Deep Spike Learning With Local Classifiers

Chenxiang Ma, Rui Yan, *Member, IEEE*, Zhaofei Yu, and Qiang Yu, *Senior Member, IEEE*

***Abstract*—Backpropagation has been successfully generalized to optimize deep spiking neural networks (SNNs), where, nevertheless, gradients need to be propagated back through all layers, resulting in a massive consumption of computing resources and an obstacle to the parallelization of training. A biologically motivated scheme of local learning provides an alternative to efficiently train deep networks but often suffers a low performance of accuracy on practical tasks. Thus, how to train deep SNNs with the local learning scheme to achieve both efficient and accurate performance still remains an important challenge. In this study, we focus on a supervised local learning scheme where each layer is independently optimized with an auxiliary classifier. Accordingly, we first propose a spike-based efficient local learning rule by only considering the direct dependencies in the current time. We then propose two variants that additionally incorporate temporal dependencies through a backward and forward process, respectively. The effectiveness and performance of our proposed methods are extensively evaluated with six mainstream datasets. Experimental results show that our methods can successfully scale up to large networks and substantially outperform the spike-based local learning baselines on all studied benchmarks. Our results also reveal that gradients with temporal dependencies are essential for high performance on temporal tasks, while they have negligible effects on rate-based tasks. Our work is significant as it brings the performance of spike-based local learning to a new level with the computational benefits being retained.**

***Index Terms*—Deep learning, local learning, neuromorphic computing, online learning, spiking neural networks (SNNs).**

I. Introduction

HUMAN brain, one of the most remarkable products of evolution, shows impressive capabilities of learning, reasoning, decision making, efficient computation, etc. [1], [2]. Such fascinating capabilities have attracted increasing attention to not only reveal the underlying mechanisms but also take inspiration to develop an approaching intelligence as the brain.

Manuscript received 22 April 2022; revised 5 June 2022; accepted 18 June 2022. Date of publication 22 July 2022; date of current version 21 April 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62176179, and in part by Zhejiang Lab under Grant 2021KC0AB03. This article was recommended by Associate Editor P. Shi. *(Corresponding author: Qiang Yu.)*

Chenxiang Ma is with the Tianjin Key Laboratory of Cognitive Computing and Application, College of Intelligence and Computing, Tianjin University, Tianjin 300072, China.

Rui Yan is with the College of Computer Science, Zhejiang University of Technology, Hangzhou 310012, China.

Zhaofei Yu is with the Institute for Artificial Intelligence, Peking University, Beijing 100871, China.

Qiang Yu is with the College of Intelligence and Computing, Tianjin University, Tianjin 300072, China, and also with Peng Cheng Laboratory, Shenzhen 518055, China (e-mail: yuqiang@tju.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCYB.2022.3188015.

Digital Object Identifier 10.1109/TCYB.2022.3188015

As an elementary unit of the brain, biological neurons connect with each other via synapses to build different functional networks. By simply abstracting such structures, artificial neural networks (ANNs) [3] are developed as one of the early steps toward artificial intelligence and have achieved a performance of human-level or even better accuracy across a wide range of tasks owing to the recent emergence of deep learning [4]. However, running ANNs usually incurs a substantial energy overhead, impeding further development under energy-constrained conditions [5]. In contrast, the human brain is capable of achieving an efficient computation with a very small power consumption [2]. The spike-based machinery for both communication and processing in the brain is believed to play an essential role in such an advantageous efficiency [1], [6]. Spiking neural networks (SNNs) [7] are thus developed by taking inspiration from this spike-based representation and processing.

SNNs need to be properly adapted in a way such that they can deliver desired responses to external signals. However, training SNNs, especially for deep ones, is rather challenging due to the nondifferentiable spike firing state and the complex spatiotemporal dynamics [8]. Some studies make efforts to directly convert weights from a pretrained ANN into an SNN of the same structure and achieve a comparable accuracy [9]–[12]. However, this indirect training is relatively inefficient in both time and energy since the converted SNNs typically require a large number of spike events and time steps for a good performance.

Differently, some other studies try to utilize the prominent backpropagation (BP) [13] from ANNs to directly train deep SNNs. However, BP cannot be directly applied due to the nondifferentiable spike firing function. In order to address this problem, SpikeProp [14] utilizes a linear assumption to approximate gradients of the spike firing function. Alternatively, a surrogate gradient [15] can be used to substitute the one for the firing function, and it becomes a prevalent way to address the nondifferentiable problem due to its simplicity and effectiveness. Recent advances with the mechanism of BP through time (BPTT) have further improved the accuracy performance of SNNs by jointly evaluating gradients from both spatial and temporal dimensions [16]–[21]. Despite the high accuracy, BP needs to propagate gradients from the output layer back through the whole network, and the weights of hidden layers cannot be updated until the backward propagation is completed. This impedes efficient parallelization of the training process, resulting in a relatively large requirement on computational resources and significant challenges for neuromorphic implementations [22].

Instead of being aware of the overall network, biological synapses may only have access to locally available

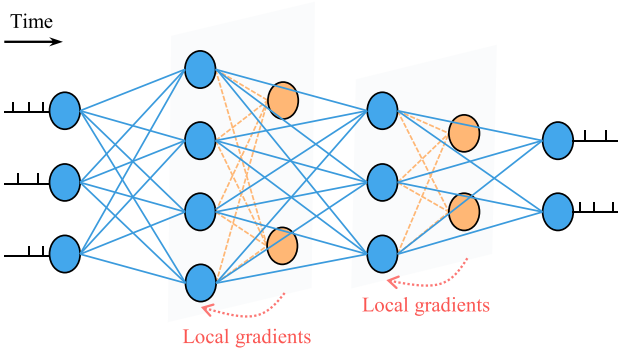


Fig. 1. Illustration of the spike-based local learning scheme. Each layer (blue units) is trained locally with an auxiliary classifier (orange units).

information to modulate synaptic weights in a highly parallel way [23], [24]. This local learning scheme could independently update the weights of each layer, thus providing an alternative for efficient training of deep SNNs [5], [25]. Unsupervised learning rules are naturally suited for the local learning category [26], [27]. For example, the spike-timing-dependent plasticity (STDP) [24] could modulate synaptic weights based on the relative timings of pre- and post-synaptic spikes. However, due to the lack of instructor signals, SNNs trained with unsupervised learning rules are normally limited to shallow structures and achieve relatively low accuracy on practical tasks [26], [28]–[30]. Recently, a supervised local learning scheme emerges by utilizing layer-wise loss functions such that local updates can be performed with training labels [31]–[33]. Using auxiliary classifiers is a straightforward yet effective way to construct the layer-wise loss functions. As a spike-based variant of this scheme, deep continuous local learning (DECOLLE) [34] adopts fixed and random auxiliary classifiers and is able to continuously train deep SNNs at each time step by using temporally local information. However, the performance is still relatively limited, especially on large networks, as compared to BP-based learning rules. Therefore, it is still challenging to obtain both accurate and efficient performance for deep SNNs with local learning rules.

In this study, we focus on the supervised local learning scheme with layer-wise auxiliary classifiers (see Fig. 1 for illustration). According to the way how gradients are calculated, we propose three new spike-based efficient local learning rules, namely, ELL, BELL, and FELL. The ELL simply ignores the temporal dependencies, while the BELL and FELL propagate these through a backward and forward process, respectively. Our major contributions can be summarized as follows.

1) The performance of SNNs with local learning is substantially improved. Our results show that the as-proposed methods can successfully scale up to large networks and significantly outperform the spike-based local learning baselines in terms of accuracy on all evaluated benchmark datasets. The advanced performance of our local learning rules provides important and valuable references for spike-based processing and learning.
2) Our learning rules have computational advantages in terms of memory, training time, and firing sparsity, indicating high energy efficiency. In addition, we provide a clear explanation of the accuracy advantage of our methods. We reveal that the gradients with temporal dependencies are essential for high performance on temporal tasks, but have little effect under a rate-based code. These findings could provide better guidance for choosing appropriate methods for various tasks.
3) Our proposed rules can be generalized under certain biologically plausible constraints. The biologically unrealistic weight symmetry requirement can be relieved with nearly lossless accuracy for large networks on a challenging task.

The remainder of this article is organized as follows. Section II describes the details of our proposed methods, followed by experimental results and discussions in Sections III and IV, respectively. Finally, a conclusion is provided in Section V.

II. Methods

A. Spiking Neuron Model

SNNs are dynamical systems that involve complex information transmission and processing under both spatial and temporal dimensions. There are many mathematical models to emulate the dynamics of a spiking neuron with different levels of biofidelity and computational complexity [35]. The current-based leaky integrate-and-fire neuron model is adopted in this study due to its simplicity and analytical tractability. Each spiking neuron sustains an internal state called membrane potential and continuously integrates afferent spikes from its upstream neurons. Whenever the membrane potential exceeds a certain threshold from below, an output spike will be elicited and transmitted to downstream neurons. After firing, the neuron will undergo a resetting process with a decline on its membrane potential. The neuronal dynamics can be formally described as

$$u_i^l(t) = \sum_{j=1}^{M^{l-1}} w_{ij}^l \sum_{t_j^k \le t} \varkappa\left(t - t_j^k\right) - \vartheta \sum_{t_i^k < t} \varkappa\left(t - t_i^k\right) \tag{1}$$

where $u_i^l(t)$ is the membrane potential of the ith neuron in the lth layer at time t. M^{l-1} represents the number of afferent neurons. w_{ij}^l denotes the synaptic weight from neuron j to i. t_j^k and t_i^k indicate the corresponding input and output spike time, respectively. ϑ denotes the firing threshold. A single-exponential kernel, $\varkappa(t) = \exp(-t/\tau)$, is adopted here due to its efficiency for both processing and learning [36].

Equation (1) is a continuous-time version to precisely describe the dynamics of neuronal states, and it can be easily transformed into a discrete-time description that is commonly used in mainstream deep learning frameworks [8]. Therefore, (1) is discretized with a unit-scale time step, as follows:

$$u_i^l[t] = \exp\left(-\frac{1}{\tau}\right)\left(u_i^l[t-1] - \vartheta o_i^l[t-1]\right) + \sum_{j=1}^{M^{l-1}} w_{ij}^l o_j^{l-1}[t] \tag{2}$$

$$o_i^l[t] = \begin{cases} 1, & \text{if } u_i^l[t] \geq \vartheta \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

where τ is a time constant that governs the decay rate of the membrane potential toward resting. $o_i^l[t]$ represents the output firing status of the neuron, with 1 reflecting fired and 0 for not.

B. Efficient Local Learning Rule

Under the local learning scheme illustrated in Fig. 1, we first propose a new spike-based efficient local learning (ELL) rule. For simplicity and clarity, we detail the proposed rule by exemplifying a hidden layer l as follows.

The lth hidden layer continuously receives input spikes from its preceding one and then transmits useful information with output spikes. Notably, the output spikes are also transmitted to the auxiliary classifier during the training stage. The number of spiking neurons in the classifier is the same as the dimension of the desired outputs. The square error is adopted as the layer-wise loss function to measure the discrepancy between the actual and desired outputs at each time step, as formulated by

$$L = \frac{1}{2}\sum_{t=1}^{N_t} \left\| \boldsymbol{y}[t] - \boldsymbol{o}^{\mathrm{a}}[t] \right\|_2^2 \tag{4}$$

where L denotes the total loss, and N_t represents the whole time interval. The superscript "a" is used to denote the auxiliary classifier, and its output at time step t is represented by $\boldsymbol{o}^{\mathrm{a}}[t]$. $\boldsymbol{y}[t]$ represents the desired output.

According to the neuronal dynamics, synaptic weights can affect the loss L through both immediate neuronal states at each time step and past ones due to the temporal dependency. Such complex dependencies will decrease the training efficiency. Since the neuron's current states contain its preceding ones, we can thus ignore the temporal dependencies for simplicity. The weight changes at each time step t in both the hidden layer and its auxiliary classifier can thus be given as

$$\Delta \boldsymbol{w}^{\mathrm{a}}[t] = -\eta \frac{\partial L}{\partial \boldsymbol{o}^{\mathrm{a}}[t]} \frac{\partial \boldsymbol{o}^{\mathrm{a}}[t]}{\partial \boldsymbol{u}^{\mathrm{a}}[t]} \frac{\partial \boldsymbol{u}^{\mathrm{a}}[t]}{\partial \boldsymbol{w}^{\mathrm{a}}} \tag{5}$$

$$\Delta \boldsymbol{w}^{l}[t] = -\eta \frac{\partial L}{\partial \boldsymbol{o}^{\mathrm{a}}[t]} \frac{\partial \boldsymbol{o}^{\mathrm{a}}[t]}{\partial \boldsymbol{u}^{\mathrm{a}}[t]} \frac{\partial \boldsymbol{u}^{\mathrm{a}}[t]}{\partial \boldsymbol{o}^{l}[t]} \frac{\partial \boldsymbol{o}^{l}[t]}{\partial \boldsymbol{u}^{l}[t]} \frac{\partial \boldsymbol{u}^{l}[t]}{\partial \boldsymbol{w}^{l}} \tag{6}$$

where η denotes a learning rate. Both calculations rely on the gradient of output spike with respect to the corresponding membrane potential. This gradient is zero almost everywhere, which impedes the learning process. A surrogate one is commonly adopted [15], [16] to overcome this problem. Similarly, we choose a surrogate as

$$\frac{\partial \boldsymbol{o}[t]}{\partial \boldsymbol{u}[t]} = \exp(-|\boldsymbol{u}[t] - \vartheta|). \tag{7}$$

Taking this surrogate, the weight changes in (5) and (6) can be available immediately at each time step, and each weight thus can be modified in an efficiently parallel way during the training stage. Fig. 2 demonstrates the flow of local gradients with ELL at the representative time step t. The pseudocodes for ELL are shown in Algorithm 1.

With layer-wise classifiers, our ELL is able to locally train each hidden layer with training labels. In addition, our ELL can continuously adjust weights at each time step. During inference, the auxiliary classifier will be discarded, and input spikes are propagated through the network to generate output ones for a decision.

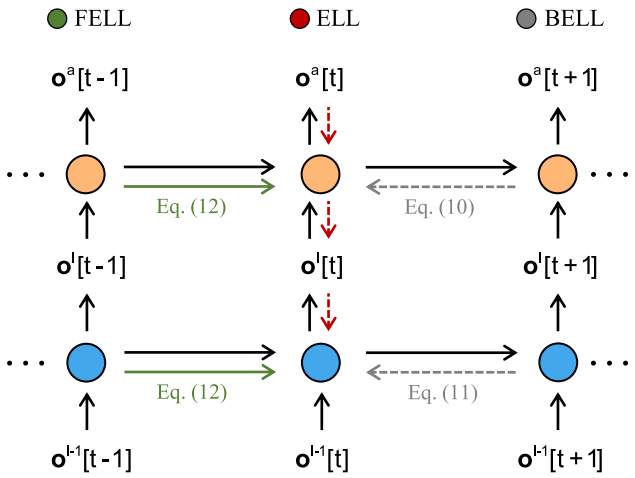


Fig. 2. Illustration of gradient propagations at a representative time step t. ELL takes only the direct influence of weights on the loss at the current time step, which is illustrated with the red lines. Differently, BELL and FELL additionally incorporate temporal dependencies through a local backward (gray) and forward (green) propagation, respectively. Black lines denote the forward information flow in the network.

Algorithm 1: ELL for Training a Hidden Layer

Input : input spikes $\boldsymbol{o}^{l-1}[t]$
target spikes $\boldsymbol{y}[t]$, $t = 1, ..., N_t$
Output: output spikes $\boldsymbol{o}^{l}[t]$, $t = 1, ..., N_t$

```
Initialize weights of the hidden layer and its classifier
for t ← 1 to N_t do
    Compute output spikes o^l[t], o^a[t] with Eq. (2, 3)
    Compute Δw^a[t] and Δw^l[t] using Eq. (5, 6)
    Update all weights based on their changes (could
    also be modified to accumulate a certain number of
    changes and then update.)
end for
return (o^l[t])_{t=1,...,N_t}
```

C. Backward ELL Rule

Despite its simplicity and efficiency, ELL neglects the inherently temporal dependencies of neuronal dynamics that could limit the performance under certain conditions. Following the routine of a typical BP algorithm, we propose BELL (with "B" standing for "backward") to obtain accurate gradients of the loss with respect to weights.

Applying the gradient descent with (4), the derivatives of the overall loss with respect to weights of both the hidden layer and its auxiliary classifier can be obtained by unrolling the loss into each relevant component through both space and time, as formulated by

$$\Delta \boldsymbol{w}^{\mathrm{a}} = -\eta \frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{w}^{\mathrm{a}}} = -\eta \sum_{t=1}^{N_t} \frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t]} \frac{\partial \boldsymbol{u}^{\mathrm{a}}[t]}{\partial \boldsymbol{w}^{\mathrm{a}}} \tag{8}$$

Algorithm 2: BELL for Training a Hidden Layer

Input : input spikes $\boldsymbol{o}^{l-1}[t]$
target spikes $\boldsymbol{y}[t]$, $t = 1, ..., N_t$
Output: output spikes $\boldsymbol{o}^{l}[t]$, $t = 1, ..., N_t$

Initialize weights of the hidden layer and its classifier
for $t \leftarrow 1$ **to** N_t **do**
  Compute output spikes $\boldsymbol{o}^{l}[t], \boldsymbol{o}^{\mathrm{a}}[t]$ with Eq. (2, 3)
end for
Compute L using Eq. (4) with $(\boldsymbol{o}^{\mathrm{a}}[t], \boldsymbol{y}[t])_{t=1,...,N_t}$
for $t \leftarrow N_t$ **to** 1 **do**
  Compute $\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t]}$ using Eq. (10)
  Compute $\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{u}^{l}[t]}$ using Eq. (11)
end for
Compute $\Delta\boldsymbol{w}^{\mathrm{a}}$ and $\Delta\boldsymbol{w}^{l}$ using Eq. (8, 9)
Update all weights based on their changes
return $(\boldsymbol{o}^{l}[t])_{t=1,...,N_t}$

$$\Delta\boldsymbol{w}^{l} = -\eta\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{w}^{l}} = -\eta\sum_{t=1}^{N_t}\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{u}^{l}[t]}\frac{\partial\boldsymbol{u}^{l}[t]}{\partial\boldsymbol{w}^{l}}. \tag{9}$$

In the auxiliary classifier, the membrane potential at the current time can also affect the loss through the next one. Taking this temporal dependency, the derivative with respect to the membrane potential is given by

$$\begin{aligned}\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t]} &= \frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t+1]}\frac{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t+1]}{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t]} + \frac{\partial L}{\partial\boldsymbol{o}^{\mathrm{a}}[t]}\frac{\partial\boldsymbol{o}^{\mathrm{a}}[t]}{\partial\boldsymbol{u}^{\mathrm{a}}[t]} \\ &= \frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t+1]}\left(\frac{\partial\boldsymbol{u}^{\mathrm{a}}[t+1]}{\partial\boldsymbol{u}^{\mathrm{a}}[t]} + \frac{\partial\boldsymbol{u}^{\mathrm{a}}[t+1]}{\partial\boldsymbol{o}^{\mathrm{a}}[t]}\frac{\partial\boldsymbol{o}^{\mathrm{a}}[t]}{\partial\boldsymbol{u}^{\mathrm{a}}[t]}\right) \\ &\quad + \frac{\partial L}{\partial\boldsymbol{o}^{\mathrm{a}}[t]}\frac{\partial\boldsymbol{o}^{\mathrm{a}}[t]}{\partial\boldsymbol{u}^{\mathrm{a}}[t]}.\end{aligned} \tag{10}$$

The derivative with respect to membrane potential at any time step can thus be recursively obtained by a backward propagation in time with (10).

The membrane potential in the hidden layer also has an effect on the potential of its classifier. Correspondingly, the potential derivative can be evaluated as

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{u}^{l}[t]} = \frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{u}^{l}[t+1]}\frac{\mathrm{d}\boldsymbol{u}^{l}[t+1]}{\mathrm{d}\boldsymbol{u}^{l}[t]} + \frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t]}\frac{\partial\boldsymbol{u}^{\mathrm{a}}[t]}{\partial\boldsymbol{o}^{l}[t]}\frac{\partial\boldsymbol{o}^{l}[t]}{\partial\boldsymbol{u}^{l}[t]}. \tag{11}$$

Equation (11) is also a recursive expression. Thus, the derivative with respect to membrane potential at any time step in the hidden layer can be obtained.

Therefore, (8) and (9) can be easily computed with the above equations. Fig. 2 demonstrates the computation of derivatives with BELL at the representative time step t. As compared to ELL, BELL additionally involves the temporal dependencies by a backward flow. The pseudocodes for BELL are shown in Algorithm 2.

D. Forward ELL Rule

In the BELL rule, weights can only be updated at the last time step due to the requirement of the backward flow, resulting in a decrease in training efficiency. It is intriguing whether a local learning rule could incorporate the temporal dependencies with a forward flow. Motivated by eligibility traces [37]–[39], here we propose a new rule named FELL (with "F" standing for "forward").

Algorithm 3: FELL for Training a Hidden Layer

Input : input spikes $\boldsymbol{o}^{l-1}[t]$
target spikes $\boldsymbol{y}[t]$, $t = 1, ..., N_t$
Output: output spikes $\boldsymbol{o}^{l}[t]$, $t = 1, ..., N_t$

Initialize weights of the hidden layer and its classifier
for $t \leftarrow 1$ **to** N_t **do**
  Compute output spikes $\boldsymbol{o}^{l}[t], \boldsymbol{o}^{\mathrm{a}}[t]$ with Eq. (2, 3)
  Compute $\frac{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t]}{\mathrm{d}\boldsymbol{w}^{\mathrm{a}}}$ and $\frac{\mathrm{d}\boldsymbol{u}^{l}[t]}{\mathrm{d}\boldsymbol{w}^{l}}$ with Eq. (12) recursively
  Compute $\Delta\boldsymbol{w}^{\mathrm{a}}[t]$ and $\Delta\boldsymbol{w}^{l}[t]$ using Eq. (13, 14)
  Update all weights based on their changes (could also be modified to accumulate a certain number of changes and then update.)
end for
return $(\boldsymbol{o}^{l}[t])_{t=1,...,N_t}$

We start from the auxiliary classifier. The eligibility trace contains the influence of $\boldsymbol{w}^{\mathrm{a}}$ on $\boldsymbol{u}^{\mathrm{a}}[t]$, that is, $[(\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t])/(\mathrm{d}\boldsymbol{w}^{\mathrm{a}})]$. It can be recursively computed in a forward manner, as given by

$$\frac{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t]}{\mathrm{d}\boldsymbol{w}^{\mathrm{a}}} = \begin{cases}\frac{\partial\boldsymbol{u}^{\mathrm{a}}[t]}{\partial\boldsymbol{w}^{\mathrm{a}}}, & \text{if } t = 1 \\ \frac{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t]}{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t-1]}\frac{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t-1]}{\mathrm{d}\boldsymbol{w}^{\mathrm{a}}} + \frac{\partial\boldsymbol{u}^{\mathrm{a}}[t]}{\partial\boldsymbol{w}^{\mathrm{a}}}, & \text{otherwise.}\end{cases} \tag{12}$$

The weight updates of the classifier at the current time point t can thus be calculated as

$$\Delta\boldsymbol{w}^{\mathrm{a}}[t] = -\eta\frac{\partial L}{\partial\boldsymbol{o}^{\mathrm{a}}[t]}\frac{\partial\boldsymbol{o}^{\mathrm{a}}[t]}{\partial\boldsymbol{u}^{\mathrm{a}}[t]}\frac{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t]}{\mathrm{d}\boldsymbol{w}^{\mathrm{a}}}. \tag{13}$$

Similarly, (12) can be generalized to the hidden layer, that is, $[(\mathrm{d}\boldsymbol{u}^{l}[t])/(\mathrm{d}\boldsymbol{w}^{l})]$. Incorporating such eligibility trace, the weight updates of the hidden layer can be calculated as

$$\Delta\boldsymbol{w}^{l}[t] = -\eta\frac{\partial L}{\partial\boldsymbol{o}^{\mathrm{a}}[t]}\frac{\partial\boldsymbol{o}^{\mathrm{a}}[t]}{\partial\boldsymbol{u}^{\mathrm{a}}[t]}\frac{\partial\boldsymbol{u}^{\mathrm{a}}[t]}{\partial\boldsymbol{o}^{l}[t]}\frac{\partial\boldsymbol{o}^{l}[t]}{\partial\boldsymbol{u}^{l}[t]}\frac{\mathrm{d}\boldsymbol{u}^{l}[t]}{\mathrm{d}\boldsymbol{w}^{l}}. \tag{14}$$

Therefore, our FELL can obtain the temporal-dependent weight gradients with the eligibility traces in a forward manner. The pseudocodes for FELL are shown in Algorithm 3. An illustration of the gradient flow is shown in Fig. 2.

E. Complexity Analyses

SNNs optimized with variants of BPTT [16]–[18] have shown impressive performance on practical tasks. Here, we compare the complexity of BPTT and our proposed methods. Our measurement is based solely on the computational requirements of the gradient computation [40], [41]. The complexity results based on our analyses are summarized in Table I.

As a global learning algorithm, BPTT requires propagating gradients back through both the whole network and the total time interval step by step. Therefore, all intermediate outputs are required to be stored in memory, yielding a memory complexity of $\mathcal{O}(N_l N_t n)$. Here, N_l represents the number of layers in an SNN, and n denotes the maximum number of neurons

TABLE I
COMPARISON OF COMPUTATIONAL COMPLEXITY

Method	Memory	Time
BPTT	$\mathcal{O}(N_l N_t n)$	$\mathcal{O}(N_l N_t n^2)$
RTRL	$\mathcal{O}(N_l n^3)$	$\mathcal{O}(N_l n^4)$
BELL	$\mathcal{O}(N_l N_t n)$	$\mathcal{O}(N_t n^2)$
FELL	$\mathcal{O}(N_l n^2)$	$\mathcal{O}(n^2)$
ELL	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$

in a layer. Similar to BELL, the backward process of BPTT also requires computing the product of the Jacobian matrices in (11). The time complexity of BPTT is determined by the number of multiplications of the Jacobian product and scales with both spatial and temporal dimensions as $\mathcal{O}(N_l N_t n^2)$.

As an online variant of BPTT, real-time recurrent learning (RTRL) [42] is local in time but not local in space. However, RTRL is more computationally expensive. It needs to recursively store and update intermediate derivatives, resulting in $\mathcal{O}(N_l n^3)$ memory and $\mathcal{O}(N_l n^4)$ time complexity [41].

Different from BPTT, our BELL is able to independently update each layer due to layer-wise loss functions. Therefore, the time complexity of BELL is $\mathcal{O}(N_t n^2)$ which is independent of N_l. The memory complexity of BELL is identical to the one of BPTT since intermediate variables are required to be stored until the last time step.

Both our FELL and ELL can operate in a forward and online way. After computing the weight changes, intermediate variables are not required to be kept in memory anymore, therefore removing the factor of the time duration from the memory complexity. The memory complexity of FELL is $\mathcal{O}(N_l n^2)$ due to the requirement of storing the eligibility traces for the subsequent time step. Differently, ELL requires memory to store variables for only the current step, yielding the memory complexity of $\mathcal{O}(n)$. The time complexity of both ELL and FELL is $\mathcal{O}(n^2)$ that is independent of both N_l and N_t due to the online way.

F. Variant Without Weight Transport

The weight transport [43], [44] is considered biologically implausible since feedback weights are required to exactly measure the strengths of their feedforward counterparts. Some alternative strategies have been proposed to circumvent this problem by introducing different feedback weights [45]–[48]. Due to the term $[(\partial \boldsymbol{u}^{\mathrm{a}}[t])/(\partial \boldsymbol{o}^{l}[t])]$ appeared in (6), (11), and (14), the weight transport problem exists in the gradient flow from the local classifier to the hidden layer in all of our proposed ELL, BELL, and FELL rules. Here, we present the extensions of our methods to circumvent the requirement of symmetric weights. We adopt the recently developed Kolen–Pollack (KP) algorithm [48] due to its effectiveness and promising performance. Specifically, a distinct set of feedback weight $\boldsymbol{B}^{\mathrm{a}}$ is used to substitute the transpose of the weight of the local classifier $(\boldsymbol{w}^{\mathrm{a}})^T$, and both $\boldsymbol{B}^{\mathrm{a}}$ and $\boldsymbol{w}^{\mathrm{a}}$ are adjusted during training. Notably, all of our three rules can be extended with KP, and we only show FELL in this study for simplicity. The new obtained method is denoted as FELL-KP. The changes in $\boldsymbol{w}^{\mathrm{a}}$ and $\boldsymbol{B}^{\mathrm{a}}$ at time step t are given as

$$\Delta \boldsymbol{w}^{\mathrm{a}}[t] = -\eta \boldsymbol{\delta}^{\mathrm{a}} \boldsymbol{\sigma}^{l} - \lambda \boldsymbol{w}^{\mathrm{a}} \tag{15}$$

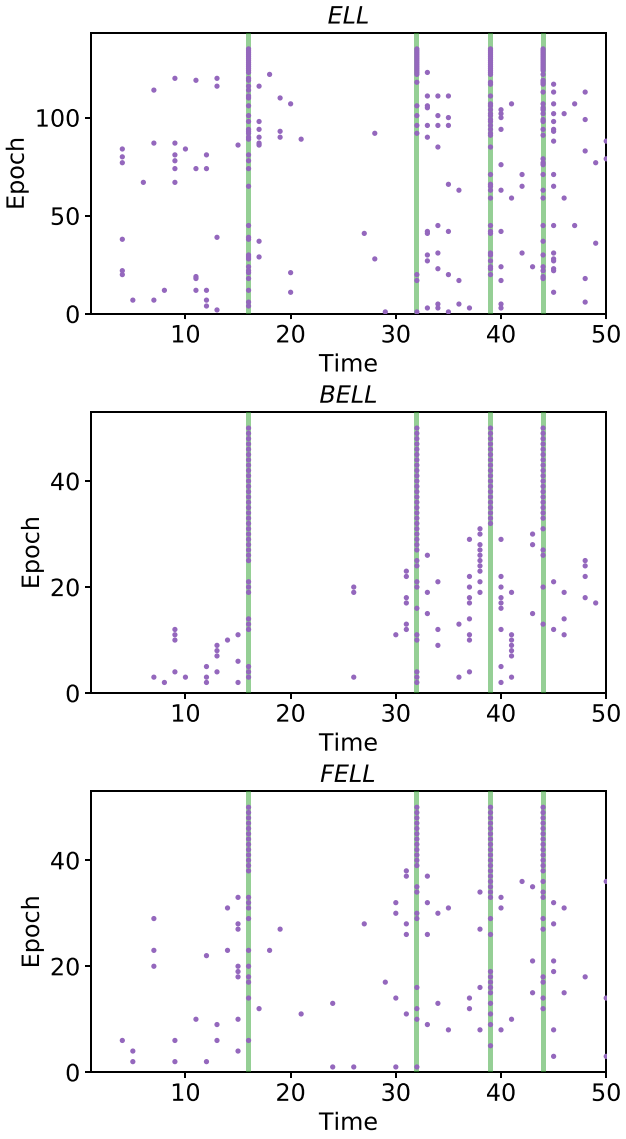


Fig. 3. Learning process in the spike train mapping task. An SNN is trained with the proposed methods to elicit spikes at the desired time points that are denoted by the vertical bars. Each dot represents an actual output spike.

$$\Delta \boldsymbol{B}^{\mathrm{a}}[t] = -\eta \left(\boldsymbol{\sigma}^{l}\right)^{T} \left(\boldsymbol{\delta}^{\mathrm{a}}\right)^{T} - \lambda \boldsymbol{B}^{\mathrm{a}} \tag{16}$$

where $\boldsymbol{\delta}^{\mathrm{a}}$ and $\boldsymbol{\sigma}^{l}$ denote the product of $[\partial L/(\partial \boldsymbol{o}^{\mathrm{a}}[t])][(\partial \boldsymbol{o}^{\mathrm{a}}[t])/(\partial \boldsymbol{u}^{\mathrm{a}}[t])]$ and $[(\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t])/(\mathrm{d}\boldsymbol{w}^{\mathrm{a}})]$ in (13), respectively. λ represents a weight decay factor.

III. EXPERIMENTAL RESULTS

A. Spike Train Mapping

This experiment is devised to show the effectiveness of our proposed methods to produce the desired spike pattern in response to a given input one. The input and desired spike patterns are randomly generated with a Poisson distribution over a time interval of 50. An SNN with a structure of 100-50-1 is used to produce the target spike train.

As is shown in Fig. 3, the output neuron initially fires at arbitrary time points that are quite different from the desired. Along with the learning for all three methods, the undesired output spikes are suppressed and the desired ones gradually

TABLE II
COMPARISON WITH OTHER LEARNING METHODS ON MNIST, FASHION-MNIST, SVHN, AND CIFAR10 DATASETS. BOLD TEXTS INDICATE THE RESULTS OF OUR METHODS. "C," "P," AND A SINGLE DIGIT DENOTE A CONVOLUTIONAL LAYER, AN AVERAGE POOLING LAYER, AND A FULLY CONNECTED LAYER, RESPECTIVELY

Dataset	Learning Scheme	Method	Network	Accuracy
MNIST	Local	STDP [28]	1600-10	95.00%
	Non-local	BA [46]	630-370-10	97.05%
	Non-local	eRBP [47]	500-500-10	97.98%
	Local	**BELL**	**800-10**	**98.59±0.04%**
	Local	**ELL**	**800-10**	**98.61±0.10%**
	Local	**FELL**	**800-10**	**98.62±0.06%**
	Local	STDP+SVM [29]	30c5-p2-100c5-p2-SVM	98.40%
	Non-local	BPTT*	12c5-p2-64c5-p2-10	99.28±0.03%
	Non-local	SLAYER [16]	12c5-p2-64c5-p2-10	99.36±0.05%
	Local	**FELL**	**12c5-p2-64c5-p2-10**	**99.38±0.03%**
	Local	**BELL**	**12c5-p2-64c5-p2-10**	**99.40±0.03%**
	Local	**ELL**	**12c5-p2-64c5-p2-10**	**99.41±0.04%**
Fashion -MNIST	**Local**	**ELL**	**6400-10**	**90.08±0.08%**
	Local	**FELL**	**6400-10**	**90.25±0.04%**
	Local	**BELL**	**6400-10**	**90.86±0.07%**
	Non-local	BPTT*	32c5-p2-64c5-p2-1024-10	91.81±0.16%
	Local	**FELL**	**32c5-p2-64c5-p2-1024-10**	**92.46±0.05%**
	Local	**ELL**	**32c5-p2-64c5-p2-1024-10**	**92.56±0.23%**
	Non-local	TSSL-BP [21]	32c5-p2-64c5-p2-1024-10	92.69±0.09%
	Local	**BELL**	**32c5-p2-64c5-p2-1024-10**	**92.97±0.20%**
SVHN	Local	DECOLLE*[34]	Net1	82.48±0.79%
	Non-local	BPTT*	Net1	93.06±0.19%
	Local	**BELL**	**Net1**	**95.49±0.10%**
	Local	**FELL**	**Net1**	**95.61±0.09%**
	Local	**ELL**	**Net1**	**95.70±0.04%**
	Non-local	Lee's method (BP-based) [20]	Net1	96.06%
CIFAR10	Local	Panda's method [26]	32c5-p2-32c5-p2-64c4-10	70.16%
	Local	DECOLLE*[34]	Net2	74.70±0.68%
	Non-local	STBP [18]	Net2	85.24%
	Non-local	BPTT*	Net2	86.31±0.38%
	Local	**BELL**	**Net2**	**88.01±0.05%**
	Local	**FELL**	**Net2**	**88.36±0.16%**
	Non-local	TSSL-BP [21]	Net2	89.22%
	Local	**ELL**	**Net2**	**89.40±0.14%**
	Non-local	PLIF+BPTT [49]	Net3	93.50%

* This indicates our implementation results to provide a fair comparison under the same network structure
Net1: 64c3-64c3-p2-128c3-128c3-128c3-p2-1024-10
Net2: 96c3-256c3-p2-384c3-p2-384c3-256c3-1024-1024-10
Net3: 256c3-256c3-256c3-p2-256c3-256c3-256c3-p2-2048-100-10

appear within tens of training epochs. This learning result indicates that all of our proposed methods can successfully train an SNN to associate an input pattern with the desired spike train.

B. Image Classification

In this part, we will examine the effectiveness and scalability of our methods with various network structures on practical tasks, including the Modified National Institute of Standards and Technology (MNIST), Fashion-MNIST, as well as the challenging Street View House Numbers (SVHN) and Canadian Institute for Advanced Research (CIFAR10) datasets.

All of our simulations are implemented with the PyTorch framework. To implement the local scheme, we detach the outputs of each hidden layer from the computational graph to prevent backward gradients from the output layer.

The MNIST dataset [50] is a standard benchmark to evaluate the performance of a learning algorithm for SNNs. It contains 60 000 training and 10 000 testing gray images of handwritten digits with a size of 28×28. Serving as a drop-in replacement for MNIST, Fashion-MNIST [51] shares the same image size, number of samples, and dataset separation as MNIST, but consists of different classes of clothing. SVHN [52] is also a digit recognition task, but each sample has a naturalistic background in the RGB format. The standard split of 73 000 training and 26 000 testing samples is used. CIFAR10 [53] contains 60 000 color images with a size of 32×32. For CIFAR10, the standard data augmentation is adopted in the training set, where 4 black pixels are padded on each side of samples followed by a 32×32 crop and a random horizontal flip.

Network structures associated with corresponding datasets are given in Table II. In order to show performance solely attributable to the algorithms, regularization techniques like dropout are not used in our experiments. Pixel values of

images are directly fed into the first hidden layer in order to remove variability [11], [54].

Training setups are configured as follows. We use the Adam optimizer with a reduced schedule of learning rate and a batch size of 100. Other hyperparameters at the training stage are grid-searched for each task (see Table IV in Appendix A). We stop training when the number of epochs reaches a predetermined one. For instance, when FELL is used to learn the MNIST dataset with the fully connected network, the initial learning rate is set to 5×10^{-4} which is then divided by 5 every 15 epochs within a total number of 50 epochs. The target spike trains can be given flexibly. For simplicity, we train the target output neuron to fire at every time step while keeping the others silent. The categorical decision made by the output layer is determined by the neuron that generates the highest spike count. The best testing accuracy in a trial is collected, and we report the averaged best testing accuracy over five different independent trials as the result for each setting. For a fair comparison, we also train the networks with BPTT that adopts the same loss function and surrogate gradient.

The classification results of our proposed methods are shown in Table II along with other spike-based learning approaches. For the relatively simple task of MNIST with a structure of one-hidden layer, our ELL, BELL, and FELL achieve the accuracies of 98.61%, 98.59%, and 98.62%, respectively, all of which outperform the STDP-based method [28] with a significant improvement of over 3% in accuracy. The accuracies are also better than the ones [46], [47] where more waiting time is required as their error signals for hidden layers are projected from the output. Our performance can be further improved by using the advanced convolutional network structure, resulting in better accuracies of 99.41%, 99.40%, and 99.38%, respectively. Our methods consistently achieve good accuracies on the Fashion-MNIST dataset with both the fully connected and convolutional architectures. For instance, our BELL has an accuracy of 92.97% that is even better than the recently reported result with the global BP-based approach [21].

Most of the existing local learning algorithms for deep SNNs mainly examine their performances with shallow networks on MNIST. It remains unexplored whether the local learning scheme can scale up to larger networks for more complicated tasks. Therefore, we evaluate our proposed methods on the challenging tasks, including the SVHN and CIFAR10 datasets. We also train the same network structures on both datasets with DECOLLE [34] to provide baselines. The training setups of DECOLLE are the same as our ELL and FELL for fair comparisons. We find that good accuracies can be achieved with all of our methods. Specifically, ELL, BELL, and FELL reach competitive accuracies compared to the state-of-the-art results obtained by the global BP-based learning methods [20], [21] with the same network architectures. As a comparison, DECOLLE has much lower accuracies with the same network structures on both the SVHN and CIFAR10 datasets. Intriguingly, ELL consistently outperforms FELL and BELL with the large networks on both SVHN and CIFAR10. Its reason could be that the neglected temporal gradients of ELL provide better generalization with the redundant

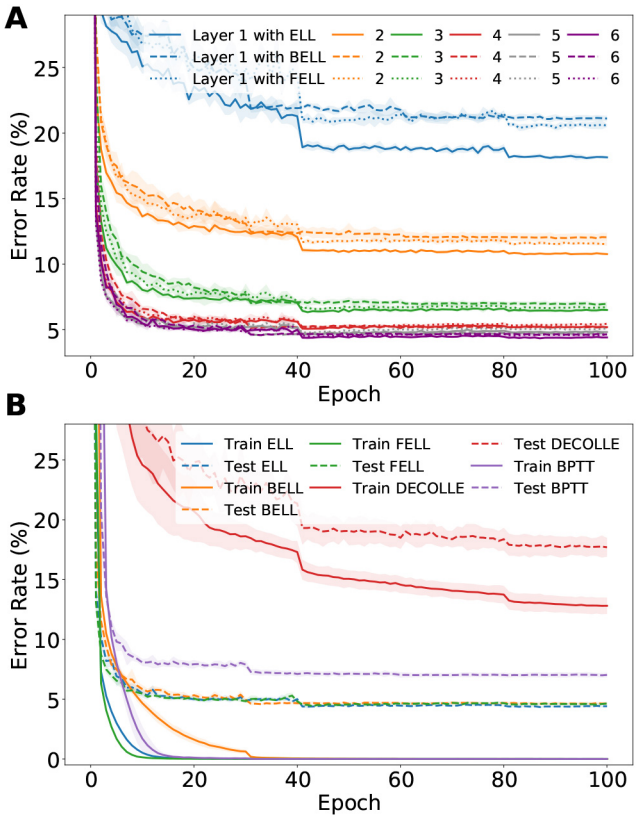


Fig. 4. Visualization of the learning process with Net1 on the SVHN dataset. (a) Test error curves of each layer with our methods. (b) Comparison of training and test error curves of the output layer.

inputs. The advanced performances on the challenging datasets demonstrate the scalability and generalization of our proposed local learning methods. Note that a concurrent work [49] achieves significantly higher accuracy on CIFAR10 by using learnable membrane time constants, regularization techniques, and a larger network. Incorporating such techniques could be a possible way to further improve our performance.

To provide insights into the learning process of our methods, we visualize learning curves with Net1 on the SVHN dataset in Fig. 4. As the training proceeds, the test error of each layer goes down and then converges within tens of epochs. This again demonstrates the effectiveness of our methods. We further compare training and test error curves of the output layer among DECOLLE, BPTT as well as our methods. As is shown in Fig. 4(b), DECOLLE is unable to fit the training data accurately, resulting in a higher error rate on both the training and test set. This weaker performance is mainly attributed to its fixed and random classifiers. As a comparison, the training error of BPTT and our methods are close to zero. However, there is a relatively large divergence between the training and test error, indicating strong overfitting. Our methods are still better than BPTT in terms of test accuracy, reflecting our local learning rules are able to reduce overfitting by searching for solutions that generalize well.

Considering all of the four datasets, our proposed methods consistently outperform other spike-based local learning methods and are competitive with the global BP-based learning

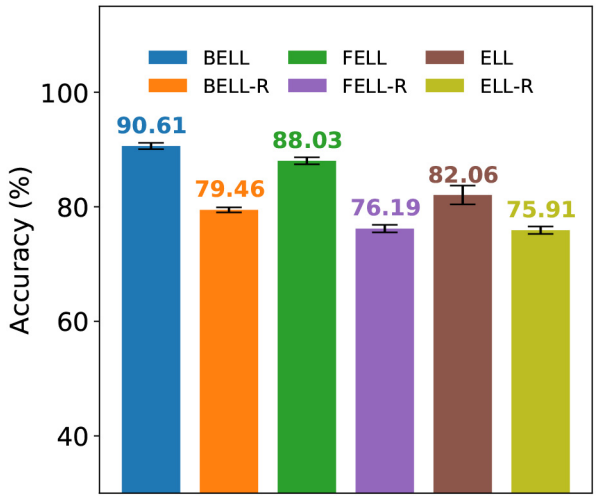


Fig. 5. Stripping analysis of our methods with Net1 on the SVHN dataset under a spike latency code. "R" represents the corresponding variant with random and fixed classifiers.

algorithms which, nevertheless, require more computational resources than ours. To the best of our knowledge, our accuracies are the best among the current spike-based local learning approaches on the four datasets, highlighting our contributions in improving the performance.

C. Latency Encoding Task

In previous experiments, we adopt the commonly used rate code for fair comparisons with baselines. Rate codes normally result in a high accuracy owing to the redundant representation with a high number of spikes, but the temporal characteristic cannot be fully exploited for efficient processing [55]. Here, in order to analyze and compare our proposed methods in depth, we use the more challenging latency encoding to distribute pixel information into the temporal dimension. Specifically, each normalized pixel will be encoded into a spike timing based on a simple linear conversion, as formulated by

$$t_s = \text{round}(N_t - N_t \times I) \tag{17}$$

where t_s is the time of the converted spike, and N_t is the whole time window. I represents the intensity of the input pixel. The operator round(.) returns the closest integer.

As is shown in Fig. 5, BELL reaches a test accuracy of 90.61% on the SVHN dataset under a latency code. To show the importance of the adaptive classifiers, we replace them with random and fixed ones. This leads to severe accuracy degradation to 79.46%, indicating that adaptive auxiliary weights play a nontrivial role in high performance of accuracy. Our result also shows that there is an accuracy gap between BELL and FELL, which is as expected since BELL computes the gradients more accurately through a backward flow than FELL. ELL further suffers more accuracy loss, reflecting the importance to obtain gradients with temporal dependencies. Again, when the adaptive classifier is replaced with a random and fixed one for both FELL and ELL, a great drop in accuracy appears, indicating the advantage of trainable capabilities.

TABLE III
COMPARISON OF THE SPOKEN DIGIT DATASETS

Dataset	Method	Network	Accuracy
TIDIGITS	ReSuMe-DW [59]	Single-layer SNN	92.45%
	Abdollahi's method [60]	AER Silicon Cochlea+SVM	95.58%
	ELL	**20-2048-2048-2048-11**	**95.58±0.23%**
	FELL	**20-2048-2048-2048-11**	**96.70±0.26%**
	Wu's method [58]	SOM+Tempotron	97.60%
	BELL	**20-2048-2048-2048-11**	**98.35±0.11%**
SHD	BPTT [57]	700-128-128-128-20	47.50±2.30%
	ELL	**700-512-1024-512-20**	**62.38±1.55%**
	RSNN [57]	700-128-20	71.40±1.90%
	FELL	**700-512-1024-512-20**	**75.04±0.74%**
	BELL	**700-512-1024-512-20**	**78.24±0.42%**

D. Spoken Digit Classification

The above experiments suggest that the three proposed methods have different levels of performance on the temporal task. In this section, we utilize the TIDIGITS [56] and Spiking Heidelberg Digits (SHD) datasets [57] to further examine their performance since the temporal structure plays an essential role in carrying audio information.

The TIDIGITS dataset consists of spoken digit utterances from "zero" to "nine" and "oh." These utterances are randomly split into 3950 and 1000 samples for training and testing, respectively. We adopt the preprocessing with mel-scaled filter banks to extract features [58]. The SHD dataset contains 10 420 spoken digits in a form of spatiotemporal spike patterns, which are generated from the Heidelberg Digits dataset with biologically inspired models. We adopt the standard preprocessing and separation of 8332 training and 2088 testing samples. The training configurations in this part are the same as our previous experiments, and the hyperparameters are given in Table IV in Appendix A.

The experimental results are consistent with the latency encoding task in Section III-C. As can be seen from Table III, ELL has a relatively lower accuracy than both BELL and FELL on both datasets as it neglects the temporal dependencies for computing gradients. There is also an accuracy gap between BELL and FELL, reflecting the accuracy advantage of BELL by incorporating a more accurate gradient evaluation with backward propagation. Notably, the performances of all of our methods outperform the accuracies of baselines on both datasets. The accuracies of BELL are even better than the one with a self-organizing map (SOM) [58] on TIDIGITS and the one with recurrent SNNs [57] on SHD, respectively. These results demonstrate the accuracy advantage of our local learning rules.

E. Firing Sparsity

Firing sparsity is one of the key features of energy efficiency. We thus evaluate the firing sparsity of deep SNNs trained with our proposed methods. We choose Net1 on the SVHN dataset and Net2 on the CIFAR10 dataset as examples due to their deep structures and the challenge of datasets. Training setups are consistent with Section III-B. We randomly select 500 samples for the trained networks and record the number of neurons according to their output spikes.

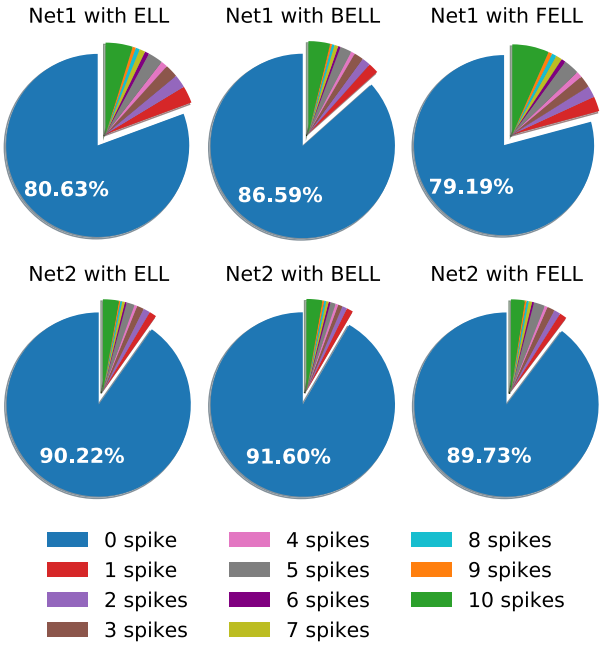


Fig. 6. Visualization of the firing activity of Net1 and Net2 with our methods.

The results are shown with pie charts in Fig. 6. Our ELL and FELL achieve similar sparseness results with around 80% and 90% of neurons being silent under Net1 and Net2, respectively. BELL results in a sparser activity than the other two, the reason for which could be that the accurate gradients could make an efficient use of the available resources. The above results show that a large proportion of neurons trained with our methods are silent all the time, which is favorable for energy-efficient implementations.

F. Influence of Network Depth

All of our proposed methods can adjust each layer individually thanks to the locally available gradients. Could such local learning methods retain hierarchical representations for better performances? We conduct this experiment to examine whether our methods can exploit the power of deep networks. The deep Net2 network on the CIFAR10 dataset is selected in this experiment. During the test, we record the categorical decisions indicated by each layer-wise classifier.

Fig. 7 shows the classification results of each convolutional layer, visualized with confusion matrices. As can be seen, classification accuracy gradually increases with the depth through the network. This demonstrates that all of our methods can successfully take advantage of hierarchical structures for better refinement of classification.

G. Extensions Without Weight Transport

In this experiment, we empirically evaluate the performance of FELL-KP with the deep Net2 network on the CIFAR10 dataset. The classification result is shown in Fig. 8.

As can be seen from the confusion matrix, most of the categories can be recognized with high accuracies. An overall accuracy of 88.14% is obtained on the testing set. As a comparison, FELL has an accuracy of 88.36% (shown in Table II) with the same training configurations. The result indicates that FELL-KP successfully achieves a nearly lossless accuracy.

For further analysis, we measure the similarity between the transpose of the forward matrix $(\boldsymbol{w}^{\mathrm{a}})^T$ and the backward matrix $\boldsymbol{B}^{\mathrm{a}}$ in each auxiliary classifier. The similarity between two vectors, such as $\boldsymbol{x}$ and $\boldsymbol{y}$, is quantified by

$$\cos(\theta) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\| \|\boldsymbol{y}\|}. \tag{18}$$

Fig. 9 shows the similarities between the two matrices during learning. As observed from the figure, in most layers, FELL-KP achieves alignments between the forward and feedback matrices within several epochs. This suggests that FELL-KP can facilitate the match between forward and backward matrices, resulting in a nearly lossless accuracy.

IV. Discussion

This study focuses on a supervised local learning scheme where each layer is locally trained with an auxiliary classifier (illustrated in Fig. 1). Accordingly, we propose three rules, namely, ELL, BELL, and FELL, to adapt each layer and its classifier together. BELL can acquire more accurate gradients owing to a backward propagation, but the changes in weights can be available only after the last time step, leading to the requirement of more computational resources and training delay than the other methods. Differently, ELL and FELL are capable of adjusting weights at every time step in a forward flow by neglecting certain temporal-dependent gradients (detailed in Appendix B), which can alleviate the cost of training time. Notably, FELL is different from RTRL with local classifiers that compute the exact gradient information in an online manner with higher computational cost. FELL is also different from DECOLLE in at least two aspects: one is that FELL adopts trainable classifiers rather than random and fixed ones in DECOLLE; another is that FELL integrates eligibility traces through gradients while DECOLLE depends on a specific neuronal model.

We first validate the effectiveness and scalability of the proposed methods with various network structures based on four vision datasets, including MNIST, Fashion-MNIST, SVHN, and CIFAR10 under a rate code. The results show that all of our methods can scale well and perform better than other spike-based local learning methods with substantially improved accuracies (see Table II), highlighting the advantages of ELL, BELL, and FELL. The better performance of our methods compared to DECOLLE also suggests layer-wise trainable classifiers can facilitate the learning for a better refinement on classification than fixed ones (also see Fig. 5).

On the datasets with a rate code, all our three methods achieve almost similar accuracies. In order to analyze and compare our proposed methods in depth, we perform further evaluations on temporal tasks including SVHN under a latency code, the TIDIGITS and SHD datasets. Experimental results show that BELL achieves the best accuracy while ELL performs the worst among the three (see Fig. 5 and Table III). This suggests when spike timings are used to carry useful information, gradients with temporal dependencies play a nontrivial role in high performance of accuracy.

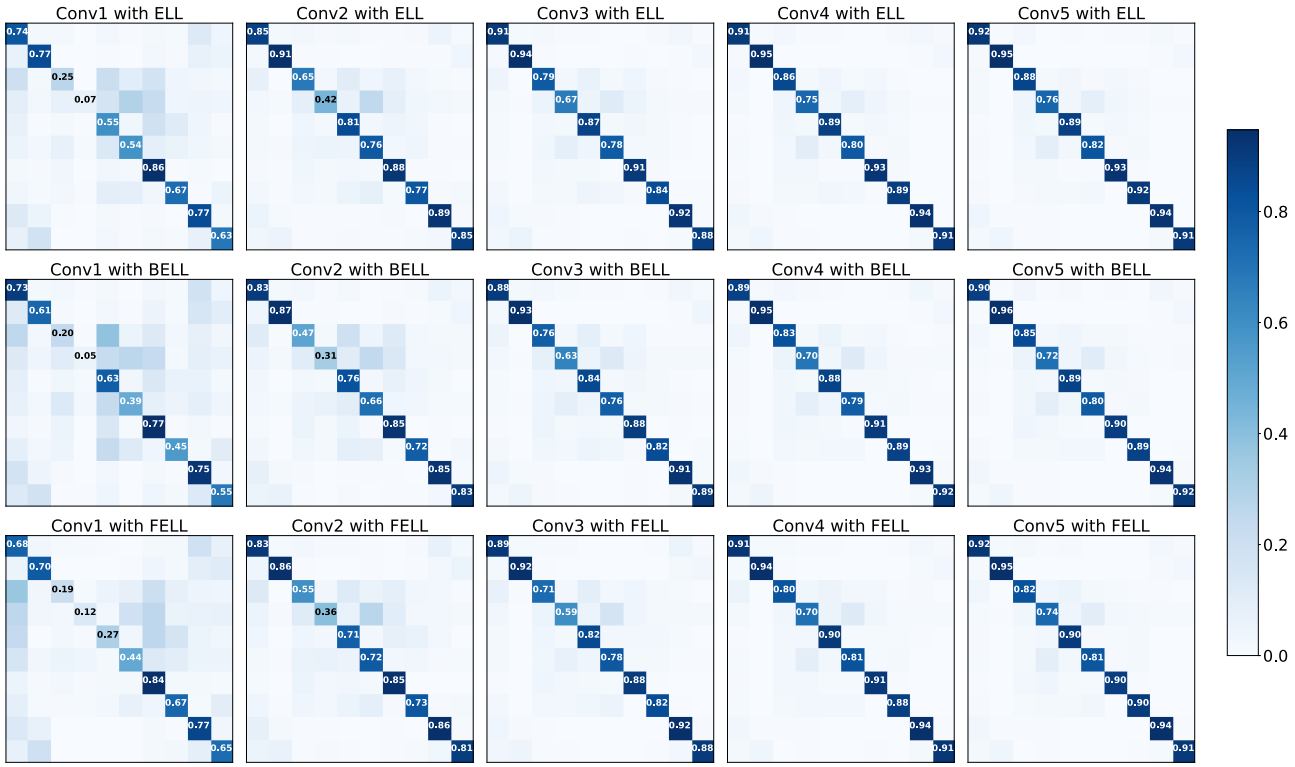


Fig. 7. Confusion matrices as a visualization of layer-wise classification results obtained by auxiliary classifiers with our proposed methods. The Net2 on the CIFAR10 dataset is used here.

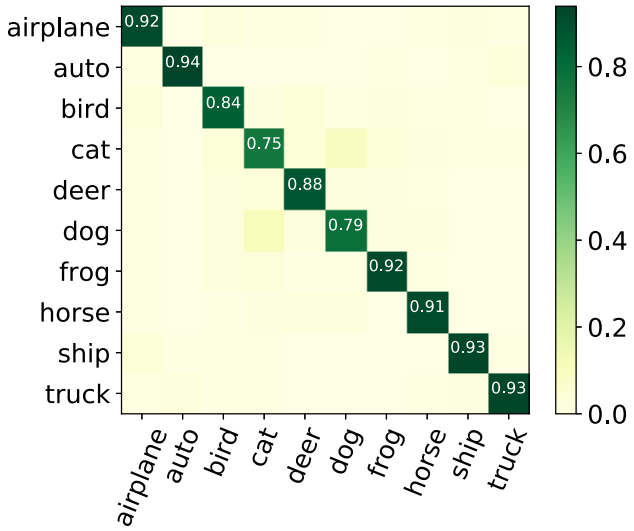


Fig. 8. Classification performance with FELL-KP on CIFAR10.

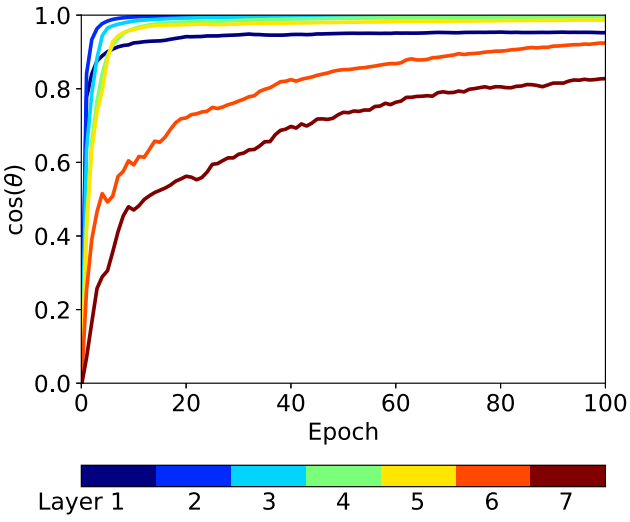


Fig. 9. Similarity of feedforward and feedback matrices in each local layer.

Our proposed rules could be suitable for on-chip learning on neuromorphic hardware since deep SNNs can be efficiently trained with locally available information in a highly parallel way [5], [22], [61]. In addition, the sparsity of our methods, as shown in Fig. 6, indicates their potential merit for high energy efficiency since the power consumption of SNNs is roughly proportional to the number of spikes [9].

We also demonstrate that our local methods can successfully build deep layers to effectively utilize features learned by preceding ones for a better classification (see Fig. 7). This means that our methods can deliver useful signals to downstream layers and exploit the representational power of hierarchical structures.

Despite no feedback between layers, the biologically unrealistic weight transport problem [43], [44] still arises in the layer-wise training with our methods. We present extensions to circumvent this problem by adopting the KP method [48]. Experimental results show that the forward and feedback matrices can be aligned throughout learning (see Fig. 9), and a nearly lossless accuracy can be realized. Our methods thus

TABLE IV
HYPERPARAMETER SETTINGS USED IN DIFFERENT EXPERIMENTS

	Learning Rate		Epochs		τ	ϑ	N_t
	BELL	ELL/FELL	BELL	ELL/FELL	All	All	All
MNIST-DNN	5e-4	5e-4	150	50	1	1	10
MNIST-CNN	5e-4	5e-4	150	50	1	1	10
FMNIST-DNN	5e-4	5e-4	150	50	1	1	10
FMNIST-CNN	3e-3	5e-4	150	50	1	1	10
SVHN	3e-3	5e-4	100	100	1	1	10
CIFAR10	1e-3	5e-4	400	200	1	1	10
Latency-SVHN	5e-4	5e-4	100	100	40	1	10
TIDIGITS	5e-4	5e-4	200	100	40	1	100
SHD	5e-4	5e-4	150	50	40	1	100

can be generalized under certain biologically plausible constraints. The bio-plausible efforts could be favorable as a step to revealing the underlying learning mechanisms in the brain.

V. CONCLUSION

In this work, we proposed ELL, BELL, and FELL to optimize deep SNNs with local gradients generated by layer-wise adaptive classifiers. The BELL integrates more accurate derivatives to modulate weights by a backward flow while resulting in a cost of higher computational overhead than the other methods. The ELL and the FELL are able to continuously change synaptic weights in an online forward flow, yet with a slight sacrifice on accuracy due to the simplified calculations of gradients. We validated the proposed methods with various networks on six datasets. The results showed that all of our proposed rules are advantageous to achieve more accurate performance than other spike-based local learning methods across all examined networks and datasets. We further demonstrated that our methods can be generalized without accuracy loss under certain biologically plausible constraints. Our efforts contribute to improving the performance of spike-based local learning, which could be of great merit for neuromorphic computing. In the future, we would be interested in extending our proposed methods to account for other network models, such as residual connections and recurrent networks.

APPENDIX A
EXPERIMENTAL PARAMETERS

The experimental parameters used in this article are summarized in Table IV.

APPENDIX B
DIFFERENCES IN GRADIENTS

In this part, we detail the gradient differences of both ELL and FELL as compared to BELL.

We first transform the equations of the weight derivatives in BELL into a similar form with the ones in both ELL and FELL. We start from the auxiliary classifier. The weight derivatives in (8) can be decomposed by recursively substituting the voltage derivatives with (10) as

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{w}^{\mathrm{a}}} = \sum_{t=1}^{N_t}\left(\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t+1]}\frac{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t+1]}{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t]} + \frac{\partial L}{\partial \boldsymbol{o}^{\mathrm{a}}[t]}\frac{\partial \boldsymbol{o}^{\mathrm{a}}[t]}{\partial \boldsymbol{u}^{\mathrm{a}}[t]}\right)\frac{\partial \boldsymbol{u}^{\mathrm{a}}[t]}{\partial \boldsymbol{w}^{\mathrm{a}}}$$

$$= \sum_{t=1}^{N_t}\left[\sum_{t'=t+1}^{N_t}\left(\frac{\partial L}{\partial \boldsymbol{o}^{\mathrm{a}}[t']}\frac{\partial \boldsymbol{o}^{\mathrm{a}}[t']}{\partial \boldsymbol{u}^{\mathrm{a}}[t']}\prod_{\hat{t}=t}^{t'-1}\left(\frac{\mathrm{d}\boldsymbol{u}^{a}[\hat{t}+1]}{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[\hat{t}]}\right)\frac{\partial \boldsymbol{u}^{\mathrm{a}}[t]}{\partial \boldsymbol{w}^{\mathrm{a}}}\right)\right] + \sum_{t=1}^{N_t}\frac{\partial L}{\partial \boldsymbol{o}^{\mathrm{a}}[t]}\frac{\partial \boldsymbol{o}^{\mathrm{a}}[t]}{\partial \boldsymbol{u}^{\mathrm{a}}[t]}\frac{\partial \boldsymbol{u}^{\mathrm{a}}[t]}{\partial \boldsymbol{w}^{\mathrm{a}}}. \tag{19}$$

By changing the order of the double summation and then factoring out the common product of $[\partial L/(\partial \boldsymbol{o}^{\mathrm{a}}[t])][(\partial \boldsymbol{o}^{\mathrm{a}}[t])/(\partial \boldsymbol{u}^{\mathrm{a}}[t])]$, (19) can be written as

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{w}^{\mathrm{a}}} = \sum_{t=1}^{N_t}\left\{\frac{\partial L}{\partial \boldsymbol{o}^{\mathrm{a}}[t]}\frac{\partial \boldsymbol{o}^{\mathrm{a}}[t]}{\partial \boldsymbol{u}^{\mathrm{a}}[t]}\left[\frac{\partial \boldsymbol{u}^{\mathrm{a}}[t]}{\partial \boldsymbol{w}^{\mathrm{a}}} + \sum_{t'=1}^{t-1}\left(\prod_{\hat{t}=t'}^{t-1}\left(\frac{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[\hat{t}+1]}{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[\hat{t}]}\right)\frac{\partial \boldsymbol{u}^{\mathrm{a}}[t']}{\partial \boldsymbol{w}^{\mathrm{a}}}\right)\right]\right\} \tag{20}$$

where the term in square brackets can be recursively computed with a similar form like the eligibility trace in (12). Equation (20) thus can be transformed into

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{w}^{\mathrm{a}}} = \sum_{t=1}^{N_t}\frac{\partial L}{\partial \boldsymbol{o}^{\mathrm{a}}[t]}\frac{\partial \boldsymbol{o}^{\mathrm{a}}[t]}{\partial \boldsymbol{u}^{\mathrm{a}}[t]}\frac{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t]}{\mathrm{d}\boldsymbol{w}^{\mathrm{a}}}. \tag{21}$$

Comparing (21) with (5) and (13), we can find that both ELL and FELL relegate the gradient computations to a simplified form accordingly.

Following the same routine, the weight derivatives in the hidden layer in (9) can be transformed into

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{w}^{l}} = \sum_{t=1}^{N_t}\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{u}^{\mathrm{a}}[t]}\frac{\partial \boldsymbol{u}^{\mathrm{a}}[t]}{\partial \boldsymbol{o}^{l}[t]}\frac{\partial \boldsymbol{o}^{l}[t]}{\partial \boldsymbol{u}^{l}[t]}\frac{\mathrm{d}\boldsymbol{u}^{l}[t]}{\mathrm{d}\boldsymbol{w}^{l}}. \tag{22}$$

The corresponding comparison shows that FELL ignores the gradients relying on future membrane potentials of the classifier by using $[\partial L/(\partial \boldsymbol{u}^{\mathrm{a}}[t])]$, while ELL neglects all of the temporal-dependent gradients.

REFERENCES

[1] E. R. Kandel *et al.*, *Principles of Neural Science*, vol. 4. New York, NY, USA: McGraw-Hill, 2000.

[2] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*, vol. 806. Cambridge, MA, USA: MIT Press, 2001.

[3] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychol. Rev.*, vol. 65, no. 6, p. 386, 1958.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[5] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019.

[6] J. D. Victor and K. P. Purpura, "Metric-space analysis of spike trains: Theory, algorithms and application," *Netw. Comput. Neural Syst.*, vol. 8, no. 2, pp. 127–164, 1997.

[7] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Netw.*, vol. 10, no. 9, pp. 1659–1671, 1997.

[8] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural Netw.*, vol. 111, pp. 47–63, Mar. 2019.

[9] 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, 2015.

[10] 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)*, 2015, pp. 1–8.

[11] 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," *Front. Neurosci.*, vol. 11, p. 682, Dec. 2017.

[12] Q. Yu, C. Ma, S. Song, G. Zhang, J. Dang, and K. C. Tan, "Constructing accurate and efficient deep spiking neural networks with double-threshold and augmented schemes," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 4, pp. 1714–1726, Apr. 2022, doi: 10.1109/TNNLS.2020.3043415.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[14] S. M. Bohte, J. N. Kok, and H. La Poutré, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, nos. 1–4, pp. 17–37, 2002.

[15] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks," *IEEE Signal Process. Mag.*, vol. 36, no. 6, pp. 51–63, Nov. 2019.

[16] S. B. Shrestha and G. Orchard, "SLAYER: Spike layer error reassignment in time," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, 2018, pp. 1412–1421.

[17] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990.

[18] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Front. Neurosci.*, vol. 12, p. 331, May 2018.

[19] P. Gu, R. Xiao, G. Pan, and H. Tang, "STCA: Spatio-temporal credit assignment with delayed feedback in deep spiking neural networks," in *Proc. IJCAI*, 2019, pp. 1366–1372.

[20] C. Lee, S. S. Sarwar, P. Panda, G. Srinivasan, and K. Roy, "Enabling spike-based backpropagation for training deep neural network architectures," *Front. Neurosci.*, vol. 14, p. 119, Feb. 2020.

[21] W. Zhang and P. Li, "Temporal spike sequence learning via backpropagation for deep spiking neural networks," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 12011–12022.

[22] F. Zenke and E. O. Neftci, "Brain-inspired learning on Neuromorphic substrates," *Proc. IEEE*, vol. 109, no. 5, pp. 935–950, May 2021, doi: 10.1109/JPROC.2020.3045625.

[23] D. O. Hebb, *The Organization of Behavior: A Neuropsycholocigal Theory* (A Wiley Book in Clinical Psychology), vol. 62. New York, NY, USA: Wiley, 1949, p. 78.

[24] G.-Q. Bi and M.-M. Poo, "Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type," *J. Neurosci.*, vol. 18, no. 24, pp. 10464–10472, 1998.

[25] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Front. Neurosci.*, vol. 12, p. 774, Oct. 2018.

[26] P. Panda and K. Roy, "Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2016, pp. 299–306.

[27] B. Illing, W. Gerstner, and J. Brea, "Biologically plausible deep learning—But how far can we go with shallow networks?" *Neural Netw.*, vol. 118, pp. 90–101, Oct. 2019.

[28] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Front. Comput. Neurosci.*, vol. 9, p. 99, Aug. 2015.

[29] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "STDP-based spiking deep convolutional neural networks for object recognition," *Neural Netw.*, vol. 99, pp. 56–67, Mar. 2018.

[30] D. Liu and S. Yue, "Event-driven continuous STDP learning with deep structure for visual pattern recognition," *IEEE Trans. Cybern.*, vol. 49, no. 4, pp. 1377–1390, Apr. 2019.

[31] E. S. Marquez, J. S. Hare, and M. Niranjan, "Deep cascade learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 11, pp. 5475–5485, Nov. 2018.

[32] H. Mostafa, V. Ramesh, and G. Cauwenberghs, "Deep supervised learning using local errors," *Front. Neurosci.*, vol. 12, p. 608, Aug. 2018.

[33] A. Nøkland and L. H. Eidnes, "Training neural networks with local error signals," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 4839–4850.

[34] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic plasticity dynamics for deep continuous local learning (DECOLLE)," *Front. Neurosci.*, vol. 14, p. 424, May 2020.

[35] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge, U.K.: Cambridge Univ. Press, 2002.

[36] Q. Yu, S. Li, H. Tang, L. Wang, J. Dang, and K. C. Tan, "Toward efficient processing and learning with spikes: New approaches for multispike learning," *IEEE Trans. Cybern.*, vol. 52, no. 3, pp. 1364–1376, Mar. 2022, doi: 10.1109/TCYB.2020.2984888.

[37] G. Bellec *et al.*, "A solution to the learning dilemma for recurrent networks of spiking neurons," *Nat. Commun.*, vol. 11, p. 3625, Jul. 2020.

[38] 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, 2013, Art. no. e78318.

[39] F. Zenke and S. Ganguli, "Superspike: Supervised learning in multilayer spiking neural networks," *Neural Comput.*, vol. 30, no. 6, pp. 1514–1541, 2018.

[40] R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity," in *Back-Propagation: Theory, Architectures and Applications*. Hillsdale, NJ, USA: Erlbaum, 1995, ch. 13, pp. 433–486.

[41] O. Marschall, K. Cho, and C. Savin, "A unified framework of online learning algorithms for training recurrent neural networks," *J. Mach. Learn. Res.*, vol. 21, no. 135, pp. 1–34, 2020.

[42] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Comput.*, vol. 1, no. 2, pp. 270–280, Jun. 1989.

[43] S. Grossberg, "Competitive learning: From interactive activation to adaptive resonance," *Cogn. Sci.*, vol. 11, no. 1, pp. 23–63, 1987.

[44] F. Crick, "The recent excitement about neural networks," *Nature*, vol. 337, no. 6203, pp. 129–132, 1989.

[45] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nat. Commun.*, vol. 7, no. 1, p. 13276, 2016.

[46] A. Samadi, T. P. Lillicrap, and D. B. Tweed, "Deep learning with dynamic spiking neurons and fixed feedback weights," *Neural Comput.*, vol. 29, no. 3, pp. 578–602, Mar. 2017.

[47] E. O. Neftci, C. Augustine, S. Paul, and G. Detorakis, "Event-driven random back-propagation: Enabling neuromorphic deep learning machines," *Front. Neurosci.*, vol. 11, p. 324, Jun. 2017.

[48] M. Akrout, C. Wilson, P. C. Humphreys, T. Lillicrap, and D. B. Tweed, "Deep learning without weight transport," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, 2019, pp. 976–984.

[49] W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang, and Y. Tian, "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 2661–2671.

[50] 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.

[51] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*.

[52] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2011, pp. 1–9.

[53] A. Krizhevsky, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, USA, Rep. TR-2009, 2009. [Online]. Available: http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[54] Q. Yu, S. Song, C. Ma, L. Pan, and K. C. Tan, "Synaptic learning with augmented spikes," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 3, pp. 1134–1146, Mar. 2022, doi: 10.1109/TNNLS.2020.3040969.

[55] Q. Yu, H. Li, and K. C. Tan, "Spike timing or rate? neurons learn to make decisions for both through threshold-driven plasticity," *IEEE Trans. Cybern.*, vol. 49, no. 6, pp. 2178–2189, Jun. 2019.

[56] R. G. Leonard and G. Doddington, *Tidigits Speech Corpus*, Texas Instrum., Inc., Dallas, TX, USA, 1993.

[57] B. Cramer, Y. Stradmann, J. Schemmel, and F. Zenke, "The Heidelberg spiking data sets for the systematic evaluation of spiking neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Dec. 30, 2020, doi: 10.1109/TNNLS.2020.3044364.

[58] J. Wu, Y. Chua, and H. Li, "A biologically plausible speech recognition framework based on spiking neural networks," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2018, pp. 1–8.

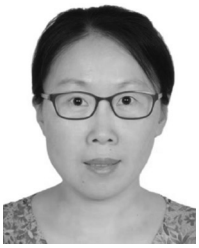
[59] M. Zhang *et al.*, "Supervised learning in spiking neural networks with synaptic delay-weight plasticity," *Neurocomputing*, vol. 409, pp. 103–118, Oct. 2020.

[60] M. Abdollahi and S.-C. Liu, "Speaker-independent isolated digit recognition using an AER silicon cochlea," in *Proc. IEEE Biomed. Circuits Syst. Conf. (BioCAS)*, 2011, pp. 269–272.

[61] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan./Feb. 2018.

Chenxiang Ma received the B.Eng. degree in computer science from the China University of Petroleum, Qingdao, China, in 2019. He is currently pursuing the master's degree with the College of Intelligence and Computing, Tianjin University, Tianjin, China.

His current research interests include learning algorithms in spiking neural networks and deep learning.

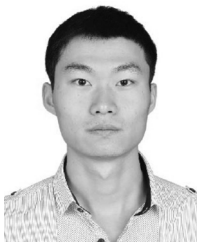


Rui Yan (Member, IEEE) received the bachelor's and master's degrees from the Department of Mathematics, Sichuan University, Chengdu, China, in 1998 and 2001, respectively, and the Ph.D. degree from the Department of Electrical and Computer Engineering, National University of Singapore, Singapore, in 2006.

She was a Postdoctoral Research Fellow with the University of Queensland, Brisbane, QLD, Australia, from 2006 to 2008, and a Research Scientist with the Institute for Infocomm Research, A*STAR, Singapore, from 2009 to 2014. She was a Professor with the College of Computer Science, Sichuan University from 2014 to 2020. She is currently a Professor with the College of Computer Science, Zhejiang University of Technology, Hangzhou, China. Her current research interests include intelligent robots, brain-inspired computing, and cognitive systems.

Zhaofei Yu received the B.S. degree from the Hong Shen Honors School, College of Optoelectronic Engineering, Chongqing University, Chongqing, China, in 2012, and the Ph.D. degree from the Automation Department, Tsinghua University, Beijing, China, in 2017.

He is an Assistant Professor with the Institute for Artificial Intelligence, Peking University, Beijing. His current interests include artificial intelligence, brain-inspired computing, and computational neuroscience.



Qiang Yu (Senior Member, IEEE) received the B.Eng. degree in electrical engineering and automation from the Harbin Institute of Technology, Harbin, China, in 2010, and the Ph.D. degree in electrical and computer engineering from the National University of Singapore, Singapore, in 2014.

He is currently an Associate Professor with the College of Intelligence and Computing, Tianjin University, Tianjin, China, and also with Peng Cheng Laboratory, Shenzhen, China. His current research interests include learning algorithms in spiking neural networks, neural coding, cognitive computations, and machine learning.

Dr. Yu was a recipient of the 2016 IEEE Outstanding TNNLS Paper Award.