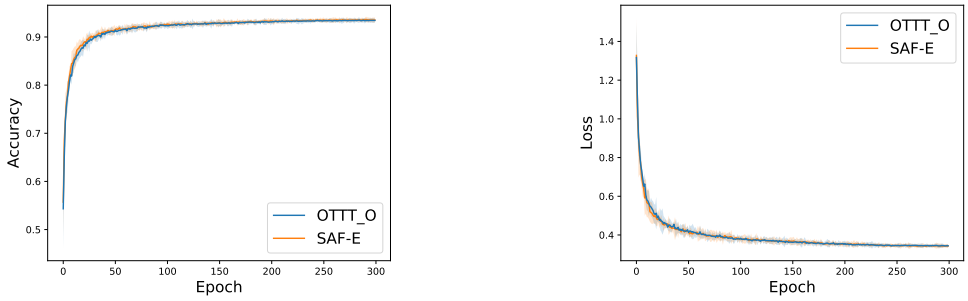
Table 1: Performance comparison of SAF-E and OTTT_O on CIFAR-10 and CIFAR-100. The values in parentheses were the changes in accuracy and total firing rate due to inference by the SNN composed of LIF neurons. Note that training times were measured in one minibatch, and training time and memory were not perturbed between trials.

CIFAR-10

Method	T	**Memory [GB]**	**Training Time [sec]**	**Firing rate [%]**	**Accuracy [%]**
OTTT_O	6	1.656	0.666	15.14±0.17	93.44±0.15
SAF-E	6	1.184	0.468	14.76±0.15 (1.048×10^{-5})	93.54±0.17 (0.016)

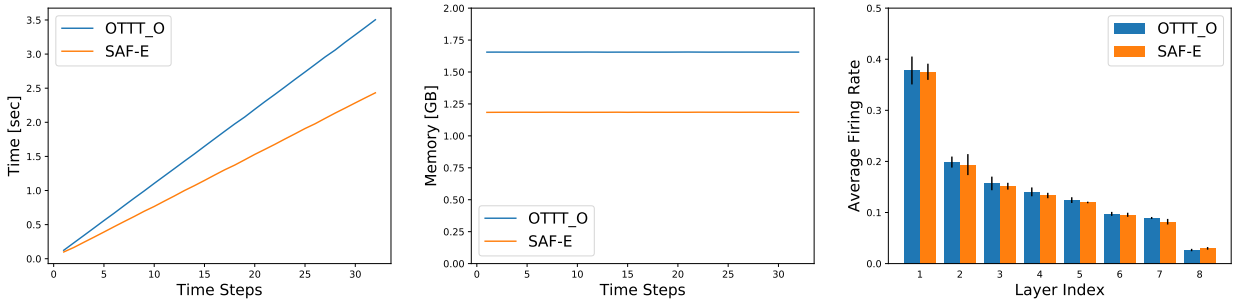
CIFAR-100

Method	T	**Memory [GB]**	**Training Time [sec]**	**Firing rate [%]**	**Accuracy [%]**
OTTT_O	6	1.656	0.666	17.27±0.19	70.70±0.19
SAF-E	6	1.186	0.464	16.77±0.08 (1.513×10^{-5})	71.56±0.35 (0.042)



(A) Accuracy (B) Loss

Figure 2: Accuracy and loss curves of SAF-E and OTTT_O on CIFAR-10 ($T = 6$).



(A) Training Time (B) Memory Consumption (C) Firing rate

Figure 3: Training time, memory consumption, and firing rate of each layer of SAF-E and OTTT_O on CIFAR-10.

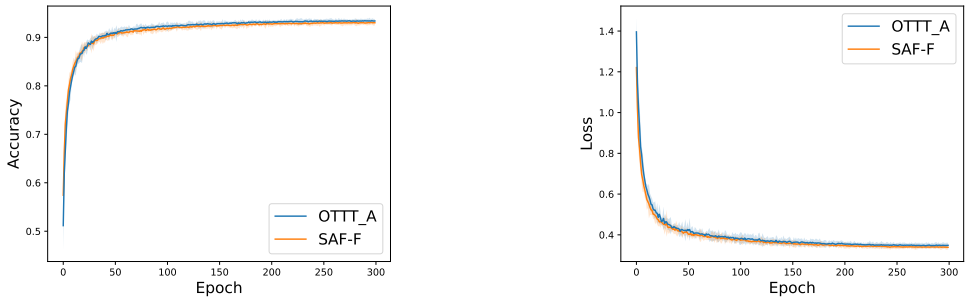
Table 2: Performance comparison of SAF-F and OTTT_A on CIFAR-10 and CIFAR-100. The values in parentheses are the changes in accuracy and total firing rate due to inference by the SNN composed of LIF neurons. Note that training times were measured in one minibatch, and training time and memory were not perturbed between trials.

CIFAR-10

Method	T	**Memory [GB]**	**Training Time [sec]**	**Firing rate [%]**	**Accuracy [%]**
OTTT_A	6	1.656	0.661	15.51±0.10	93.39±0.16
OTTT_A	32	1.656	3.474	13.96±0.20	93.62±0.04
SAF-F	6	1.157	0.247	10.50±0.19 (3.306×10^{-5})	93.09±0.15 (0.076)
SAF-F	32	1.157	1.077	10.65±0.12 (0.965×10^{-5})	93.25±0.07 (0.002)

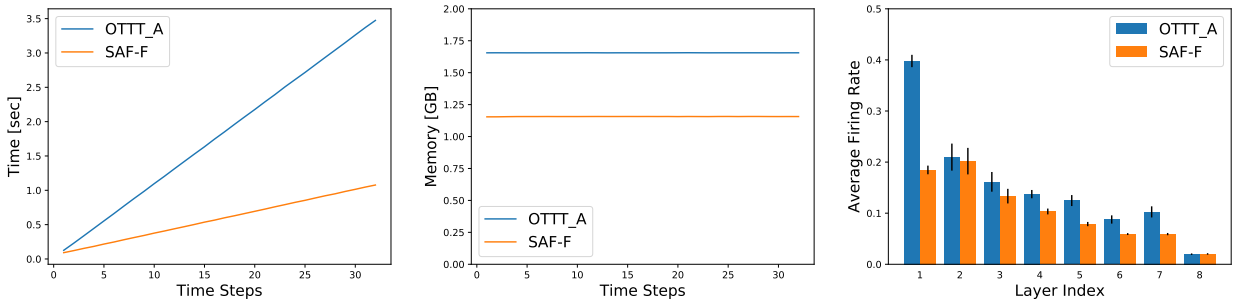
CIFAR-100

Method	T	**Memory [GB]**	**Training Time [sec]**	**Firing rate [%]**	**Accuracy [%]**
OTTT_A	6	1.656	0.666	18.26±0.21	70.18±0.26
OTTT_A	32	1.656	3.479	16.62±0.20	70.77±0.16
SAF-F	6	1.157	0.249	12.19±0.22 (1.106×10^{-5})	70.58±0.34 (0.002)
SAF-F	32	1.157	1.077	12.21±0.15 (1.016×10^{-5})	71.73±0.04 (0.096)



(A) Accuracy (B) Loss

Figure 4: Accuracy and loss curves of SAF-F and OTTT_A on CIFAR-10 ($T = 6$).



(A) Training Time (B) Memory Consumption (C) Firing rate

Figure 5: Training time, memory consumption, and firing rate of each layer of SAF-F and OTTT_A on CIFAR-10. It can be seen that the firing rates do not match, which agree with the theoretical result that SAF-F is not always consistent with OTTT_A.

6 Limitation and Discussion

In this paper, we have shown that SAF-E coincides with $\mathrm{OTTT_O}$ and that SAF-F coincides with $\mathrm{OTTT_A}$. On the other hand, Xiao et al. (2022) shows that OTTT methods are more accurate than BPTT. Given the concordance between SAF and OTTT, SAF is more accurate than BPTT. We consider that the better accuracy than BPTT is due to the difficulty of achieving optimal rollout at all times in BPTT, as known by vanishing gradients, and the fact that most of the datasets used in the field of deep SNNs are time-independent labels. Therefore, we assume that $\partial \widehat{\boldsymbol{a}}^N[t-1]/\partial \mathbf{W}^l$ is zero (see App. B.2). This assumption is intended to even out the gradient's effect on parameter updates at each time, as described in App. B.2, and is also valid for widely used time-independent datasets. On the other hand, since SAF trains using information up to t by the spike accumulation $\widehat{\boldsymbol{a}}$, not just the current time, SAF can implicitly train at each time while using information up to the previous time. The above assumption can be easily removed; however, its validation would require a labeled time-dependent data set. This would require the preparation of an appropriate data set, which is outside the scope of this paper, considering theoretical consistency with OTTT and is therefore considered one for future research.

7 Conclusion and Future Work

This article proposed SAF. SAF is a training method of SNNs that propagates the spike accumulation during training; however, SAF propagates the spike trains during inference, as do other SNNs. This article showed that SAF trained at each time step (SAF-E) is equivalent to $\mathrm{OTTT_O}$, and SAF trained at the final time step (SAF-F) is also equivalent to Spike Representation. We conducted experiments on the CIFAR-10 dataset and confirmed that the experimental results are consistent with these assertions and that training time and memory of SAF are reduced compared to OTTT.

Most SNNs, including OTTT, Spike Representation, and SAF, train under time-independent labeled training data. Then, the issue of improving SAF to train even if the labels change at each time step remains for future work. Other remaining tasks are to experimentally confirm the similar results of this work for other surrogate gradients and to investigate the theoretical relationship with other learning rules such as SuperSpike.

SAF, as presented in this article, assumes training on a GPU. Therefore, it may not be suitable for training on neuromorphic chips. However, we believe executing training on GPUs and inference on neuromorphic chips is reasonable. In the future, we also plan to verify inference on the neuromorphic chip, including the extent to which numerical errors are affected by differences in computing environments.

References

Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, et al. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. IEEE transactions on computer-aided design of integrated circuits and systems, 34(10):1537–1557, 2015.

Yoshua Bengio, Thomas Mesnard, Asja Fischer, Saizheng Zhang, and Yuhuai Wu. Stdp as presynaptic activity times rate of change of postsynaptic activity. arXiv preprint arXiv:1509.05936, 2015.

Guo-qiang Bi and Mu-ming Poo. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. Journal of neuroscience, 18(24):10464–10472, 1998.

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

Sayeed Shafayet Chowdhury, Nitin Rathi, and Kaushik Roy. One timestep is all you need: Training spiking neural networks with ultra low latency. arXiv preprint arXiv:2110.05929, 2021.

Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. Ieee Micro, 38(1):82–99, 2018.

Shikuang Deng and Shi Gu. Optimal conversion of conventional artificial neural networks to spiking neural networks. In International Conference on Learning Representations, 2020.

Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael 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.

Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 13728 – 13737, 2021.

Nicolas Frémaux and Wulfram Gerstner. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. Frontiers in neural circuits, 9:85, 2016.

Samy Wu Fung, Howard Heaton, Qiuwei Li, Daniel McKenzie, Stanley Osher, and Wotao Yin. Jfb:jacobian-free backpropagation for implicit networks. In Proceedings of the AAAI Conference on Artificial Intelligence, 2022.

Zhengyang Geng, Xin-Yu Zhang, Shaojie Bai, Yisen Wang, and Zhouchen Lin. On training implicit models. In Advances in Neural Information Procesing Systems, 2021.

Bing Han, Gopalakrishnan Srinivasan, and Kaushik Roy. Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 13558–13567, 2020.

DO Hebb. The Organization of Behavior: A Neuropsychological Theory. Psychology Press, 2005.

Jacques Kaiser, Hesham Mostafa, and Emre Neftci. Synaptic plasticity dynamics for deep continuous local learning (decolle). Frontiers in Neuroscience, 14:515306, 2020.

Seijoon Kim, Seongsik Park, Byunggook Na, and Sungroh Yoon. Spiking-yolo: spiking neural network for energy-efficient object detection. In Proceedings of the AAAI conference on artificial intelligence, volume 34, pages 11270–11277, 2020.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, 2009. URL `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

Louis Lapique. Researches quantatives sur l'excitation electrique des nerfs traitee comme une polarization. Journal of Physiology, Pathology and Genetics, 9:620–635, 1907.

Shuang Lian, Jiangrong Shen, Qianhui Liu, Ziming Wang, Rui Yan, and Huajin Tang. Learnable surrogate gradient for direct training spiking neural networks. In Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23, pages 3002–3010, 8 2023.

Tao Luo, Weng-Fai Wong, Rick Siow Mong Goh, Anh Tuan Do, Zhixian Chen, Haizhou Li, Wenyu Jiang, and Weiyun Yau. Achieving green ai with energy-efficient deep learning using neuromorphic computing. Communications of the ACM, 66(7):52–57, 2023.

Qingyan Meng, Mingqing Xiao, Shen Yan, Yisen Wang, Zhouchen Lin, and Zhi-Quan Luo. Training high-performance low-latency spiking neural networks by differentiation on spike representation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 12444–12453, 2022.

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

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 32, 2019.

Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. Micro-batch training with batch-channel normalization and weight standardization. arXiv preprint arXiv:1903.10520, 2019.

Jinye Qu, Zeyu Gao, Tielin Zhang, Yanfeng Lu, Huajin Tang, and Hong Qiao. Spiking neural network for ultra-low-latency and high-accurate object detection. arXiv preprint arXiv:2306.12010, 2023.

João D Semedo, Anna I Jasper, Amin Zandvakili, Aravind Krishna, Amir Aschner, Christian K Machens, Adam Kohn, and Byron M Yu. Feedforward and feedback interactions between visual cortical areas use different population activity patterns. Nature communications, 13(1):1099, 2022.

Sumit B Shrestha and Garrick Orchard. Slayer: Spike layer error reassignment in time. Advances in neural information processing systems, 31, 2018.

Richard B Stein. A theoretical analysis of neuronal variability. Biophysical Journal, 5(2):173–194, 1965.

Kazuma Suetake, Shin ichi Ikegawa, Ryuji Saiin, and Yoshihide Sawada. S^3NN: Time step reduction of spiking surrogate gradients for training energy efficient single-step spiking neural networks. Neural Networks, 159: 208–219, 2023.

Johannes C Thiele, Olivier Bichler, and Antoine Dupret. Spikegrad: An ann-equivalent computation model for implementing backpropagation with spikes. In International Conference on Learning Representations, 2019.

Jibin Wu, Yansong Chua, Malu Zhang, Guoqi Li, Haizhou Li, and Kay Chen Tan. A tandem learning rule for effective training and rapid inference of deep spiking neural networks. IEEE Transactions on Neural Networks and Learning Systems, 2021.

Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. Frontiers in neuroscience, 12:331, 2018.

Mingqing Xiao, Qingyan Meng, Zongpeng Zhang, Yisen Wang, and Zhouchen Lin. Training Feedback Spiking Neural Networks by Implicit Differentiation on the Equilibrium State. Advances in Neural Information Processing Systems, 18(NeurIPS):14516–14528, 2021.

Mingqing Xiao, Qingyan Meng, Zongpeng Zhang, Di He, and Zhouchen Lin. Online training through time for spiking neural networks. Advances in Neural Information Processing Systems, 35:20717–20730, 2022.

Mingqing Xiao, Qingyan Meng, Zongpeng Zhang, Yisen Wang, and Zhouchen Lin. SPIDE: A purely spike-based method for training feedback spiking neural networks. Neural Networks, 161:9–24, 2023.

Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. Neural computation, 30(6):1514–1541, 2018.

Hanle Zheng, Yujie Wu, Lei Deng, Yifan Hu, and Guoqi Li. Going deeper with directly-trained larger spiking neural networks. In Proceedings of the AAAI conference on artificial intelligence, volume 35, pages 11062–11070, 2021.

Shibo Zhou, Xiaohua Li, Ying Chen, Sanjeev T Chandrasekaran, and Arindam Sanyal. Temporal-coded deep spiking neural network with easy training and robust performance. In Proceedings of the AAAI conference on artificial intelligence, volume 35, pages 11143–11151, 2021.

Appendix

A List of main Formulas

The neurons and gradients are as follows. Note that $\boldsymbol{s}^l[t]$, $\boldsymbol{u}^l[t]$, $\boldsymbol{W}^l$ and $\boldsymbol{b}^l$ are the spike train, membrane potential, weight, and bias of l-th (also denoted by p-th and q-th) layer, respectively. Also, $\lambda \leq 1$ is the leaky term, V_{th} is the threshold, L, L_E, and L_F are the loss functions, H is the element-wise Heaviside step function, N is the number of layers, $\boldsymbol{a}[t] = \sum_{\tau=0}^{t} \lambda^{t-\tau} \boldsymbol{s}[\tau] / \sum_{\tau=0}^{t} \lambda^{t-\tau}$ is the weighted firing rate, $\widehat{\boldsymbol{a}}[t] = \sum_{\tau=0}^{t} \lambda^{t-\tau} \boldsymbol{s}[\tau]$ is the (weighted) spike accumulation, and $\widehat{\boldsymbol{U}}^{l+1}[t] = \lambda \widehat{\boldsymbol{U}}^{l+1}[t-1] + \boldsymbol{W}^l(\widehat{\boldsymbol{a}}^l[t] - \lambda \widehat{\boldsymbol{a}}^l[t-1]) + \boldsymbol{b}^{l+1}$ is the (weighted) potential accumulation.

A.1 Neurons

LIF neuron (1):

$$\begin{cases} \boldsymbol{u}^{l+1}[t] = \lambda(\boldsymbol{u}^{l+1}[t-1] - V_{\text{th}}\, \boldsymbol{s}^{l+1}[t-1]) + \boldsymbol{W}^l \boldsymbol{s}^l[t] + \boldsymbol{b}^{l+1}, \\ \boldsymbol{s}^{l+1}[t] = H(\boldsymbol{u}^{l+1}[t] - V_{\text{th}}). \end{cases}$$

SAF neuron (4):

$$\begin{cases} \widehat{\boldsymbol{U}}^{l+1}[t] = \boldsymbol{W}^l \widehat{\boldsymbol{a}}^l[t] + \boldsymbol{b}^{l+1} \sum_{\tau=0}^{t-1} \lambda^{t-\tau} + \lambda^t \widehat{\boldsymbol{U}}^{l+1}[0], \\ \widehat{\boldsymbol{a}}^{l+1}[t] = \lambda \widehat{\boldsymbol{a}}^{l+1}[t-1] + H(\widehat{\boldsymbol{U}}^{l+1}[t] - V_{\text{th}}(\lambda \widehat{\boldsymbol{a}}^{l+1}[t-1] + 1)). \end{cases}$$

A.2 Neurons with feedforward connection

LIF neuron (11):

$$\begin{cases} \boldsymbol{u}^{q+1}[t] = \lambda(\boldsymbol{u}^{q+1}[t-1] - V_{\text{th}}\, \boldsymbol{s}^{q+1}[t-1]) + \boldsymbol{W}^q \boldsymbol{s}^q[t] + \boldsymbol{b}^{q+1} + \boldsymbol{W}_f \boldsymbol{s}^p[t], \\ \boldsymbol{s}^{q+1}[t] = H(\boldsymbol{u}^{q+1}[t] - V_{\text{th}}). \end{cases}$$

SAF neuron (12):

$$\begin{cases} \widehat{\boldsymbol{U}}^{q+1}[t] = \boldsymbol{W}^q \widehat{\boldsymbol{a}}^q[t] + \boldsymbol{b}^{q+1}(\sum_{\tau=0}^{t-1} \lambda^\tau) + \lambda^t \widehat{\boldsymbol{U}}^{q+1}[0] + \boldsymbol{W}_f\, \widehat{\boldsymbol{a}}^p[t], \\ \widehat{\boldsymbol{a}}^{q+1}[t] = \lambda \widehat{\boldsymbol{a}}^{q+1}[t-1] + H(\widehat{\boldsymbol{U}}^{q+1} - V_{\text{th}}(\lambda \widehat{\boldsymbol{a}}^{q+1}[t-1] + 1)). \end{cases}$$

A.3 Neurons with feedback connection

LIF neuron (13):

$$\begin{cases} \boldsymbol{u}^{q+1}[t] = \lambda(\boldsymbol{u}^{q+1}[t-1] - V_{\text{th}}\, \boldsymbol{s}^{q+1}[t-1]) + \boldsymbol{W}^q \boldsymbol{s}^q[t] + \boldsymbol{b}^{q+1} + \boldsymbol{W}_b\, \boldsymbol{s}^p[t-1], \\ \boldsymbol{s}^{q+1}[t] = H(\boldsymbol{u}^{q+1}[t] - V_{\text{th}}). \end{cases}$$

SAF neuron (14):

$$\begin{cases} \widehat{\boldsymbol{U}}^{q+1}[t] = \boldsymbol{W}^q \widehat{\boldsymbol{a}}^q[t] + \boldsymbol{b}^{q+1}(\sum_{\tau=0}^{t-1} \lambda^\tau) + \lambda^t \widehat{\boldsymbol{U}}^{q+1}[0] + \boldsymbol{W}_b\, \widehat{\boldsymbol{a}}^p[t-1], \\ \widehat{\boldsymbol{a}}^{q+1}[t] = \lambda \widehat{\boldsymbol{a}}^{q+1}[t-1] + H(\widehat{\boldsymbol{U}}^{q+1} - V_{\text{th}}(\lambda \widehat{\boldsymbol{a}}^{q+1}[t-1] + 1)). \end{cases}$$

A.4 Gradients

Spike Representation (2):

$$\left(\frac{\partial L}{\partial \boldsymbol{W}^l}\right)_{\mathrm{SR}} = \frac{\partial L}{\partial \boldsymbol{a}^N[T]} \left(\prod_{i=N-1}^{l+1} \frac{\partial \boldsymbol{a}^{i+1}[T]}{\partial \boldsymbol{a}^i[T]}\right) \frac{\partial \boldsymbol{a}^{l+1}[T]}{\partial \boldsymbol{W}^l}.$$

$\mathrm{OTTT_O}$ (3):

$$\left(\frac{\partial L[t]}{\partial \boldsymbol{W}^l}\right)_{\mathrm{OT}} = \widehat{\boldsymbol{a}}^l[t] \frac{\partial L[t]}{\partial \boldsymbol{s}^N[t]} \left(\prod_{i=N-1}^{l+1} \frac{\partial \boldsymbol{s}^{i+1}[t]}{\partial \boldsymbol{s}^i[t]}\right) \frac{\partial \boldsymbol{s}^{l+1}[t]}{\partial \boldsymbol{u}^{l+1}[t]}.$$

$\mathrm{OTTT_A}$:

$$\left(\frac{\partial L}{\partial \boldsymbol{W}^l}\right)_{\mathrm{OT}} = \sum_t^T \widehat{\boldsymbol{a}}^l[t] \frac{\partial L[t]}{\partial \boldsymbol{s}^N[t]} \left(\prod_{i=N-1}^{l+1} \frac{\partial \boldsymbol{s}^{i+1}[t]}{\partial \boldsymbol{s}^i[t]}\right) \frac{\partial \boldsymbol{s}^{l+1}[t]}{\partial \boldsymbol{u}^{l+1}[t]}.$$

SAF-E (6), or (7):

$$\frac{\partial L_E[t]}{\partial \boldsymbol{W}^l} = \widehat{\boldsymbol{a}}^l[t] \frac{\partial L_E[t]}{\partial \widehat{\boldsymbol{a}}^N[t]} \left(\prod_{i=N-1}^{l+1} \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[t]}{\partial \widehat{\boldsymbol{a}}^i[t]}\right) \frac{\partial \widehat{\boldsymbol{a}}^{l+1}[t]}{\partial \widehat{\boldsymbol{U}}^{l+1}[t]}.$$

SAF-F (8):

$$\frac{\partial L_F}{\partial \boldsymbol{W}^l} = \widehat{\boldsymbol{a}}^l[T] \frac{\partial L_F}{\partial \widehat{\boldsymbol{a}}^N[T]} \left(\prod_{i=N-1}^{l+1} \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[T]}{\partial \widehat{\boldsymbol{a}}^i[T]}\right) \frac{\partial \widehat{\boldsymbol{a}}^{l+1}[T]}{\partial \widehat{\boldsymbol{U}}^{l+1}[T]}.$$

B Derivation and Proofs

B.1 Derivation of (4) and (5)

In this subsection, we derive (4) and (5) through proving that (4) and (5) hold for any $t \in \{1, \ldots, T\}$ with mathematical induction.

First, we prove that (4) and (5) hold for $t = 1$. We compute $\widehat{\boldsymbol{U}}^{l+1}[1]$ based on definition as follows:

$$\begin{aligned}\widehat{\boldsymbol{U}}^{l+1}[1] &= \lambda \widehat{\boldsymbol{U}}^{l+1}[0] + \boldsymbol{W}^l(\widehat{\boldsymbol{a}}^l[1] - \lambda \widehat{\boldsymbol{a}}^l[0]) + \boldsymbol{b^{l+1}} \\ &= \boldsymbol{W}^l \widehat{\boldsymbol{a}}^l[1] + \boldsymbol{b}^{l+1} + \lambda \widehat{\boldsymbol{U}}^{l+1}[0].\end{aligned}$$

Taking account into $\widehat{\boldsymbol{a}}^{l+1}[0] = \widehat{\boldsymbol{s}}^{l+1}[0] = 0$, we obtain

$$\begin{aligned}\boldsymbol{u}^{l+1}[1] &= \lambda(\boldsymbol{u}^{l+1}[0] - V_{\mathrm{th}}\, \boldsymbol{s}^{l+1}[0]) + \boldsymbol{W}^l \boldsymbol{s}^l[1] + \boldsymbol{b^{l+1}} \\ &= \widehat{\boldsymbol{U}}^{l+1}[0] - V_{\mathrm{th}}\, \widehat{\boldsymbol{a}}^{l+1}[0]) + \boldsymbol{W}^l(\widehat{\boldsymbol{a}}^l[1] - \lambda \widehat{\boldsymbol{a}}^l[0]) + \boldsymbol{b^{l+1}} \\ &= \widehat{\boldsymbol{U}}^{l+1}[1] - V_{\mathrm{th}}\, \widehat{\boldsymbol{a}}^{l+1}[0].\end{aligned}$$

With this equation, we have

$$\begin{aligned}\boldsymbol{s}^{l+1}[1] &= H(\boldsymbol{u}^{l+1}[1] - V_{\mathrm{th}}) \\ &= H(\widehat{\boldsymbol{U}}^{l+1}[1] - V_{\mathrm{th}}(\lambda \widehat{\boldsymbol{a}}^{l+1}[0] + 1)).\end{aligned}$$

Then, $\widehat{\boldsymbol{a}}^{l+1}[1]$ can be computed as follows:

$$\begin{aligned}\widehat{\boldsymbol{a}}^{l+1}[1] &= \sum_{\tau=0}^{1} \lambda^{1-\tau} \boldsymbol{s}^{l+1}[\tau] \\ &= \lambda \widehat{\boldsymbol{a}}^{l+1}[0] + \boldsymbol{s}^{l+1}[1] \\ &= \lambda \widehat{\boldsymbol{a}}^{l+1}[0] + H(\widehat{\boldsymbol{U}}^{l+1}[1] - V_{\mathrm{th}}(\lambda \widehat{\boldsymbol{a}}^{l+1}[0] + 1)).\end{aligned}$$

Therefore, (4) and (5) hold for $t = 1$.

Next, we prove that (4) and (5) hold for t when they hold for any $\tau \in \{1, \ldots, t-1\}$. Assuming that (4) and (5) hold for any $\tau \in \{1, \ldots, t-1\}$, we have

$$\begin{aligned}\widehat{\boldsymbol{U}}^{l+1}[t] &= \lambda \widehat{\boldsymbol{U}}^{l+1}[t-1] + \boldsymbol{W}^l(\widehat{\boldsymbol{a}}^l[t] - \lambda \widehat{\boldsymbol{a}}^l[t-1]) + \boldsymbol{b^{l+1}} \\ &= \lambda \left(\boldsymbol{W}^l \widehat{\boldsymbol{a}}^l[t-1] + \boldsymbol{b}^{l+1} \sum_{\tau=0}^{t-2} \lambda^{t-\tau} + \lambda^{t-1} \widehat{\boldsymbol{U}}^{l+1}[0] \right) + \boldsymbol{W}^l(\widehat{\boldsymbol{a}}^l[t] - \lambda \widehat{\boldsymbol{a}}^l[t-1]) + \boldsymbol{b^{l+1}} \\ &= \boldsymbol{W}^l \widehat{\boldsymbol{a}}^l[t] + \boldsymbol{b}^{l+1} \sum_{\tau=0}^{t-1} \lambda^{t-\tau} + \lambda^t \widehat{\boldsymbol{U}}^{l+1}[0].\end{aligned}$$

Also, $\boldsymbol{u}^{l+1}[t]$ is computed as follows:

$$\begin{aligned}\boldsymbol{u}^{l+1}[t] &= \lambda(\boldsymbol{u}^{l+1}[t-1] - V_{\mathrm{th}}\, \boldsymbol{s}^{l+1}[t-1]) + \boldsymbol{W}^l \boldsymbol{s}^l[t] + \boldsymbol{b^{l+1}} \\ &= \lambda(\widehat{\boldsymbol{U}}^{l+1}[t-1] - V_{\mathrm{th}}(\lambda \widehat{\boldsymbol{a}}^{l+1}[t-2] + \boldsymbol{s}^{l+1}[t-1])) + \boldsymbol{W}^l(\widehat{\boldsymbol{a}}^l[t] - \lambda \widehat{\boldsymbol{a}}^l[t-1]) + \boldsymbol{b^{l+1}} \\ &= \widehat{\boldsymbol{U}}^{l+1}[t] - V_{\mathrm{th}} \lambda \widehat{\boldsymbol{a}}^{l+1}[t-1].\end{aligned}$$

With this equation, we have

$$\begin{aligned}\boldsymbol{s}^{l+1}[t] &= H(\boldsymbol{u}^{l+1}[t] - V_{\mathrm{th}}) \\ &= H(\widehat{\boldsymbol{U}}^{l+1}[t] - V_{\mathrm{th}}(\lambda \widehat{\boldsymbol{a}}^{l+1}[t-1] + 1)).\end{aligned}$$

Then, $\widehat{\boldsymbol{a}}^{l+1}[t]$ can be computed as follows:

$$\begin{aligned}\widehat{\boldsymbol{a}}^{l+1}[t] &= \sum_{\tau=0}^{t} \lambda^{t-\tau} \boldsymbol{s}^{l+1}[\tau] \\ &= \lambda \widehat{\boldsymbol{a}}^{l+1}[t-1] + \boldsymbol{s}^{l+1}[t] \\ &= \lambda \widehat{\boldsymbol{a}}^{l+1}[t-1] + H(\widehat{\boldsymbol{U}}^{l+1}[t] - V_{\mathrm{th}}(\lambda \widehat{\boldsymbol{a}}^{l+1}[t-1] + 1)).\end{aligned}$$

Hence, (4) and (5) hold for t when they hold for any $\tau \in \{1, \ldots, t-1\}$.

Therefore, (4) and (5) hold any $t \in \{1, \ldots, T\}$.

B.2 Derivation of (7) and (8)

First, since $L_E[t] = \mathcal{L}(\boldsymbol{s}^N[t], \boldsymbol{y})/T$, we have

$$\frac{\partial L_E[t]}{\partial \boldsymbol{W}^l} = \frac{\partial L_E[t]}{\partial \boldsymbol{s}^N[t]} \frac{\partial \boldsymbol{s}^N[t]}{\partial \boldsymbol{W}^l}.$$

Assuming that $L_E[t]$ depends only on $\widehat{\boldsymbol{a}}^l[t]$ and $\widehat{\boldsymbol{U}}^l[t]$, i.e., not on anything up to $t-1$, we regard $\widehat{\boldsymbol{a}}^N[t]$ as $\boldsymbol{s}^N[t] + \mathrm{Const}$. Then, we calculate that

$$\frac{\partial L_E[t]}{\partial \boldsymbol{W}^l} = \frac{\partial L_E[t]}{\partial \boldsymbol{s}^N[t]} \frac{\partial \boldsymbol{s}^N[t]}{\partial \widehat{\boldsymbol{a}}^N[t]} \frac{\partial \widehat{\boldsymbol{a}}^N[t]}{\partial \boldsymbol{W}^l} = \frac{\partial L_E[t]}{\partial \widehat{\boldsymbol{a}}^N[t]} \frac{\partial \widehat{\boldsymbol{a}}^N[t]}{\partial \boldsymbol{W}^l}. \tag{15}$$

Next, it follows from (4) that

$$\begin{aligned}\frac{\partial \widehat{\boldsymbol{a}}^N[t]}{\partial \boldsymbol{W}^l} &= \frac{\partial \widehat{\boldsymbol{a}}^N[t]}{\partial \widehat{\boldsymbol{a}}^N[t-1]}\frac{\partial \widehat{\boldsymbol{a}}^N[t-1]}{\partial \boldsymbol{W}^l} + \frac{\partial \widehat{\boldsymbol{a}}^N[t]}{\partial \widehat{\boldsymbol{U}}^N[t]}\frac{\partial \widehat{\boldsymbol{U}}^N[t]}{\partial \boldsymbol{W}^l} \\ &= \frac{\partial \widehat{\boldsymbol{a}}^N[t]}{\partial \widehat{\boldsymbol{U}}^N[t]}\frac{\partial \widehat{\boldsymbol{U}}^N[t]}{\partial \widehat{\boldsymbol{a}}^{N-1}[t]}\frac{\partial \widehat{\boldsymbol{a}}^{N-1}[t]}{\partial \boldsymbol{W}^l} \qquad (16) \\ &= \frac{\partial \widehat{\boldsymbol{a}}^N[t]}{\partial \widehat{\boldsymbol{a}}^{N-1}[t]}\frac{\partial \widehat{\boldsymbol{a}}^{N-1}[t]}{\partial \boldsymbol{W}^l}, \qquad (17)\end{aligned}$$

where we regard $\partial \widehat{\boldsymbol{a}}^N[t-1]/\partial \boldsymbol{W}^l$ as 0, and $\widehat{\boldsymbol{U}}^N[t]$ as a function of $\widehat{\boldsymbol{a}}^{N-1}[t]$. It should be noted that the assumption $\partial \widehat{\boldsymbol{a}}[t-1]/\partial \boldsymbol{W} = 0$ implies equalizing the effect of gradient on parameter updates at each time. Indeed, $\partial \widehat{\boldsymbol{a}}[t-1]/\partial \boldsymbol{W}$ is already used for updating parameters at time $t-1$. Therefore, using also $\partial \widehat{\boldsymbol{a}}[t-1]/\partial \boldsymbol{W}$ in the calculation of $\partial \widehat{\boldsymbol{a}}[t]/\partial \boldsymbol{W}$ will result in excessive influence of the gradient at $t-1$ on the parameter update at t.

By repeating the process from (16) to (17) in the same way, we derive

$$\frac{\partial \widehat{\boldsymbol{a}}^N[t]}{\partial \boldsymbol{W}^l} = \left(\prod_{i=N-1}^{l+1} \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[t]}{\partial \widehat{\boldsymbol{a}}^i[t]}\right)\frac{\partial \widehat{\boldsymbol{a}}^{l+1}[t]}{\partial \boldsymbol{W}^l}.$$

Therefore, the gradient of SAF-E is calculated as follows:

$$\begin{aligned}\frac{\partial L_E[t]}{\partial \boldsymbol{W}^l} &= \frac{\partial L_E[t]}{\partial \widehat{\boldsymbol{a}}^N[t]}\left(\prod_{i=N-1}^{l+1} \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[t]}{\partial \widehat{\boldsymbol{a}}^i[t]}\right)\frac{\partial \widehat{\boldsymbol{a}}^{l+1}[t]}{\partial \boldsymbol{W}^l} \\ &= \widehat{\boldsymbol{a}}^l[t]\,\frac{\partial L_E[t]}{\partial \widehat{\boldsymbol{a}}^N[t]}\left(\prod_{i=N-1}^{l+1} \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[t]}{\partial \widehat{\boldsymbol{a}}^i[t]}\right)\frac{\partial \widehat{\boldsymbol{a}}^{l+1}[t]}{\partial \widehat{\boldsymbol{U}}^{l+1}[t]}, \qquad (18)\end{aligned}$$

where note that the Heaviside step function H is element-wise. This concludes the derivation of (7) by setting

$$\boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{l+1}[t] = \frac{\partial L_E[t]}{\partial \widehat{\boldsymbol{a}}^N[t]}\left(\prod_{i=N-1}^{l+1} \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[t]}{\partial \widehat{\boldsymbol{a}}^i[t]}\right)\frac{\partial \widehat{\boldsymbol{a}}^{l+1}[t]}{\partial \widehat{\boldsymbol{U}}^{l+1}[t]}.$$

Derivation of (8) is almost the same as that of (7). As with SAF-E, suppose that L_F depends only on $\widehat{\boldsymbol{a}}^l[T]$ and $\widehat{\boldsymbol{U}}^l[T]$. Since $L_F = \mathcal{L}(\widehat{\boldsymbol{a}}^N[t]/\sum_{t=0}^{T}\lambda^{T-t}, \boldsymbol{y})$, (15) holds when t is replaced with T, and $L_E[t]$ is replaced with L_F. The rest derivation is the same as in case (7). Then we have

$$\frac{\partial L_F}{\partial \boldsymbol{W}^l} = \widehat{\boldsymbol{a}}^l[T]\,\frac{\partial L_F}{\partial \widehat{\boldsymbol{a}}^N[T]}\left(\prod_{i=N-1}^{l+1} \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[T]}{\partial \widehat{\boldsymbol{a}}^i[T]}\right)\frac{\partial \widehat{\boldsymbol{a}}^{l+1}[T]}{\partial \widehat{\boldsymbol{U}}^{l+1}[T]} = \widehat{\boldsymbol{a}}^l[T]\,\boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{l+1}[T]. \qquad (19)$$

Note that $\partial \widehat{\boldsymbol{a}}^{l+1}[t]/\partial \widehat{\boldsymbol{U}}^{l+1}[t]$ in (18) and (19) is non-differentiable; we approximate it with the SG.

B.3 Proof of Theorem 1

Theorem 1. *The backward processes of SAF-E and OTTT$_{\mathrm{O}}$ are identical , that is,* $\dfrac{\partial L_E[t]}{\partial \boldsymbol{W}^l} = \left(\dfrac{\partial L_E[t]}{\partial \boldsymbol{W}^l}\right)_{\mathrm{OT}}$.

Proof. The gradient of SAF-E and OTTT$_{\mathrm{O}}$ are as follows:

$$\frac{\partial L_E[t]}{\partial \boldsymbol{W}^l} = \widehat{\boldsymbol{a}}^l[t]\,\boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{l+1}[t], \quad \left(\frac{\partial L_E[t]}{\partial \boldsymbol{W}^l}\right)_{\mathrm{OT}} = \widehat{\boldsymbol{a}}^l[t]\,\frac{\partial L_E[t]}{\partial \boldsymbol{s}^N[t]}\left(\prod_{i=N-1}^{l+1} \frac{\partial \boldsymbol{s}^{i+1}[t]}{\partial \boldsymbol{s}^i[t]}\right)\frac{\partial \boldsymbol{s}^{l+1}[t]}{\partial \boldsymbol{u}^{l+1}[t]}.$$

We show that the gradient of SAF-E is equal to the gradient of $\mathrm{OTTT_O}$ by transforming $\boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{l+1}[t]$. First, we calculate $\partial L_E[t]/\partial \widehat{\boldsymbol{a}}^N[t]$. Because $L_E[t]$ does not include any argument up to $t-1$, it holds that

$$\frac{\partial L_E[t]}{\partial \boldsymbol{a}^N[t]} = \frac{\partial L_E[t]}{\partial \boldsymbol{s}^N[t]} \frac{\partial \boldsymbol{s}^N[t]}{\partial \widehat{\boldsymbol{a}}^N[t]} = \frac{\partial L_E[t]}{\partial \boldsymbol{s}^N[t]}.$$

Then, we have

$$\boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{l+1}[t] = \frac{\partial L_E[t]}{\partial \boldsymbol{s}^N[t]} \left(\prod_{i=N-1}^{l+1} \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[t]}{\partial \widehat{\boldsymbol{a}}^{i}[t]} \right) \frac{\partial \widehat{\boldsymbol{a}}^{l+1}[t]}{\partial \widehat{\boldsymbol{U}}^{l+1}[t]}.$$

Second, because of the forward process of SAF, i.e., (4) and (5), we obtain that

$$\begin{aligned} \frac{\partial \widehat{\boldsymbol{a}}^{l+1}[t]}{\partial \widehat{\boldsymbol{U}}^{l+1}[t]} &= \delta(\widehat{\boldsymbol{U}}^{l+1}[t] - V_{\mathrm{th}}(\lambda \widehat{\boldsymbol{a}}^{l+1}[t-1] + 1) \\ &= \delta(\boldsymbol{u}^{l+1}[t] - V_{\mathrm{th}}), \end{aligned}$$

where δ represents the delta function. On the other hand, the following equation can be derived from the forward process of $\mathrm{OTTT_O}$, i.e., (1):

$$\frac{\partial \boldsymbol{s}^{l+1}[t]}{\partial \boldsymbol{u}^{l+1}[t]} = \delta(\boldsymbol{u}^{l+1}[t] - V_{\mathrm{th}}).$$

Hence,

$$\frac{\partial \widehat{\boldsymbol{a}}^{l+1}[t]}{\partial \widehat{\boldsymbol{U}}^{l+1}[t]} = \frac{\partial \boldsymbol{s}^{l+1}[t]}{\partial \boldsymbol{u}^{l+1}[t]}. \tag{20}$$

Then, we have that

$$\boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{l+1}[t] = \frac{\partial L_E[t]}{\partial \boldsymbol{s}^N[t]} \left(\prod_{i=N-1}^{l+1} \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[t]}{\partial \widehat{\boldsymbol{a}}^{i}[t]} \right) \frac{\partial \boldsymbol{s}^{l+1}[t]}{\partial \boldsymbol{u}^{l+1}[t]}.$$

From (4), $\widehat{\boldsymbol{a}}^{i+1}[t]$ depends on $\widehat{\boldsymbol{a}}^{i+1}[t-1]$ and $\widehat{\boldsymbol{U}}^{i}[t]$, while $\widehat{\boldsymbol{a}}^{i+1}[t-1]$ does not depend on $\widehat{\boldsymbol{a}}^{i+1}[t]$, Thus, we calculate that

$$\frac{\partial \widehat{\boldsymbol{a}}^{i+1}[t]}{\partial \widehat{\boldsymbol{a}}^{i}[t]} = \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[t]}{\partial \widehat{\boldsymbol{U}}^{i+1}[t]} \frac{\partial \widehat{\boldsymbol{U}}^{i+1}[t]}{\partial \widehat{\boldsymbol{a}}^{i}[t]} + \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[t]}{\partial \widehat{\boldsymbol{a}}^{i+1}[t-1]} \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[t-1]}{\partial \widehat{\boldsymbol{a}}^{i}[t]} = \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[t]}{\partial \widehat{\boldsymbol{U}}^{i+1}[t]} \boldsymbol{W}^i. \tag{21}$$

Moreover, $\boldsymbol{s}^{i+1}[t]$ depends on $\boldsymbol{u}^{i+1}[t]$ because the forward process of $\mathrm{OTTT_O}$ is given by (1). Hence,

$$\frac{\partial \boldsymbol{s}^{i+1}[t]}{\partial \boldsymbol{s}^{i}[t]} = \frac{\partial \boldsymbol{s}^{i+1}[t]}{\partial \boldsymbol{u}^{i+1}[t]} \frac{\partial \boldsymbol{u}^{i+1}[t]}{\partial \boldsymbol{s}^{i}[t]} = \frac{\partial \boldsymbol{s}^{i+1}[t]}{\partial \boldsymbol{u}^{i+1}[t]} \boldsymbol{W}^i. \tag{22}$$

Combining (20), (21) and (22), we have

$$\frac{\partial \widehat{\boldsymbol{a}}^{i+1}[t]}{\partial \widehat{\boldsymbol{a}}^{i}[t]} = \frac{\partial \boldsymbol{s}^{i+1}[t]}{\partial \boldsymbol{s}^{i}[t]}.$$

In the end, we transform $\boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{l+1}[t]$ as follows:

$$\boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{l+1}[t] = \frac{\partial L_E[t]}{\partial \boldsymbol{s}^N[t]} \left(\prod_{i=N-1}^{l+1} \frac{\partial \boldsymbol{s}^{i+1}[t]}{\partial \boldsymbol{s}^{i}[t]} \right) \frac{\partial \boldsymbol{s}^{l+1}[t]}{\partial \boldsymbol{u}^{l+1}[t]}. \tag{23}$$

This concludes the proof. □

B.4 Proof of Theorem 2

Theorem 2. *Suppose that $\boldsymbol{m}[t]$ converges when $t \to \infty$. Then, for sufficiently large T, the backward processes of SAF-F and Spike Representation are identical up to a scale factor, that is,* $\dfrac{\partial L_F}{\partial \boldsymbol{W}^l} = V_{\mathrm{th}} \left(\dfrac{\partial L_F}{\partial \boldsymbol{W}^l} \right)_{\mathrm{SR}}$.

Proof. From assumptions, $\boldsymbol{a}^{l+1}[T] \approx \sigma\left((\boldsymbol{W}^l \boldsymbol{a}^l[T] + \boldsymbol{b}^{l+1})/V_{\mathrm{th}} \right)$, where $\sigma(x) = \min(\max(0, x), 1)$. Therefore, the followings hold for $i = l+1, \ldots, N-1$:

$$\frac{\partial L_F}{\partial \boldsymbol{W}^l} = \frac{\partial L_F}{\partial \boldsymbol{a}^N[T]} \frac{\partial \boldsymbol{a}^N[T]}{\partial \boldsymbol{W}^l}, \tag{24}$$

$$\frac{\partial \boldsymbol{a}^{i+1}[T]}{\partial \boldsymbol{W}^l} = \frac{\partial \boldsymbol{a}^{i+1}[T]}{\partial \boldsymbol{a}^i[T]} \frac{\partial \boldsymbol{a}^i[T]}{\partial \boldsymbol{W}^l}. \tag{25}$$

By repeatedly substituting (25) for (24), we can calculate $\partial L_F / \partial \boldsymbol{W}^l$ as follows:

$$\frac{\partial L_F}{\partial \boldsymbol{W}^l} = \frac{\partial L_F}{\partial \boldsymbol{a}^N[T]} \left(\prod_{i=N-1}^{l+1} \frac{\partial \boldsymbol{a}^{i+1}[T]}{\partial \boldsymbol{a}^i[T]} \right) \frac{\partial \boldsymbol{a}^{l+1}[T]}{\partial \boldsymbol{W}^l}. \tag{26}$$

This is the gradient of Spike Representation and denote it by $(\partial L_F / \partial \boldsymbol{W}^l)_{\mathrm{SR}}$. We will show that $(\partial L_F / \partial \boldsymbol{W}^l)_{\mathrm{SR}}$ is in proportion to $\partial L_F / \partial \boldsymbol{W}^l = \widehat{\boldsymbol{a}}^l[T]\, \boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{l+1}[T]$, which is the gradient of SAF-F.

First, let $\Lambda = \sum_{\tau=0}^{T} \lambda^{T-\tau}$. Then $\boldsymbol{a}^l[T] = \widehat{\boldsymbol{a}}^l[T]/\Lambda$ for any layer l. From linearity of differentiation and change of variables, we have followings:

$$\frac{\partial L_F}{\partial \boldsymbol{a}^N[T]} = \frac{\partial L_F}{\Lambda \partial \widehat{\boldsymbol{a}}^N[T]}, \quad \frac{\partial \boldsymbol{a}^{i+1}[T]}{\partial \boldsymbol{a}^i[T]} = \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[T]}{\partial \widehat{\boldsymbol{a}}^i[T]}, \quad \frac{\partial \boldsymbol{a}^{l+1}[T]}{\partial \boldsymbol{W}^l} = \frac{\Lambda \partial \widehat{\boldsymbol{a}}^{l+1}[T]}{\partial \boldsymbol{W}^l}. \tag{27}$$

Substituting (27) into (26), then we obtain

$$\left(\frac{\partial L_F}{\partial \boldsymbol{W}^l} \right)_{\mathrm{SR}} = \frac{\partial L_F}{\partial \widehat{\boldsymbol{a}}^N[T]} \left(\prod_{i=N-1}^{l+1} \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[T]}{\partial \widehat{\boldsymbol{a}}^i[T]} \right) \frac{\partial \widehat{\boldsymbol{a}}^{l+1}[T]}{\partial \boldsymbol{W}^l}.$$

Second, it follows from $\boldsymbol{a}^l[T] = \widehat{\boldsymbol{a}}^l[T]/\Lambda$ and $\boldsymbol{a}^{l+1}[T] \approx \sigma\big((\boldsymbol{W}^l \boldsymbol{a}^l[T] + \boldsymbol{b}^{l+1})/V_{\mathrm{th}} \big)$ that

$$\widehat{\boldsymbol{a}}^{l+1}[T] \approx \Lambda \sigma \left(\frac{1}{V_{\mathrm{th}}} \left(\frac{1}{\Lambda} \boldsymbol{W}^l \widehat{\boldsymbol{a}}^l[T] + \boldsymbol{b}^{l+1} \right) \right).$$

Here, taking care that σ is element-wise, we calculate as follows:

$$\begin{aligned} \left(\frac{\partial L_F}{\partial \boldsymbol{W}^l} \right)_{\mathrm{SR}} &= \frac{\widehat{\boldsymbol{a}}^l[T]}{V_{\mathrm{th}} \Lambda} \left(\frac{\partial L_F}{\partial \widehat{\boldsymbol{a}}^N[T]} \left(\prod_{i=N-1}^{l+1} \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[T]}{\partial \widehat{\boldsymbol{a}}^i[T]} \right) \odot \Lambda \sigma' \left(\frac{1}{V_{\mathrm{th}}} \left(\frac{1}{\Lambda} \boldsymbol{W}^l \widehat{\boldsymbol{a}}^l[T] + \boldsymbol{b}^{l+1} \right)^\top \right) \right) \\ &= \frac{\widehat{\boldsymbol{a}}^l[T]}{V_{\mathrm{th}}} \left(\frac{\partial L_F}{\partial \widehat{\boldsymbol{a}}^N[T]} \left(\prod_{i=N-1}^{l+1} \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[T]}{\partial \widehat{\boldsymbol{a}}^i[T]} \right) \odot \boldsymbol{d}^{l+1}[T]^\top \right), \end{aligned}$$

where we set $\boldsymbol{d}^{l+1}[T] = \sigma'\left((\boldsymbol{W}^l \widehat{\boldsymbol{a}}^l[T]/\Lambda + \boldsymbol{b}^{l+1})/V_{\mathrm{th}} \right)$, and $\odot$ is the element-wise product. Now we assume that

$$\frac{\partial \boldsymbol{s}^{l+1}[T]}{\partial \boldsymbol{u}^{l+1}[T]} = \mathrm{diag}(\boldsymbol{d}^{l+1}[T]),$$

for any $l = 0, \ldots, N-1$, where $\mathrm{diag}(\boldsymbol{d}^{l+1}[T])$ is a diagonal matrix constructed from $\boldsymbol{d}^{l+1}[T]$. The reason why this assumption is valid discussed in Xiao et al. (2022). Then, we obtain from (20) that

$$\left(\frac{\partial L_F}{\partial \boldsymbol{W}^l}\right)_{\mathrm{SR}} = \frac{\widehat{\boldsymbol{a}}^l[T]}{V_{\mathrm{th}}} \frac{\partial L_F}{\partial \widehat{\boldsymbol{a}}^N[T]} \left(\prod_{i=N-1}^{l+1} \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[T]}{\partial \widehat{\boldsymbol{a}}^i[T]}\right) \frac{\partial \widehat{\boldsymbol{a}}^{l+1}[T]}{\partial \widehat{\boldsymbol{U}}^{l+1}[T]} = \frac{1}{V_{\mathrm{th}}} \widehat{\boldsymbol{a}}^l[T]\, \boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{l+1}[T].$$

□

B.5 Proof of Corollary 3

Corollary 3. *For SNN with a feedforward connection* (11)*, or a feedback connection* (13)*, the following hold.*

(i) *The backward processes of SAF-E and OTTT$_{\mathrm{O}}$ with a feedforward connection are identical.*

(ii) *Suppose that $\boldsymbol{m}[t]$ converges when $t \to \infty$. Then, for sufficiently large T, the backward processes of SAF-F and Spike Representation with a feedforward connection are identical up to a scale factor.*

(iii) *The backward processes of SAF-E and OTTT$_{\mathrm{O}}$ with a feedback connection are identical.*

Proof. First, we show (i). The gradients for parameters other than $\boldsymbol{W}_f$ are equal to the gradients when there is no feedforward connection. From Theorem 1, these gradients are same as OTTT$_{\mathrm{O}}$. Therefore, we only need to check $\partial L_E[t]/\partial \boldsymbol{W}_f = (\partial L_E[t]/\partial \boldsymbol{W}_f)_{\mathrm{OT}}$. In fact, the gradient of OTTT$_{\mathrm{O}}$ calculated (see Xiao et al. (2022)) as

$$\left(\frac{\partial L_E[t]}{\partial \boldsymbol{W}_f}\right)_{\mathrm{OT}} = \widehat{\boldsymbol{a}}^p[t] \frac{\partial L[t]}{\partial \boldsymbol{s}^N[t]} \left(\prod_{i=N-1}^{q+1} \frac{\partial \boldsymbol{s}^{i+1}[t]}{\partial \boldsymbol{s}^i[t]}\right) \frac{\partial \boldsymbol{s}^{q+1}[t]}{\partial \boldsymbol{u}^{q+1}[t]},$$

and (23) holds, then we have

$$\left(\frac{\partial L_E[t]}{\partial \boldsymbol{W}_f}\right)_{\mathrm{OT}} = \widehat{\boldsymbol{a}}^p[t]\, \boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{q+1}[t] = \frac{\partial L_E[t]}{\partial \boldsymbol{W}_f}.$$

Hence, SAF-E is equivalent to OTTT$_{\mathrm{O}}$ even if there is a feedforward connection in SNN. The same method can be used to prove (iii).

Next, we show (ii). The gradients for parameters other than $\boldsymbol{W}_f$ remain the same as when there is no feedback connection. According to Theorem 2, these gradients are identical up to a scale factor to Spike Representation. Therefore, we only need to check $\partial L_F[t]/\partial \boldsymbol{W}_b = V_{\mathrm{th}}(\partial L_F[t]/\partial \boldsymbol{W}_f)_{\mathrm{SR}}$. When there is a feedforward connection, (2) holds. Therefore, it can be proved in the same way as Appendix B.4, noting that $\boldsymbol{a}^{q+1}[T] \approx \sigma\left((\boldsymbol{W}^q \boldsymbol{a}^q[T] + \boldsymbol{b}^{l+1} + \boldsymbol{W}_f \boldsymbol{a}^p[T]/V_{\mathrm{th}}\right)$. □

B.6 Proof of Theorem 4

Theorem 4. *Suppose that $\boldsymbol{m}[t]$ converges when $t \to \infty$. Then, for sufficiently large T, the backward processes of SAF-F and Spike Representation with a feedback connection are close, that is, $\left\langle \frac{\partial L_F}{\partial \boldsymbol{\theta}}, \left(\frac{\partial L_F}{\partial \boldsymbol{\theta}}\right)_{\mathrm{SR}} \right\rangle > 0$ for all parameters $\boldsymbol{\theta}$.*

Proof. From the assumption, the firing rates of each layer converge to these equilibrium points: $(\boldsymbol{a}^{l+1})^* = f_{l+1}((\boldsymbol{a}^l)^*)$ $(l \neq q)$, $(\boldsymbol{a}^{q+1})^* = f_{q+1}(f_p \circ \cdots \circ f_{q+2}((\boldsymbol{a}^{q+1})^*), (\boldsymbol{a}^q)^*) = f_{q+1}((\boldsymbol{a}^p)^*, (\boldsymbol{a}^q)^*)$, where $f_{l+1}((\boldsymbol{a}^l)^*) = \sigma\left((\boldsymbol{W}^l(\boldsymbol{a}^l)^* + \boldsymbol{b}^{l+1})/V_{\mathrm{th}}\right)$, and $f_{q+1}((\boldsymbol{a}^p)^*, (\boldsymbol{a}^q)^*) = \sigma\left((\boldsymbol{W}^q(\boldsymbol{a}^q)^* + \boldsymbol{b}^{q+1} + \boldsymbol{W}_b(\boldsymbol{a}^p)^*/V_{\mathrm{th}}\right)$. Since $T \gg 1$, $\partial L_F/\partial \boldsymbol{\theta}$ can be calculated using the spike representation as follows (see Xiao et al. (2021),Xiao et al. (2022)):

$$\left(\frac{\partial L_F}{\partial \boldsymbol{\theta}}\right)_{\mathrm{SR}} = \frac{\partial L_F}{\partial \boldsymbol{a}^{l+1}[T]} \left(I - \frac{\partial f_{l+1}}{\partial \boldsymbol{a}^l[T]}\right)^{-1} \frac{\partial f_{l+1}(\boldsymbol{a}^l[T])}{\partial \boldsymbol{\theta}}, \tag{28}$$

where $\partial f_{l+1}/\partial \boldsymbol{a}^l[T]$ denotes Jacobian matrix and I denotes identity matrix. Here, we regard $(I - \partial f_{l+1}/\partial \boldsymbol{a}^l[T])^{-1}$ of (28) as an identity matrix, and denote it by

$$\widetilde{\left(\frac{\partial L_F}{\partial \boldsymbol{\theta}}\right)}_{\mathrm{SR}} = \frac{\partial L_F}{\partial \boldsymbol{a}^{l+1}[T]} \frac{\partial f_{l+1}(\boldsymbol{a}^l[T])}{\partial \boldsymbol{\theta}}. \tag{29}$$

Then, it is proved in Fung et al. (2022) and Geng et al. (2021) that

$$\left\langle \widetilde{\left(\frac{\partial L_F}{\partial \boldsymbol{\theta}}\right)}_{\mathrm{SR}}, \left(\frac{\partial L_F}{\partial \boldsymbol{\theta}}\right)_{SR} \right\rangle > 0,$$

where $\langle \cdot, \cdot \rangle$ denotes inner product. If $\boldsymbol{\theta}$ is not $\boldsymbol{W}_b$, right-hand side of (29) is equal to right-hand side of (26), then from Theorem 2,

$$\widetilde{\left(\frac{\partial L_F}{\partial \boldsymbol{\theta}}\right)}_{\mathrm{SR}} = \frac{1}{V_{\mathrm{th}}} \frac{\partial L_F}{\partial \boldsymbol{\theta}}.$$

Next, we consider $\boldsymbol{\theta}$ as $\boldsymbol{W}_b$. $\boldsymbol{a}^{q+1}[T]$ can be approximated by $\sigma\left((\boldsymbol{W}^q \boldsymbol{a}^q[T] + \boldsymbol{b}^{q+1} + \boldsymbol{W}_b\, \boldsymbol{a}^p[T])/V_{\mathrm{th}}\right)$ and $\boldsymbol{a}^p[T] \approx \boldsymbol{a}^p[T-1]$ because T is large. Therefore, $\boldsymbol{a}^{q+1}[T] \approx \sigma\left((\boldsymbol{W}^q \boldsymbol{a}^q[T] + \boldsymbol{b}^{q+1} + \boldsymbol{W}_b\, \boldsymbol{a}^p[T-1])/V_{\mathrm{th}}\right)$. Setting $\boldsymbol{d}_b^{l+1}[T] = \sigma'\left((\boldsymbol{W}^l \boldsymbol{a}^l[T] + \boldsymbol{b}^{l+1} + \boldsymbol{W}_b\, \boldsymbol{a}^k[T-1])/V_{\mathrm{th}}\right)$ and calculating similar to Sec. B.4, we derive that

$$\begin{aligned}
\widetilde{\left(\frac{\partial L_F}{\partial \boldsymbol{W}_b}\right)}_{\mathrm{SR}} &= \frac{\widehat{\boldsymbol{a}}^k[T-1]}{V_{\mathrm{th}}} \left(\frac{\partial L_F}{\partial \widehat{\boldsymbol{a}}^N[T]} \left(\prod_{i=N-1}^{l+1} \frac{\partial \widehat{\boldsymbol{a}}^{i+1}[T]}{\partial \widehat{\boldsymbol{a}}^i[T]}\right) \odot \boldsymbol{d}_b^{l+1}[T]\right) \\
&= \frac{1}{V_{\mathrm{th}}} \widehat{\boldsymbol{a}}^p[T-1]\, \boldsymbol{g}_{\widehat{\boldsymbol{U}}}^{q+1}[T] \\
&= \frac{1}{V_{\mathrm{th}}} \left(\frac{\partial L_F}{\partial \boldsymbol{W}_b}\right).
\end{aligned}$$

In the end, take the inner product between their gradient we obtain that

$$\left\langle \left(\frac{\partial L_F}{\partial \boldsymbol{\theta}}\right), \left(\frac{\partial L_F}{\partial \boldsymbol{\theta}}\right)_{SR} \right\rangle = V_{\mathrm{th}} \left\langle \widetilde{\left(\frac{\partial L_F}{\partial \boldsymbol{\theta}}\right)}_{\mathrm{SR}}, \left(\frac{\partial L_F}{\partial \boldsymbol{\theta}}\right)_{SR} \right\rangle > 0.$$

□

C Implementation detail

In the experiments of our study, we used the VGG network as follows:

64C3-128C3-AP2-256C3-256C3-AP2-512C3-512C3-AP2-512C3-512C3-GAP-FC,

where xCy represents the convolutional layer with x-output channels and y-stride, APx represents the average pooling with the kernel size 2, GAP represents the global average pooling, and FC represents the fully connected layer.

As for the time step, we set it to 6 if there is no mention. We used the stochastic gradient descent (SGD) as the optimizer with the batch size, epoch, initial learning rate for the cosine annealing, and momentum at

Table 3: Gradient comparison of SAF-E and $\mathrm{OTTT_O}$ on CIFAR-10 and CIFAR-100.

Dataset	T	**Correlation Coefficient**	**MAE** ($\times 10^{-6}$)
CIFAR-10	6	0.984±0.001	2.26±0.10
CIFAR-100	6	0.969±0.001	12.4±0.40

Table 4: Gradient comparison of SAF-F and $\mathrm{OTTT_A}$ on CIFAR-10 and CIFAR-100.

Dataset	T	**Correlation Coefficient**	**MAE** ($\times 10^{-6}$)
CIFAR-10	6	0.566±0.097	23.2±4.40
CIFAR-10	32	0.557±0.064	22.5±4.00
CIFAR-100	6	0.590±0.021	72.6±12.8
CIFAR-100	32	0.557±0.030	73.6±17.2

128, 300, 0.1, and 0.9. As for the loss function, we applied the combination of the cross-entropy (CE) and the mean-squared error (MSE) losses, i.e., $L = (1-\alpha)\mathrm{CE} + \alpha\mathrm{MSE}$ (where $\alpha = 0.05$). In addition, we set the leaky term as $\lambda = 0.5$ and the threshold as $V_{\mathrm{th}} = 1$ and used the scaled weight standardization (sWS) (Qiao et al., 2019). We applied derivative of a sigmoid function as a surrogate gradient:

$$\frac{\partial \widehat{a}^{l+1}[t]}{\partial \widehat{U}^{l+1}[t]} := \frac{1}{\beta} \frac{\exp((V_{\mathrm{th}} - \widehat{U}^{l+1}[t])/\beta)}{(1+\exp((V_{\mathrm{th}} - \widehat{U}^{l+1}[t])/\beta))^2},$$

where β is a hyperparameter and we set $\beta = 4$. Note that all settings, including the above, were the same as Xiao et al. (2022).

D Comparison of gradient

Tables 3 and 4 show the correlation coefficients and mean absolute errors (MAE) of gradients of the SAF and OTTT. These values were computed in the input layer after training, where the gradient error between SAF and OTTT were the largest. As can be seen from the table, it is considered that SAF-E and $\mathrm{OTTT_O}$ have the same gradients, while SAF-F and $\mathrm{OTTT_A}$ have different (but close direction) gradients. These results clearly support our theory.

E Experimental Analysis of SAF-E with Feedback and FeedForward Connection

In this section, we confirm that SAF-E and $\mathrm{OTTT_O}$ are equivalent even if there is a feedback or feedforward connection. First, we compare the performance of SAF-E with $\mathrm{OTTT_O}$ when both networks have a feedback connection from the first layer to the N-th layer. The setup was the same as in the experiments of Xiao et al. (2022).

Table 5 shows the accuracy and total firing rate when we set $T = 6$. As shown in this table, SAF-E and $\mathrm{OTTT_O}$ with feedback connection are almost close. Also, the change due to inference with SNNs consisting of LIF neurons is also almost negligible. Figure 6 shows the accuracy, loss curve, and the firing rates of each layer. From these results, we confirmed experimentally that SAF-E and $\mathrm{OTTT_O}$ with feedback connection are close, which consistent with the assertion in Corollary. 3.

Next, we compare the performance of SAF-E with $\mathrm{OTTT_O}$ when both networks have a feedforward connection. We used the RepVGG network (Ding et al., 2021). The results are shown in Table 6 and Figure 7. As in the case of feedback connections, we can also see that SAF-E and $\mathrm{OTTT_O}$ with a feedforward connection are close.

Table 5: Comparison of SAF-E and OTTT_O with feedback connection on CIFAR-10 ($T = 6$). The values in parentheses are the changes in firing rate and accuracy due to inference by the SNN composed of LIF.

Method	**Firing rate** [%]	**Accuracy** [%]
OTTT_O with feedback	14.74±0.34	93.23±0.28
SAF-E with feedback (ours)	14.32±0.12 (1.78×10^{-6})	93.20±0.18 (6.67×10^{-5})

Table 6: Comparison of SAF-E and OTTT_O with feedforward connection on CIFAR-10 ($T = 6$). The values in parentheses are the changes in firing rate and accuracy due to inference by the SNN composed of LIF.

Method	**Firing rate** [%]	**Accuracy** [%]
OTTT_O with feedforward	15.03±0.10	93.53±0.15
SAF-E with feedforward (ours)	14.72±0.04 (3.342×10^{-5})	93.53±0.17 (0.004)

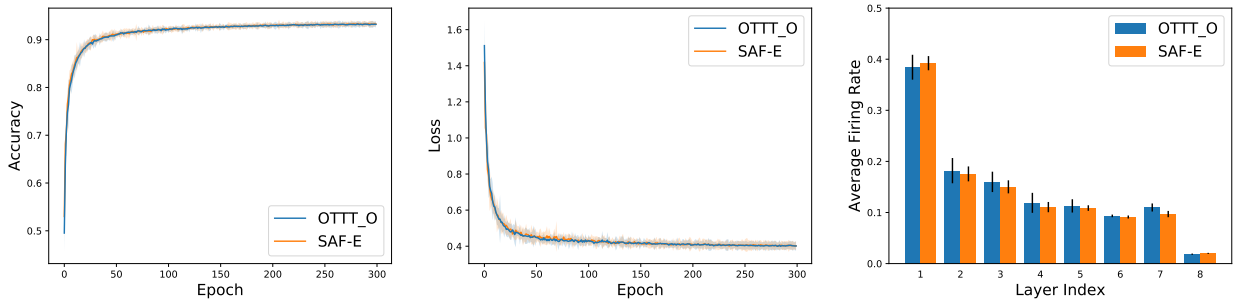


Figure 6: Accuracy and loss curves, and firing rates of each layer of SAF-E and OTTT_O with feedback connection on CIFAR-10 ($T = 6$).

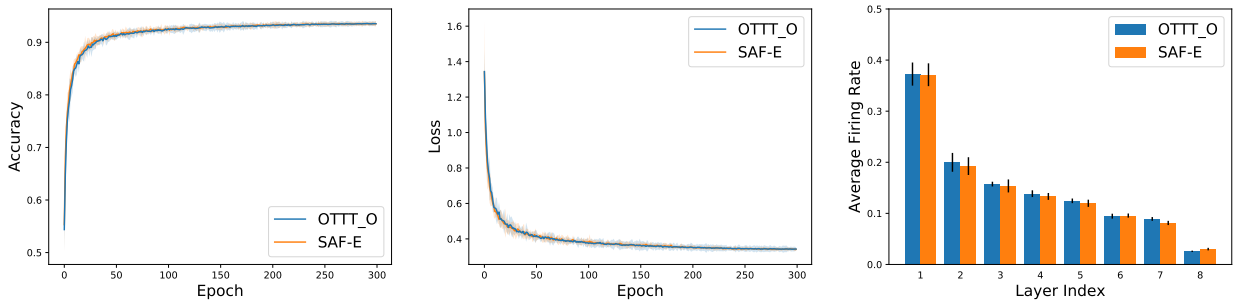


Figure 7: Accuracy and loss curves, and firing rates of each layer of SAF-E and OTTT_O with feedforward connection on CIFAR-10 ($T = 6$).