Accurate online training of dynamical spiking neural networks through Forward Propagation Through Time

Bojian Yin[1,*], Federico Corradi[2,3], and Sander M. Bohté[1,4,5]

[1]CWI, Machine Learning group, Amsterdam, The Netherlands
[2]Eindhoven University of Technology, Electrical Engineering, Eindhoven, The Netherlands
[3]Stichting IMEC Netherlands, Holst Centre, Eindhoven, The Netherlands
[4]Univ of Amsterdam, Faculty of Science, Amsterdam, The Netherlands
[5]Rijksuniversiteit Groningen, Faculty of Science and Engineering, Groningen, The Netherlands
[*]byin@cwi.nl (corresponding author)

ABSTRACT

With recent advances in learning algorithms, recurrent networks of spiking neurons are achieving performance competitive with vanilla recurrent neural networks. Still, these algorithms are limited to small networks of simple spiking neurons and modest-length temporal sequences, as they impose high memory requirements, have difficulty training complex neuron models, and are incompatible with online learning. Here, we show how recently developed 'Forward-Propagation-Through-Time' (FPTT) learning combined with novel Liquid Time-Constant spiking neurons resolves these limitations. Applying FPTT to networks of such complex spiking neurons, we demonstrate online learning of exceedingly long sequences while outperforming current online methods and approaching or outperforming offline methods on temporal classification tasks. FPTT's efficiency and robustness furthermore enables us to directly train a deep and performant spiking neural network for joint object localization and recognition, demonstrating the ability to train large-scale dynamic and complex spiking neural network architectures.
With recent advances in learning algorithms, recurrent networks of spiking neurons are achieving performance competitive with vanilla recurrent neural networks. Still, these algorithms are limited to small networks of simple spiking neurons and modest-length temporal sequences, as they impose high memory requirements, have difficulty training complex neuron models, and are incompatible with online learning. Here, we show how recently developed 'Forward-Propagation-Through-Time' (FPTT) learning combined with novel Liquid Time-Constant spiking neurons resolves these limitations. Applying FPTT to networks of such complex spiking neurons, we demonstrate online learning of exceedingly long sequences while outperforming current online methods and approaching or outperforming offline methods on temporal classification tasks. FPTT's efficiency and robustness furthermore enables us to directly train a deep and performant spiking neural network for joint object localization and recognition, demonstrating the ability to train large-scale dynamic and complex spiking neural network architectures.

Introduction

The binary, event-driven and sparse nature of communication between spiking neurons in the brain holds great promise for flexible and energy-efficient AI. Recent work has demonstrated effective and efficient performance from spiking neural networks (SNNs)[1], enabling competitive and energy-efficient applications in neuromorphic hardware[2] and novel means of investigating biological neural architectures[3,4]. This success stems principally from the use of approximating surrogate gradients[5,6] to integrate networks of spiking neurons into auto differentiating frameworks like Tensorflow and Pytorch[7], enabling the application of standard learning algorithms and in particular Back-Propagation Through Time (BPTT).

The imprecision of the surrogate gradient approach however expounds on the existing drawbacks of BPTT. In particular, BPTT has a linearly increasing memory cost as a function of sequence length T, $\Omega(T)$ and suffers from vanishing or exploding backpropagating gradients, which limits its applicability on long time sequences[8] and large-scale SNN models[9]. Alternative approaches like real-time recurrent learning (RTRL)[10] similarly exhibit excessive computational complexity, and low complexity approximations to BPTT like e-prop[11] or OSTL[12] at best approach BPTT performance. Training on long temporal sequences in SNNs is of particular importance when the tasks require a high temporal resolution, for instance to match the physical characteristics of low-latency clock-less neuromorphic hardware[2,13].

Kag et al.[8] recently introduced an algorithm for online learning in recurrent networks, Forward Propagation Through Time (FPTT), demonstrating better generalization on many temporal classification benchmark tasks compared to BPTT. In particular, FPTT improved over BPTT on long sequence training in Long Short-Term Memory networks (LSTMs). FPTT differs from BPTT in that it does not calculate a gradient through time, and instead considers learning-through-time as a coordinated consensus problem. Using regularized synaptic tags, FPTT enables immediate, online learning in RNNs similar to feedforward networks, eliminating the problematic dependence of the gradient calculation in BPTT on the products of partial gradients along the time dimension: FPTT exhibits linear $\Omega(T)$ computational cost per sample. For training recurrent SNNs however, as we demonstrate, a straightforward application of FPTT on long sequence training does not improve performance as it does in[8], and we observed this also with vanilla RNNs. Therefore, we deduce that FPTT particularly benefits from the gating structure inherent in LSTM-style gated RNNs, which is lacking in vanilla RNNs and SRNNs.

Taking inspiration from the concept of Liquid Time-Constant (LTCs)[14], we introduce a class of spiking neurons, the Liquid Time-Constant Spiking Neuron (LTC-SN), where time-constants internal to the neuron are dynamic and input-driven in a learned fashion, resulting in functionality similar to the gating operation in LSTMs. We integrate these neurons in networks that are trained with FPTT and demonstrate that the resulting LTC-SNNs outperform various SNNs trained with BPTT on long sequences while enabling online learning and drastically reducing memory complexity. We show this for several classical benchmarks that can easily be varied in sequence length, like the Add-task and the DVS-GESTURE benchmark[15,16]. We also show how FPTT-trained LTC-SNNs can be applied to large convolutional SNNs, where we demonstrate state-of-the-art for online learning in SNNs on a number of standard benchmarks (S-MNIST, R-MNIST, DVS-GESTURE) and to near (Fashion-MNIST, DVS-CIFAR10) or exceeding (PS-MNIST, R-MNIST) state-of-the-art performance as obtained with offline BPTT.

Finally, the training and memory efficiency of FPTT enables us to directly train SNNs in an end-to-end manner at network sizes and complexity that was previously infeasible. We demonstrate this in a large-scale You-Look-Only-Once (YOLO) LTC-SNN architecture for object detection on the Pascal Visual Object Classes (Pascal VOC) dataset[17]. Object detection is a challenging task, as it involves accurate multi-object identification and precise bounding box coordinate computation; previous SNN approaches have been limited to either ANN-to-SNN conversions[18–20], requiring many thousands of time-steps at inference time, or small scale and inefficient SNNs with performance far removed from that of modern ANNs [21]. Our FPTT-trained YOLOv4[22] implementation – SPYv4 – uses 21 layers, 6.2M LTC spiking neurons, and 14M parameters to achieve state-of-the-art for SNNs, exceeding the performance of converted ANNs while achieving extremely low latency.

With FPTT and LTC spiking neurons, we demonstrate end-to-end online training of large and high-performance SNNs comprised of complex spiking neuron models that were previously infeasible.

Related Work

The problem of training recurrent neural networks has an extensive history[23–25]. To account for past influences on current activations in a recurrent network, the network can be unrolled, and errors are computed along the paths of the unrolled network. The direct application of error-backpropagation to this unrolled graph is known as Backpropagation-Through-Time[23]. BPTT needs to wait until the last input of a sequence before being able to calculate parameter updates and, as such, cannot be applied in an online manner. Alternative online learning algorithms for RNNs have been developed, including Real-Time Recurrent Learning (RTRL)[10] and approximations thereof[26]. RTRL however is prohibitive in time and memory complexity, and while approximations improve complexity (see[12], and Table 1**a**), they yield variable and task-dependent accuracy deficits compared to exact gradients[11,27].

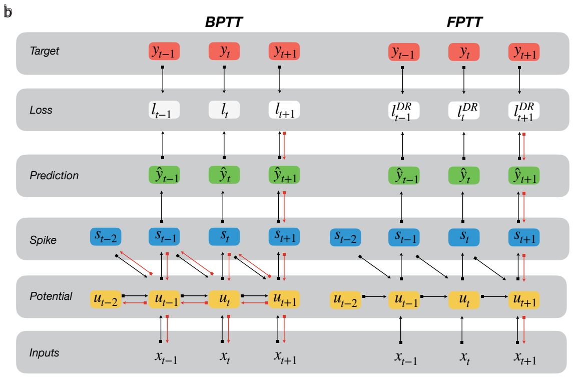
Spiking neural networks are neural networks composed of spiking neurons: stateful neural units that communicate using binary values, i.e. spikes. Their state is determined by current and past inputs, and this state then determines the (binary) value of the emitted output through a spiking mechanism. The discontinuity of the spiking mechanism challenges the application of error-backpropagation, which can be overcome using continuous approximations[5,28], so-called “surrogate gradients”[6]. Recurrent SNNs trained with surrogate gradients and BPTT now achieve competitive performance compared to classical RNNs[1,16,29]. In these and other studies, more intricate spiking neuron models, like those including adaptation, outperformed less complex models such as standard Leaky-Integrate-and-Fire neurons[11]. Additionally, training internal spiking neuron's model parameters like the time-constants of adaptation and membrane-potential decay then further improves performance[1,29].

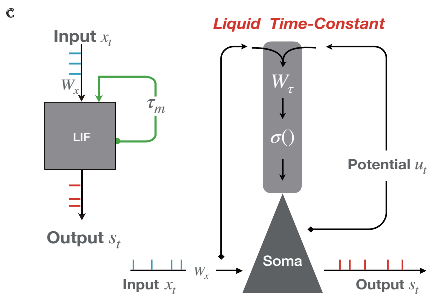
Still, the application of BPTT in SNNs has several drawbacks: in particular, BPTT accumulates the approximation error of surrogate gradients along time. Moreover, the spike-triggered reset of some state variables in typical spiking neuron models (e.g. the membrane potential) causes a vanishing gradient when applying BPTT. We found these effects to be particularly problematic when training networks with complex and more biologically detailed neuron models like Izhikevich and Hodgkin-Huxley models (unpublished). Furthermore, because the SNN training accuracy heavily depends on hyperparameters, obtaining convergence using BPTT in SNNs is non-trivial.

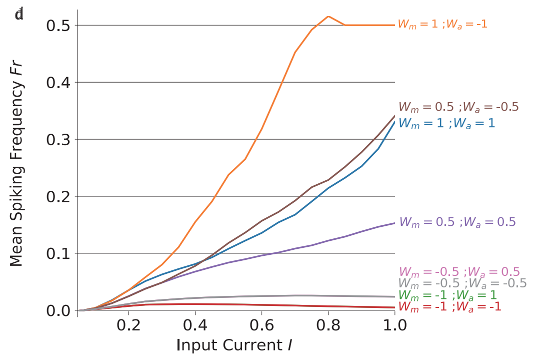
For spiking neural networks, approximations to BPTT like e-prop[11] and Online Spatio-Temporal Learning (OSTL)[12] achieve linear time complexity and have proven effective for many small-scale benchmark problems, ATARI GAMES and also large-scale networks like cortical microcircuits[30]. In terms of trained accuracy, however, none of these approximations have been shown to outperform standard BPTT and, applications to deeper networks use approximate spatial credit assignment approaches like learning-to-learn[11].

a

Algorithm	Gradient Update	parameter Update	Memory Storage
BPTT	$\Omega(c(T)T)$	$\Omega(1)$	$\Omega(NT)$
RTRL	$\Omega(c(T)T^2)$	$\Omega(T)$	$\Omega(NT)$
e-prop / OSTL	$\Omega(c(1)T)$	$\Omega(T)$	$\Omega(N)$
FPTT	$\Omega(c(1)T)$	$\Omega(T)$	$\Omega(N)$







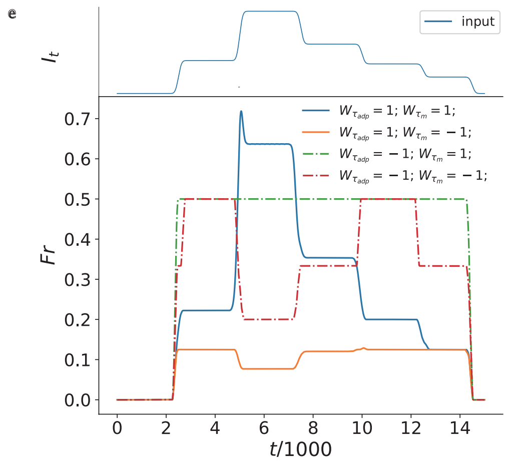


Figure 1. **a**, Computational complexity of gradients, parameter updates and memory storage per sample, with N the batch-size. Computational expense increases as the length T of the sequence grows. **b**, Roll-out of the computational graph of a spiking neuron as used for BPTT (left) and FPTT (right); **c**, Illustration of LIF (left) and LTC (right) spiking neurons as recurrent network structures. **d**, F-I curve of a LTC spiking neuron for combinations of effective LTC weights $W_\tau = \{W_{\tau_m}, W_{\tau_{adp}}\}$: $W_{\tau_m} = \{-1 -0.5, 0.5, 1\}$ and $W_{\tau_{adp}} = \{-1 -0.5, 0.5, 1\}$ associated with respective dynamic time-constant functions σ subject to input current I. **e**, Response firing rate Fr of LTC spiking neuron to varying current input I (top) for combinations of effective LTC weights $W_{\tau_m} = \{-1, 1\}$ and $W_{\tau_{adp}} = \{-1, 1\}$.

FPTT in SNNs

As introduced in[8], FPTT can be implemented directly on the computational graph of SNNs, similar to BPTT approaches. This is illustrated in Fig. 1**b**; the gradient calculation and corresponding algorithms are given in the Methods section and Alg. SA1. FPTT intuitively provides a more robust and efficient gradient approximation for spiking recurrent neural networks than BPTT, as FPTT simplifies the complex gradient computation path in SNNs. This potentially lessens surrogate gradients' cumulative effect, avoiding or reducing the gradient vanishing or explosion problem.

As we will show, FPTT applied directly to SNNs like the Adaptive Spiking Recurrent Neural Networks (ASRNNs)[1] converges but without the learning improvements reported for RNN architectures[8] – and we observed this also for vanilla non-spiking RNNs (not shown). As FPTT was successfully applied to RNNs with gating structures (LSTM), we introduce the Liquid Time-Constant Spiking Neuron (LTC-SN) as a spiking neuron model with a similar gating structure. We observe that in spiking neurons, the time-constant of the membrane potential acts similar to the forget-gate in LSTMs; the LSTM forget-gate however is dynamically controlled by learned functions of inputs. Inspired also by Hasani et al.[14], the LTC-SN's internal time-constants are a learned function of the inputs and hidden states of the network (illustrated in Fig. 1**c**; see also Methods). I.e., for adapting spiking neurons, the time-constant of the membrane potential decay, τ_m and the time-constant of the adaptation decay, τ_{adp} can be made dynamic. A spiking neuron with such varying internal dynamics can respond in flexible and unexpected ways to input currents: as shown in Fig. 1**d**, depending on the effective weights W_τ, the current-spike-frequency response curve can be muted-and-saturating, near-linear, or rapidly increasing-and-then-saturating; when subject to a dynamically varying input current, a higher input current into LTC-SN units can result in a reduced firing rate, and transient dynamics can be absent or present (Fig. 1**e**, samples of trained behavior are shown in Fig. S5).

Experiments

We demonstrate the effectiveness of FPTT training for LTC-SNNs on several well-known temporal classification benchmarks, including the Add-task as in[8] and several established SNN benchmarks (the DVS-GESTURE and DVS-CIFAR10 classification tasks, and the Sequential, Sequential-Permuted, rate-based and Fashion MNIST classification tasks). We moreover demonstrate how the memory efficiency of FPTT enables training large-scale LTC-SNNs for applications like object localization.

The **Add-task**[31] is used to evaluate the ability of RNN architectures to maintain long range memory. An example data point consists of two sequences (x_1, x_2) of length T and a target label y. The sequence x_1 contains values sampled uniformly from [0, 1], x_2 is a binary sequence containing only two 1s, and the label y is the sum of the two entries in the sequence x_1, where $x_2 = 1$. The **IBM DVS Gesture** dataset[15] consists of 11 kinds of hand and arm movements of 29 individuals under three different lighting conditions captured using a DVS128 camera. **DVS-CIFAR10** is a widely used dataset in neuromorphic vision, where the event stream is obtained by displaying moving images of the CIFAR-10 dataset[32]. The **Sequential and Permuted-Sequential** MNIST (S-MNIST, PS-MNIST) datasets were developed to measure sequence recognition and memory capabilities of learning algorithms: the S-MNIST dataset is obtained by reshaping the classical MNIST 28x28 pixel images into a set of one-dimensional sequences consisting of 784 time-steps per sample, where pixels are then sequentially entered one-by-one as input to the network; the PS-MNIST dataset is generated by performing a fixed permutation on the S-MNIST dataset. Theoretically and in practice, PS-MNIST is a more complex classification task than S-MNIST because it lacks temporally correlated patterns. The **rate-coded MNIST** (R-MNIST) is an SNN-specific benchmark where a biologically inspired encoding method is used to generate the network input that produces streaming events (a spike train), by encoding the grey values of the image with Poisson rate-coding[33]. We also applied FPTT-trained LTC-SCNNs to the traditional static **MNIST** and **Fashion-MNIST** datasets for comparison with other models trained offline. Here, we input pixel values directly as injected current into the spiking input layer of the network, repeated 20 times to mimic a constant input stream. For object localization, we used the **PASCAL VOC** benchmark[17] which is comprised of standard 416×416 RBG images with 20 different objects and corresponding bounding box locations.

FPTT-SNN requires Liquid Time-Constant Spiking Neurons. We apply both BPTT and FPTT to the Add-task as originally studied in[8] to illustrate the need for more complex spiking neurons like LTC-SNs when applying FPTT learning. We do this for various network types, including non-spiking LSTMs as a baseline, ASRNNs[1], and LTC-SNNs.

For a long adding sequence of length 1000, example loss-curves are plotted in Fig. 2**a** and averaged converged losses in Fig. 2**b**. We find that as in[8], a standard LSTM network trained with BPTT fails to converge to zero loss, while the same network does converge using FPTT. For SNNs, we find that ASRNNs trained with either FPTT or BPTT do not fully converge, and for the LTC-SNNs trained with BPTT learning rapidly diverges due to exploding gradients. LTC-SNNs trained with FPTT however successfully minimize the loss: ablating the LTC-SN dynamics, we find that the input-dependent dynamic gating of the membrane-potential time-constant is critical for convergence (Tab. S4), and the memory provided by the LTC-SN self-recurrence does in fact suffice for solving this task for shorter sequences, though not for longer ones (Fig. S4).

FPTT allows for longer sequence training. Next, we study the ability of FPTT-trained LTC-SNNs to learn increasingly long sequences. For this, we use the DVS-GESTURE dataset and systematically investigate the performance of a fixed architecture shallow SRNN model on increasingly many frames sampled from the same gesture signals as in[16], ranging from 20 to 1000 frames. For each frame-encoded dataset, we train various networks types for a fixed number of epochs with an identical number of neural units, and report the best performance for BPTT and FPTT trained networks; networks either converged before the final epoch or diverged (Fig S1**a**,**b**).

The results for different sampling frequencies are shown in Fig. 2**c**. Of all methods and networks, the LTC-SNN trained using FPTT achieves the best accuracy in all cases, also outperforming standard (BPTT-trained) LSTMs. Furthermore, the FPTT-trained LTC-SNNs and the ASRNNs exhibit constant accuracy over the whole range of sequence lengths, where the LTC-SNNs consistently outperform the ASRNNs.

In contrast, the accuracy of both LTC-SNNs and ASRNNs trained with BPTT quickly deteriorates as sequence length increases. For the LTC-SNNs, the networks failed to converge for frame-lengths 200 and 500, and best validation accuracy is reported in Fig. 2**c**. For the baseline standard LSTM, this effect is also there, albeit more moderate, as performance decreases from 88.9% at 100 frames to 82.5% at 500 frames. This suggests that indeed the gradient approximation errors in SNNs add up when training with BPTT. The plot also illustrates the memory-intensiveness of BPTT: when applied to the 1000 frame-length data, GPU-memory (24GB) was insufficient for training LSTM, ASRNN and LTC-SNN.

Comparing sparsity (average firing rate) for the different SNN models, we find no meaningful differences (Fig. S1**c**); parameter-matched networks showed similar performance as unit-matched networks, and the inclusion of an auxiliary loss[8]

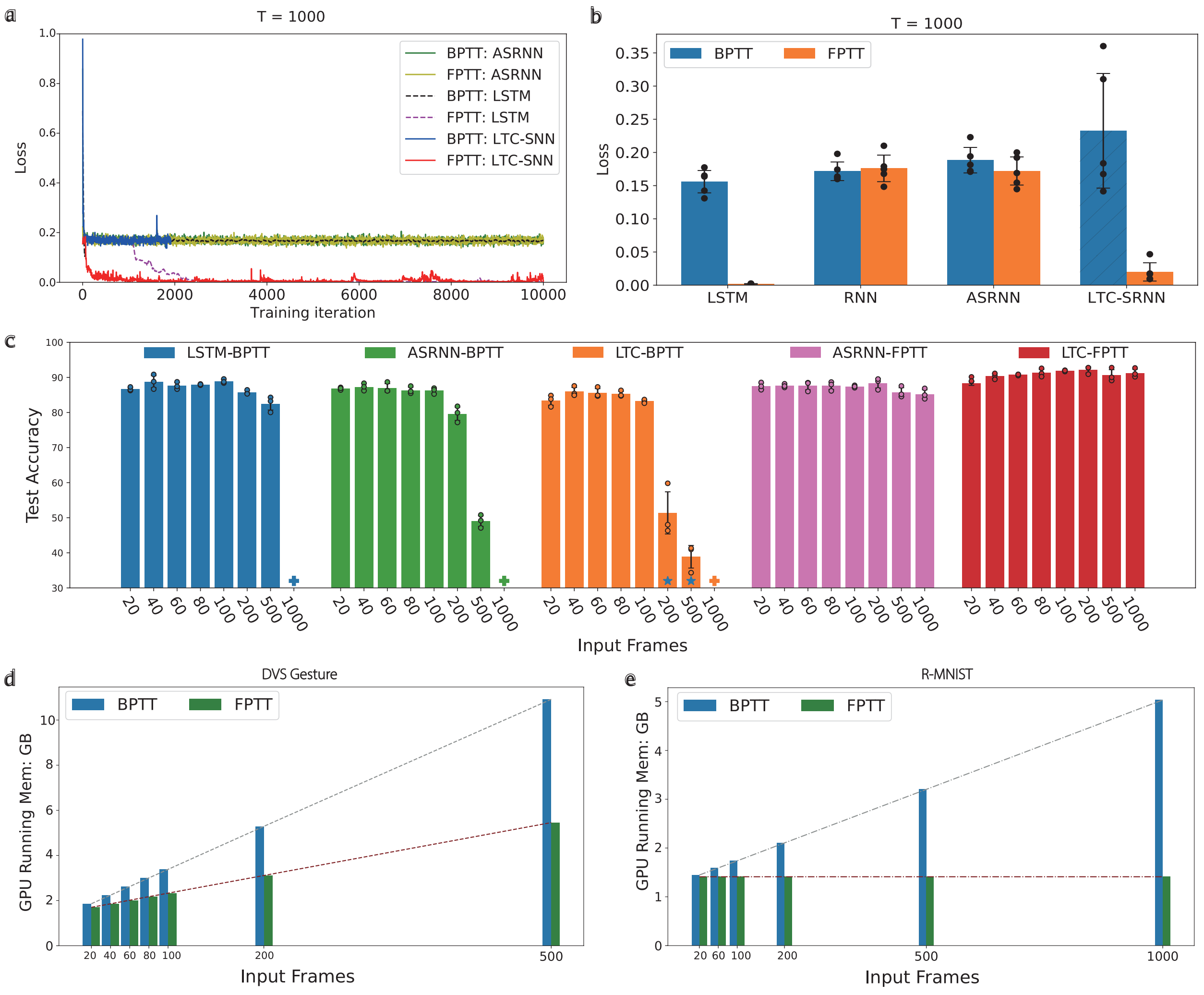


Figure 2. **a**, Example loss-curves for networks trained on the Add-task with sequence length $T = 1000$. The loss of BPTT-trained LTC-SNN becomes *NaN* after 2000 iterations. **b**, corresponding average loss and std of the last 100 training iterations (obtained from 5 runs). **c**, Plot of test accuracy for BPTT and FPTT trained shallow networks on the DVS-GESTURE dataset. Accuracy bar average and std over 3 runs and individual data points are inserted; (*): training diverged; reported accuracy is the best accuracy before divergence; (+): out-of-GPU-memory when training. **d,e**, GPU-memory use when training the DVS-GESTURE dataset (d) or R-MNIST dataset (e) for different sequence lengths with BPTT or FPTT.

only aided BPTT-trained LSTMs but not BPTT-trained SRNNs. Making only the membrane-decay time-constant dynamic has a small negative impact (ASRNN− Table S1).

FPTT Requires Less Memory. FPTT-trained LTC-SNNs require increasingly less memory as sequence length increases as measured on GPU. For the DVS-GESTURE dataset (Fig. 2**d**), FPTT memory-use is both less and increases less rapidly compared to BPTT as a function of the number of frames used per data sample. FPTT's increasing memory use can be attributed to the rapidly inflating size of the frame-encoded dataset, increasing in size from 7.4 GB for 20 frames to 368.1 GB for 1000 frames. We validated this by training an LTC-SNN network on the R-MNIST classification problem, where longer sequences are simulated by showing the same sample for an increasing number of frames. We then find, as expected, that the memory required for FPTT training remains fixed. At the same time, BPTT memory-use linearly increases (Fig. 2**e**).

FPTT with LTC Spiking Neurons improves over Online Approximate BPTT. To demonstrate the power of FPTT as an online training method, we used state-of-the-art deep spiking convolutional network architectures (SCNNs) for standard sequential benchmarks from the literature and trained these architectures with LTC-SN neurons and FPTT.

In Table 1, we compare the LTC-SNNs to existing state-of-the-art online and offline SNNs. We find that LTC-SCNNs

Table 1. Test accuracy of deep SRNNs/SCNNs on various tasks. **Bold-faced** denotes state-of-the-art (SoTa) online performance, ***slanted bold*** denotes overall SNN state-of-the-art. We apply FPTT-trained LTC-SNNs with one recurrent layer comprised of 512 neurons on the (P)S-MNIST tasks. For RMNIST, an identical network structure as in[12] is used, and for the remaining benchmarks the network structures match[16].

Task	**Online baseline**		**This work**		**Offline SoTa**	
S-MNIST	-		FPTT+LTC	**97.37%**	BPTT+ASRNN[1]	***98.7%***
PS-MNIST	-		FPTT+LTC	**93.23%**	BPTT+ASRNN[1]	***94.3%***
RMNIST	OSTL + SNU[12, 34]	95.34%	FPTT+LTC	***98.63%***	BPTT+SNU[34]	97.72%
MNIST	DECOLLE+SNN[35]	98.0%	FPTT+LTC	**99.62%**	BPTT+PLIF[16]	***99.72%***
Fashion MNIST	EMSTDP+SNN[36]	85.3%	FPTT+LTC	**93.58%**	BPTT+PLIF[16]	***94.38%***
DVS-GESTURE	DECOLLE+SNN[37]	95.54%	FPTT+LTC	**97.22%**	BPTT+PLIF[16]	***97.57%***
DVS-CIFAR10	-		FPTT+LTC	**73.2%**	BPTT+PLIF[16]	***74.8%***

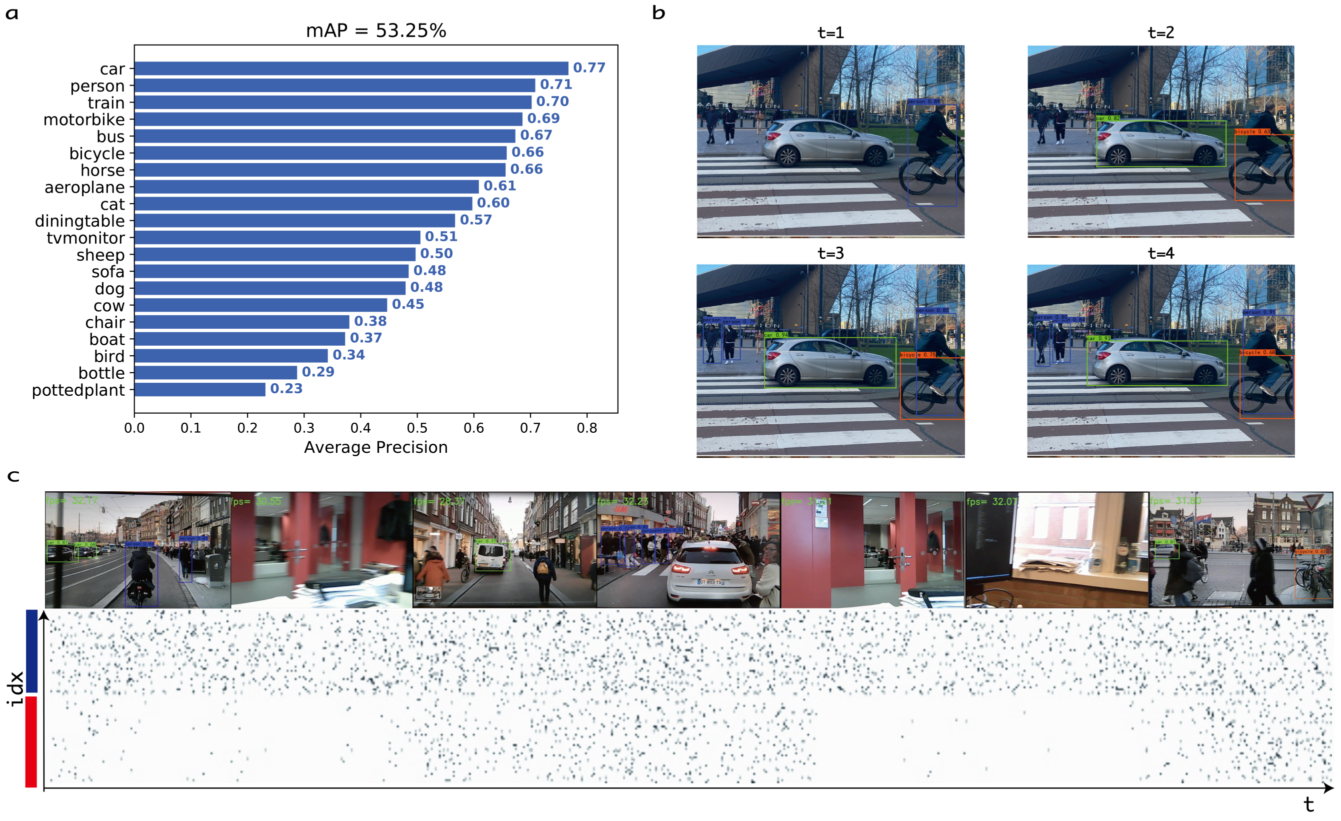


Figure 3. **a**, Mean Average Precision (mAP) and components for classification and recognition of 20 different kinds of objects in SPYv4 on the PASCAL VOC07 dataset. **b**, An example of object recognition with Spiking-YOLO on a sequence of images. Objects are localized and identified for each image independently. **c**, SPYv4 applied to streaming video. Top: example images, bottom: raster plot of spiking activity for 100 spiking neurons randomly drawn from the shallow and deep layer.

trained with FPTT consistently and substantially outperform SNNs trained with online BPTT approximations like OSTL and e-prop. Compared to offline BPTT approaches, the FPTT-trained LTC-SNNs achieve state-of-the-art for SNNs (R-MNIST) or achieve close to similar performance (PS-MNIST,S-MNIST, DVS-GESTURE), DVS-Cifar10); additionally, the memory requirements for FPTT vs. BTTP trained networks were lower by up to a factor of 5 (Table S3) while the training time was only slightly longer (Table S4).

Large-scale Object-detection: Spiking YOLO

The memory efficiency of FPTT-trained LTC-SNNs enables us to train SNNs of comparable complexity as modern ANNs: we demonstrate this by training a large spike-based object-detection model based on the Tiny YOLO-v4 architecture[22, 38]. The

'You-Only-Look-Once' (YOLO) architecture calculates both bounding boxes locations and object identities for all identifiable objects in an image using a single pass through a deep neural network.

Our SPiking tiny Yolo-v4 network - SPYv4 - has 19 spiking convolutional layers with about 6.2M spiking neurons, 2 convolutional output layers and 14M parameters in total, illustrated in Fig. S2**a**. This makes it both larger and deeper than previous end-to-end trained spiking models. Training a single time-step in the network requires less than 14GB of GPU-memory, and as BPTT scales linearly with the number of time-steps, learning with BPTT in such a large network over multiple time-steps is infeasible.

To carry out object detection, the network uses multiple reads of the input image, e.g. 4 or 8 times, as time-steps in the network to obtain the final result; trained with FPTT, SPYv4 achieves a mean Average Precision (mAP) of 51.38% at 4 reads and 53.25% at 8 reads (Fig. 3**a**) on the VOC dataset (see Methods); Fig. 3**b** shows examples of the detected and classified objects. Neural activation in the network is highly sparse, with about 10% of neurons active on average at each time-step. When receiving direct camera inputs images, inference achieves about 60 time-steps per second on an NVIDIA RTX3090 equipped workstation, corresponding to processing 7 or 15 images per second. Fig. 3**c** shows example activity from neurons from respectively a shallow (near input) and deep network layer: neurons in the deeper layer fall silent when irrelevant stimuli are shown, while neurons closer to the inputs remain active. Earlier work like Spiking-YOLO[18] achieved mAP 51.83% with 8000 simulation time-steps; our SPYv4 network thus outperforms these networks in performance, sparseness, and latency.

Discussion

We showed how a recently proposed training approach for recurrent neural networks, FPTT, can be successfully applied to long sequence learning with recurrent SNNs using Liquid Time-Constant Spiking Neurons. Compared to BPTT, FPTT is compatible with online training, has constant memory requirements, and can learn longer sequences. The increased memory efficiency of FPTT allows for training much larger SNNs as was previously feasible, as we demonstrated in the SPYv4 network for object detection. In terms of accuracy, FPTT outperforms online approximations of BPTT like OSTL and e-prop, and enabled a demonstration of online learning in tasks like DVS-CIFAR10. When training large convolutional LTC-SNNs with FPTT, excellent performance is achieved, approaching or exceeding offline BPTT-based solutions using corresponding network architectures – LTC-SNN specific architecture searches may improve results further.

To achieve efficient and accurate online learning with FPTT, we introduced Liquid Time-Constant Spiking Neurons (LTC-SNs), where the neuron's time-constants are calculated as a learned dynamic function of the current state and input. When training on various tasks, BPTT failed to converge when applied to LTC-SNNs on long sequences due to diverging gradients, while FPTT consistently converged. As we speculated, this suggests that FPTT provides for a more robust learning signal. While LTC-SNs provide additional memory in the networks, and LTC-SNNs without recurrent connections were able to solve shorter sequences, though not longer (Fig. S4), the optimal degree of network recurrency remains to be determined and may allow for further efficiency and accuracy improvements[39]. LTC-SNs maintain binary communication between neurons, but impose additional calculations to determine the neuron state. In neuromorphic implementations, LTC-SNs could be implemented as multi-compartment neurons or require novel spiking neuron model implementations.

The LTC-SN is inspired by multi-compartment modeling of pyramidal neurons in brains. Pyramidal neurons are known to have complex non-linear interactions between different morphological parts far exceeding the simple dynamics of LIF-style neurons[40,41], where the neuron's apical tuft may calculate a modulating term acting on the computation in the soma[42] that could act similar to the trainable Liquid time-constants used in this work. In a similar vein, learning rules derived from weight-specific traces may relate to synaptic tags[43,44] and are central to biologically plausible theories of learning working memory[45]. In general, we find that FPTT, unlike BPTT, can also train networks of complex biologically realistic spiking neuron models, like Izhikevich and Hodgkin-Huxley models (e.g. for the DVS-GESTURE task, Table S5). These considerations suggest variations of FPTT may be potential candidates for temporal credit assignment mechanisms in the brain. As a candidate for biologically plausible learning, the spatial error-backpropagation employed in FPTT would need to be replaced with a plausible spatial credit assignment solution, where we anticipate that at least some of the current proposals[46,47] may be compatible. With such local spatial credit-assignment, FPTT-training of LTC-SNNs can also likely be implemented efficiently on neuromorphic hardware.

Taken together, our work suggests that FPTT is an excellent training paradigm for large-scale SNNs comprised of complex spiking neurons, with implications for both decentralized AI based on local neuromorphic computing and investigations of biologically plausible neural processing.

Methods

Forward Propagation through Time. FPTT considers learning as a consensus problem between the network updates at different time-steps, where the network update at each single time-step needs to move toward the same converged optimal weights. To achieve this, FPTT updates the network parameters W by optimising the instantaneous risk function ℓ_t^{dyn}, which includes the ordinary objective $\mathscr{L}_t$ and also a dynamic regularisation penalty $\mathscr{R}_t$ based on previously observed losses $\ell_t^{dyn} = \mathscr{L}_t + \mathscr{R}_t$ (see Appendix A in the SI for details). In FPTT, the empirical objective $\mathscr{L}(y_t, \hat{y}_t)$ is the same as for BPTT, representing a function of the gap between target values y_t and real time predictions $\hat{y}_t$.

As in[8], the FPTT-specific dynamic regularization is controlled by a form of running average of all the weight-updates calculated so far $\bar{W}$, where the update schema of this regularizer $\mathscr{R}_t$ is as follows:

$$\mathscr{R}(W_t) = \frac{\alpha}{2} \parallel W_t - \bar{W}_t - \frac{1}{2\alpha}\nabla l_{t-1}(W_t) \parallel^2 \tag{1a}$$

$$W_{t+1} = W_t - \eta \nabla_W l(W_t) \tag{1b}$$

$$\bar{W}_{t+1} = \frac{1}{2}(\bar{W}_t + W_{t+1}) - \frac{1}{2\alpha}\nabla l_t(W_{t+1}). \tag{1c}$$

Here, the state vector $\bar{W}_t$ summarises past losses: the update is first a normal update of parameters W_t based on gradient optimization with fixed $\bar{W}_t$, after which $\bar{W}_t$ is optimized with fixed W_t. This approach allows the RNN parameters to converge to a stationary solution of the traditional RNN objective[8]. Note that in Eq. 1c, the loss $\nabla l_t(W_{t+1})$ is estimated as in[8], avoiding propagating the gradient through Eq. 1b, where the $\nabla_W l(W_t)$ calculates the direct spatial gradient.

The plain FPTT learning process requires the acquisition of an instantaneous loss l_t at each time step. This is natural for sequence-to-sequence modeling tasks and streaming tasks where a loss is available for each time step; for temporal sequence classification tasks however, the target value is only determined after processing the entire time series. To apply FPTT to learning such tasks in an online manner, a divergence term was introduced[8] in the form of an auxiliary loss to reduce the distance between the prediction distribution $\hat{P}$ and target label distribution Q:

$$l_t = \beta l_t^{CE}(\hat{y}_y, y) + (1-\beta) l_t^{div}, \tag{2}$$

where $\beta \in [0,1]$; l_t^{CE} is the classical cross-entropy for a classification loss and $l_t^{div} = -\sum_{\bar{y}} Q(\bar{y}) \log \hat{P}(\bar{y})$ is the divergence term. We use the auxiliary loss in all experiments as in[8] with $\beta = \frac{t}{T}$, where T is the sequence length. Note that variants of FPTT, FPTT-K[8], update K times during the sequence rather than every time-step, plain FPTT corresponds to FPTT-T. Absent the dynamic regularization through $\bar{W}$ FPTT-T amounts to spatial error-backpropagation (BP) without backpropagation-through-time – the absence of these regularization terms drastically reduces accuracy, both for plain BP as well as for truncated BPTT (Fig. S6).

Training networks of spiking neurons. To apply FPTT to SNNs, we define the spiking neuron model and specify how BPTT and FPTT are applied to such networks. All networks were trained using batches as in[8] to exploit GPU parallelism; reduction to batch-size=1 yielded similar results (e.g. Fig. S3).

An SNN is comprised of spiking neurons which operate with non-linear internal dynamics. These non-linear dynamics consist of three main components:

(1) Potential Updating: the neurons' membrane potential u_t updates following the equation:

$$u_t = f(u_{t-1}, x_t, s_{t-1} \| W, \tau) \tag{3}$$

where τ is the set of internal time constants and W is the set of associated parameters, like synaptic weights. The membrane potential evolves based on previous neuronal states (e.g. potential u_{t-1} and spike-state $s_{t-1} = \{0,1\}$) and current inputs x_t. Training the time constants τ in the spiking neurons is known to optimize performance by matching the neuron's temporal dynamics of the task[16,29].

(2) Spike generation: A neuron will trigger a spike $s_t = 1$ when its membrane potential u_t crosses a threshold θ from below, described as a discontinuous function:

$$s_t = f_s(u_t, \theta) = \begin{cases} 1, & \text{if } u_t \geq \theta \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

(3) Potential resting: When a neuron emits a spike ($s_t = 1$), its membrane potential will reset to resting potential u_r:

$$u_t = (1 - s_t)u_t + u_r s_t, \tag{5}$$

where in all experiments, we set $u_r = 0$.

BPTT for SNNs BPTT for SNNs amounts to the following: given a training example $\{x, y\}$ of T time steps, the SNN generates a prediction $\hat{y}_t$ at each time step. At time t, the SNN parameters are optimized by gradient descent through BPTT to minimize the instantaneous objective $\ell_t = \mathscr{L}(y_t, \hat{y}_t)$. The gradient expression is the sum of the products of the partial gradients, defined by the chain rule as

$$\frac{\partial \ell_t}{\partial W} = \frac{\partial l_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial s_t}\frac{\partial s_t}{\partial u_t}\sum_{j=1}^{t}\left(\prod_{m=j}^{t}\frac{\partial u_m}{\partial u_{m-1}}\right)\frac{\partial s_{m-1}}{\partial W}, \tag{6}$$

where the partial derivative of spiking $\frac{\partial s_t}{\partial u_t}$ is calculated by a surrogate gradient associate with membrane potential u_t. Here, we use the Multi-Gaussian surrogate gradient function $\hat{f'}_s(u_t, \theta)$[1] to approximate this partial term.

The computational graph of BPTT is illustrated in Fig1**a** and shows that the partial derivative term depends on two pathways, $\frac{\partial u_m}{\partial u_{m-1}} = \frac{\partial u_m}{\partial u_{m-1}} + \frac{\partial u_m}{\partial s_{m-1}}\frac{\partial s_{m-1}}{\partial u_{m-1}}$. The product of these partial terms may explode or vanish in RNNs, and this phenomenon becomes even more pronounced in SNNs as the discontinuous spiking process is approximated by the continuous surrogate gradient and the incurred gradient error accumulates and amplifies.

FPTT for SNNs. FPTT can be used for training SNNs by minimizing the instantaneous loss with the dynamic regularizer $\ell_t^{dyn} = \mathscr{L}(y_t, \hat{y}_t) + \mathscr{R}(\bar{W}_t)$. The update function Equation (6) then becomes:

$$\frac{\partial \ell_t^{dyn}}{\partial W} = \frac{\partial l_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial s_t}\frac{\partial s_t}{\partial u_t}\frac{\partial u_t}{\partial W}. \tag{7}$$

Compared to Equation (6), Equation (7) has no dependence on a chain of past states, and can thus be computed in an online manner.

Liquid Time-Constant Spiking Neurons The LTC-SN is modeled as a standard adaptive spiking neuron[1,11] where the time constants τ (here, membrane time constant τ_m and adaptation time constant τ_{adp}) are a dynamic and learned function of internal dynamic state variables like membrane potential u and deviation[11] b. In the network, time-constants are either calculated as a function $\alpha = \exp(-dt/\tau_m) = \sigma(Dense[x_t, u_{t-1}])$, for non-convolutional networks, or using a 2D convolution for spiking convolutional networks, $\alpha = \exp(-dt/\tau_m) = \sigma(Conv([x_t, u_{t-1}]))$, where we use a sigmoid function $\sigma(\cdot)$ to scale the inverse of the time constant to a range of 0 to 1, ensuring smooth changes when learning. This results in a Liquid Time-Constant Spiking Neuron i defined as:

$$\tau_{adp}\,update : \rho^i = \exp(-dt/\tau^i_{adp}) = \sigma(Dense_{adp}[x_t, b^i_{t-1}]) \tag{8a}$$
$$\tau_m\,update : \alpha^i = \exp(-dt/\tau^i_m) = \sigma(Dense_m[x_t, u^i_{t-1}]) \tag{8b}$$
$$\theta_t\,update : b^i_t = \rho^i b^i_{t-1} + (1-\rho^i)s^i_{t-1};\ \theta^i_t = 0.1 + 1.8 b^i_t \tag{8c}$$
$$u_t\,update : du^i = -u^i_{t-1} + x_t;\ u^i_t = \alpha^i u^i_{t-1} + (1-\alpha^i)du^i \tag{8d}$$
$$spike\,s_t : s^i_t = f_s(u^i_t, \theta^i) \tag{8e}$$
$$resetting : u^i_t = u^i_t(1-s^i_t) + u_{rest}s^i_t, \tag{8f}$$

where the neuron uses an adaptive threshold θ_t as in the Adaptive Spiking Neurons[11], resting potential $u_{rest} = 0$, and time-constants τ^i_m and τ^i_{adp} are computed as liquid time-constants of neuron i.

Datasets We focus on a number of datasets suitable for temporal classification, where the goal is to obtain high accuracy read-out on test-samples at the final time-step T of the sequence. For the Add-task, the trained networks consist of 128 recurrently connected neurons of respective types LTC-SN (LTC-SNN), LSTM, or Adaptive Spiking Neuron[11] (ASRNN), and a dense output layer with only 1 neuron.

For the DVS-GESTURE dataset, each frame is a 128-by-128 size image with 2 channels. Each sample in the DVS-GESTURE dataset was split into fixed-duration blocks as in[16], where each block is averaged to a single frame. This conversion results in sequences of up to 1000 frames depending on block length. As input for the shallow SRNN as used in the increasingly long sequence setting, we first down-sample the frame of a 128-by-128 image into a 32-by-32 image by averaging each 4-by-4 pixel block. The 2D image at each channel is then flattened into a 1D vector of length 1024. For each channel of the image, the network consists of a spike-dense input layer consisting of 512 neurons as an encoder, where the information of each channel is then fused into a 1D binary vector through concatenation. This 1D vector is then fed to a recurrently connected layer with 512 hidden neurons. Finally, a leaky integrator is applied to generate predictions, resulting in a network architecture [1024,1024]-[512D,512D]-512R-11I[1]. All networks were trained for 30 epochs using the Adam optimizer with initial learning rate 3e-3, using the same initialization schemes and learning-rate scheme for all networks to compare the effect of neural units. Hyperparameters for ASRNNs were taken from[1].

To achieve high performance with SCNNs, we applied FPTT with LTC-SNs to high-performance architectures from the literature, where we used hyperparameter settings for the surrogate gradient from[29] and from[8] for FPTT training. Specifically, in (P)S-MNIST, we used a shallow network with one recurrent layer comprised of 512 hidden neurons and an output layer consisting of 10 (number of classes) leaky integrator neurons. The FPTT-trained LTC-SNNs are optimized using Adam[48] with a batch size of 128 using 200 training epochs. We set the initial learning rate to 3e-3 and decay by half after 30, 80, and 120 epochs. For the S-MNIST and PS-MNIST tasks, we also find that the leak time-constants of the output units after training averaged 87ms $\pm$13ms (S-MNIST) and 65ms$\pm$12ms (PS-MNIST), substantially shorter than the sequence length and demonstrating that the recurrent network maintains working memory. For R-MNIST, we follow the architecture from[12]: an SNN with two hidden layers of 256 neurons, each followed by 10 output neurons. The SNN is given 20 presentations of the image, after which the classification is determined. For the static MNIST and Fashion MNIST datasets, we apply the architecture from[16]: an SCNN with 3 convolutional layers, 1 Dense layer and 1 leaky Integrator output layer (ConvK3C32S1P1-MPK2S2-ConvK3C128S1P1-MPK2S2-ConvK3C256S1P1-MPK2S2-512D-10I. To describe the network structure, we follow standard conventions as follows: ConvK7C64S1P1 represents the convolutional layer with $output\, channels = 64$, $kernel\, size = 7$, $stride = 1$ and $padding = 3$. MPK2S2 is the max-pooling layer with $kernel\, size = 2$ and $stride = 2$. 512D and 512R represents the fully connected spiking layer and recurrent spiking layer respectively with $output\, features = 512$. 10I indicates the leaky integrator with the $output\, feature = 10$.). The resulting LTC-SCNN network was then optimized using FPTT and Adamax with a batch size of 64 and an initial learning rate of 1e-3. For the DVS-GESTURE and DVS-CIFAR10 datasets, we also follow the high-performance architecture from[16] and use 20 sequential frames, where the network makes a prediction only after reading the entire sequence. The LTC-SCNN thus has a structure ConvK7C64S1P3-MPK2S2-ConvK7C128S1P3-MPK2S2-ConvK3C128S1P1-MPK2S2-ConvK3C256S1P1-MPK2S2-ConvK3C256S1P1-MPK2S2-ConvK3C512S1P1-MPK2S2-512D-11I. These networks were optimized through Adamax[48] with a batch size of 16 and initial learning rate of 1e-3.

For all networks, to measure GPU memory consumption, the GPU management interface "nvidia-smi" was used.

Spiking Tiny YOLOv4 – SPYv4. The SPYv4 network follows the Tiny YOLO-v4 architecture[22,38]. The backbone of the network consists of three CSP-Blocks in the cross-stage partial network. In contrast to regular additive residual connections, the CSP-Block is spike-based rather than event-based as residual connections are constructed by concatenation rather than using an adding operation, ensuring that only binary spikes are used for information transfer between layers. Raw pixel values are fed directly into the network as input currents. As the object recognition task necessitates a more precise output vector to draw the bounding box, we use a single standard conventional convolutional layer instead of a spiking convolutional layer.

We trained and evaluated our model on Pascal VOC dataset[17] as was done in[18]: the network was trained using a combination of VOC2007 and VOC2012 (16551 images), and evaluated on the validation dataset of VOC2007 (4952 images).

For target detection, we use a threshold on the Intersection Over Union (IOU), i.e., the ratio of the intersection of the prediction box and the ground truth box of the target, to indicate whether the detection is correct. A target is considered to be correctly detected when the IOU exceeds the threshold. The average precision (AP) is then computed as the average of all the precision for all possible recall values, and the mAP is the average of the AP of multiple classification tasks. Training maximizes the mAP@ϑ: the mean average precision of the calculated bounding box exceeding an overlap threshold ϑ over the actual boundaries of the object in the dataset, where we use $\vartheta = 0.5$.

We applied both mosaic data augmentation[38] and label smoothing during training to obtain better performance. In the mosaic enhancement, the network uses a mosaic of 4 images during training instead of a single image; this is applied only during the first 200 training cycles. The network was optimized by Adagrad with a batch size of 32 and an initial learning rate of 1e-3.

Code Availability

The code used in the study is publicly available from the GitHub repository `https://github.com/byin-cwi/sFPTT/tree/v1.0.0`[49].

References

1. Yin, B., Corradi, F. & Bohté, S. M. Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks. *Nat Mach Intell* **3**, 905–913 (2021).
2. Stuijt, J., Sifalakis, M., Yousefzadeh, A. & Corradi, F. μbrain: An event-driven and fully synthesizable architecture for spiking neural networks. *Front. neuroscience* **15**, 538 (2021).

3. Perez-Nieves, N., Leung, V. C. H., Dragotti, P. L. & Goodman, D. F. M. Neural heterogeneity promotes robust learning. *Nat. Commun.* **12**, 5791 (2021).

4. Keijser, J. & Sprekeler, H. Interneuron diversity is required for compartment-specific feedback inhibition. *bioRxiv* (2020).

5. Bohte, S. M. Error-backpropagation in networks of fractionally predictive spiking neurons. In *International Conference on Artificial Neural Networks*, 60–68 (Springer, 2011).

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

7. Paszke, A. *et al.* Pytorch: An imperative style, high-performance deep learning library. *Adv. neural information processing systems: NeurIPS* **32**, 8026–8037 (2019).

8. Kag, A. & Saligrama, V. Training recurrent neural networks via forward propagation through time. In *International Conference on Machine Learning*, 5189–5200 (PMLR, 2021).

9. Mehonic, A. & Kenyon, A. J. Brain-inspired computing needs a master plan. *Nature* **604**, 255–260 (2022).

10. Williams, R. J. & Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural computation* **1**, 270–280 (1989).

11. Bellec, G. *et al.* A solution to the learning dilemma for recurrent networks of spiking neurons. *Nat. communications* **11**, 1–15 (2020).

12. Bohnstingl, T., Woźniak, S., Pantazi, A. & Eleftheriou, E. Online spatio-temporal learning in deep neural networks. *IEEE Transactions on Neural Networks Learn. Syst.* (2022).

13. He, Y. *et al.* A 28.2 μw neuromorphic sensing system featuring snn-based near-sensor computation and event-driven body-channel communication for insertable cardiac monitoring. In *2021 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, 1–3 (2021).

14. Hasani, R., Lechner, M., Amini, A., Rus, D. & Grosu, R. Liquid time-constant networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 7657–7666 (2021).

15. Amir, A. *et al.* A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 7243–7252 (2017).

16. Fang, W. *et al.* Incorporating learnable membrane time constant to enhance learning of spiking neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2661–2671 (2021).

17. Everingham, M., Van Gool, L., Williams, C. K., Winn, J. & Zisserman, A. The pascal visual object classes (voc) challenge. *Int. journal computer vision* **88**, 303–338 (2010).

18. Kim, S., Park, S., Na, B. & Yoon, S. Spiking-yolo: spiking neural network for energy-efficient object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 11270–11277 (2020).

19. Chakraborty, B., She, X. & Mukhopadhyay, S. A fully spiking hybrid neural network for energy-efficient object detection. *IEEE Transactions on Image Process.* **30**, 9014–9029 (2021).

20. Royo-Miquel, J., Tolu, S., Schöller, F. E. & Galeazzi, R. Retinanet object detector based on analog-to-spiking neural network conversion. In *8th International Conference on Soft Computing & Machine Intelligence* (2021).

21. Zhou, S., Chen, Y., Li, X. & Sanyal, A. Deep scnn-based real-time object detection for self-driving vehicles using lidar temporal data. *IEEE Access* **8**, 76903–76912 (2020).

22. Jiang, Z., Zhao, L., Li, S. & Jia, Y. Real-time object detection method based on improved yolov4-tiny. *arXiv preprint arXiv:2011.04244* (2020).

23. Werbos, P. J. Backpropagation through time: what it does and how to do it. *Proc. IEEE* **78**, 1550–1560 (1990).

24. Elman, J. L. Finding structure in time. *Cogn. science* **14**, 179–211 (1990).

25. Mozer, M. C. Neural net architectures for temporal sequence processing. In *Santa Fe Institute Studies in the Sciences of Complexity-Proceedings Volume-*, vol. 15, 243–243 (1993).

26. Murray, J. M. Local online learning in recurrent networks with random feedback. *ELife* **8**, e43299 (2019).

27. Knight, J. C. & Nowotny, T. Efficient GPU training of LSNNs using eprop. In *Neuro-Inspired Computational Elements Conference*, NICE 2022, 8–10 (Association for Computing Machinery, New York, NY, USA, 2022).

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

29. Yin, B., Corradi, F. & Bohté, S. M. Effective and efficient computation with multiple-timescale spiking recurrent neural networks. In *International Conference on Neuromorphic Systems 2020*, 1–8 (2020).

30. Scherr, F. & Maass, W. Analysis of the computational strategy of a detailed laminar cortical microcircuit model for solving the image-change-detection task. *bioRxiv* (2021).

31. Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural computation* **9**, 1735–1780 (1997).

32. Li, H., Liu, H., Ji, X., Li, G. & Shi, L. Cifar10-dvs: an event-stream dataset for object classification. *Front. neuroscience* **11**, 309 (2017).

33. Gerstner, W., Kreiter, A. K., Markram, H. & Herz, A. V. Neural codes: firing rates and beyond. *Proc. Natl. Acad. Sci.* **94**, 12740–12741 (1997).

34. Woźniak, S., Pantazi, A., Bohnstingl, T. & Eleftheriou, E. Deep learning incorporating biologically inspired neural dynamics and in-memory computing. *Nat. Mach. Intell.* **2**, 325–336 (2020).

35. Zou, Z. *et al.* Memory-inspired spiking hyperdimensional network for robust online learning. *Sci. reports* **12**, 1–13 (2022).

36. Shrestha, A., Fang, H., Wu, Q. & Qiu, Q. Approximating back-propagation for a biologically plausible local learning rule in spiking neural networks. In *Proceedings of the International Conference on Neuromorphic Systems*, 1–8 (2019).

37. Kaiser, J., Mostafa, H. & Neftci, E. Synaptic plasticity dynamics for deep continuous local learning (decolle). *Front. Neurosci.* **14**, 424 (2020).

38. Bochkovskiy, A., Wang, C.-Y. & Liao, H.-Y. M. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934* (2020).

39. Kalchbrenner, N. *et al.* Efficient neural audio synthesis. In *International Conference on Machine Learning*, 2410–2419 (PMLR, 2018).

40. Sacramento, J., Ponte Costa, R., Bengio, Y. & Senn, W. Dendritic cortical microcircuits approximate the backpropagation algorithm. *Adv. Neural Inf. Process. Syst. NeurIPS* **31**, 8721–8732 (2018).

41. Beniaguev, D., Segev, I. & London, M. Single cortical neurons as deep artificial neural networks. *Neuron* **109**, 2727–2739 (2021).

42. Larkum, M. E., Senn, W. & Lüscher, H.-R. Top-down dendritic input increases the gain of layer 5 pyramidal neurons. *Cereb. Cortex* **14**, 1059–1070 (2004).

43. Frey, U. & Morris, R. G. Synaptic tagging and long-term potentiation. *Nature* **385**, 533–536 (1997).

44. Moncada, D., Ballarini, F., Martinez, M. C., Frey, J. U. & Viola, H. Identification of transmitter systems and learning tag molecules involved in behavioral tagging during memory formation. *Proc. Natl. Acad. Sci.* **108**, 12931–12936 (2011).

45. Rombouts, J. O., Bohte, S. M. & Roelfsema, P. R. How attention can create synaptic tags for the learning of working memories in sequential tasks. *PLoS computational biology* **11**, e1004060 (2015).

46. Pozzi, I., Bohte, S. & Roelfsema, P. Attention-gated brain propagation: How the brain can implement reward-based error backpropagation. *Adv. Neural Inf. Process. Syst. NeurIPS* **33** (2020).

47. Scellier, B. & Bengio, Y. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Front. computational neuroscience* **11**, 24 (2017).

48. Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. *ICLR* (2015).

49. Yin, B. Code of paper, DOI: 10.5281/ZENODO.7498559 (2023).

Supplementary Information

Algorithm 1: Training SNN with BPTT

```
// Dataset $B = \{x_t, y_t\}_{t=0}^{T}$, # Epochs $E$
// Set Optimizer and learning rate $\eta$
// Initialize Weight $W = \{W^h, W^y\}$
for $i = 1$ to $E$ // Training Loop
do
    Initialize neuron states $u_t, s_t$; Randomly Shuffle $B$
    for $t = 0$ to $T$ // For each sample
    do
        $s_{h,t}, u_{h,t} = \hat{f}_s(x_t, [u_{h,t-1}, s_{h,t-1}] \| W^h)$
        $\hat{y}_t = \hat{f}_s(s_{h,t}, [u_{o,t}, s_{o,t}] \| W^y)$
    Loss:
        $\ell(W) = \sum_{t=1}^{T} \ell(y_t, \hat{y}_t)$
    Update:
        $W = W - \eta \nabla_W \ell(W)|_W$
```

Algorithm 2: Training SNN with FPTT

```
// Dataset $B = \{x_t, y_t\}_{t=0}^{T}$, # Epochs $E$
// Set Optimizer and learning rate $\eta$
// Initialize weight $W = \{W^h, W^y\}$, and
    $\bar{W} = W$
for $i = 1$ to $E$ // Training Loop
do
    Initialize neuron states $u_t$, $s_t$; Randomly Shuffle $B$
    for $t = 0$ to $T$ // For each sample
    do
        $s_{h,t}, u_{h,t} = \hat{f}_s(x_t, [u_{h,t-1}, s_{h,t-1}] \| W_t^h)$
        $\hat{y}_t = \hat{f}_s(s_{h,t}, [u_{o,t}, s_{o,t}] \| W_t^y)$
        Loss: $\ell_t(W_t)$: $\ell_t(W_t) = \ell(y_t, \hat{y}_t)$
        Dynamic Loss: $\ell^{dyn}(W_t) =$
          $\ell_t(W_t) + \frac{\alpha}{2} \|W_t - \bar{W}_t - \frac{1}{2\alpha} \nabla \ell_{t-1}(W_t)\|^2$
        Update $W$:
            $W_{t+1} = W_t - \eta \nabla_W \ell^{dyn}(W_t)$
        Update $\bar{W}$:
            $\bar{W}_{t+1} = \frac{1}{2}(\bar{W}_t + W_{t+1}) - \frac{1}{2\alpha} \nabla \ell_t(W_{t+1})$
```

Algorithms SA1. BPTT (left) and FPTT algorithm. Adapted from[8] where $\nabla l_t(W_{t+1}))$ is estimated as in[8], avoiding propagating the gradient through Equation 1b. The algorithms are described for batch-size 1, while in our experiments we relax this to larger batch-sizes as in[8].

Frames	BPTT						FPTT		
	LSTM+Aux	LSTM	LTC-SRNN+Aux	LTC-SRNN	ASRNN+Aux	*ASRNN*$^+$+Aux	*LTC*$^-$	LTC-SRNN	ASRNN
20	86.69±0.43	82.29±2.46	83.42±1.35	84.37±2.27	86.82±0.31	80.78±1.99	87.83±1.21	**89.31**±0.59	87.51±0.85
40	88.77±1.71	84.95±0.71	85.96±1.16	84.37±1.24	87.29±0.87	82.99±0.70	88.53±0.57	**90.39**±0.71	87.61±0.43
60	87.61±0.86	85.15±0.75	85.62±1.18	83.91±0.71	87.02±1.19	81.77±0.34	88.08±1.40	**90.74**±0.16	87.62±1.15
80	87.97±0.14	84.83±1.42	85.30±0.71	80.44±3.6	86.34±0.87	76.84±0.85	88.66±1.14	**91.31**±0.98	87.60±1.06
100	88.89±0.49	83.79±0.71	83.21±0.43	78.70±0.91	86.22±0.71	74.61±1.25	89.93±1.13	**91.89**±0.16	87.40±0.28
200	85.76±0.49	81.87±2.58	51.39±6.0	(43.98±2.35)*	79.62±1.89	65.89±1.69	89.24±1.56	**92.13**±0.87	88.31±1.33
500	82.52±1.82	78.81±1.5	38.89±3.22	(36.46±1.5)*	49.03±1.52	52.43±1.32	86.69±0.43	**90.64**±1.56	85.76±1.30
1000	+	+	+	+	+	+	90.05±1.56	**91.28**±1.05	84.24±1.23
Param	4.2M	4.2M	4.7M	4.7M	1.6M	4.7M	3.2M	4.7M	1.6M

Table S1. Performance comparison between BPTT and FPTT on the DVS gesture dataset. Each number in the table is the average of three runs. All networks have an equal number of neural units unless indicated otherwise. (*) denotes that training diverged; reported accuracy is the best accuracy before divergence. (+) denotes out-of-GPU-memory when training. The *ASRNN*$^+$ is an ASRNN with the same number of parameters as the LTC-SRNN. *LTC*$^-$ denotes an LTC-SRNN network where only the membrane time constant is dynamic, and the adaptation time constant is a learned fixed parameter. We remark that with a parameter-matched single recurrent layer ASRNN with 1400 neurons trained w/ BPTT, we achieved accuracy of $81.134\% \pm 0.71$ on 100 frames DVS-gestures dataset, while the same network trained with eProp achieved $75.45\% \pm 1.15$. Results for Truncated BPTT including plain spatial error-backpropagation are illustrated in Fig S6.

Loss	τ_m	$\tau_m(x)$	$\tau_m(x,u)$
τ_{adp}	0.17	0.0027	0.0025
$\tau_{adp}(x)$	0.17	0.0024	0.003
$\tau_{adp}(x,b)$	0.16	0.0021	0.0019

Table S2. Ablation study of LTC-SN performance on Add Task where either τ_m, τ_{adp}, or both, are trained (τ_m, τ_{adp}), dynamic from external input x ($\tau_m(x)$, $\tau_{adp}(x)$) or both external and internal input ($\tau_m(x,u)$, $\tau_{adp}(x,b)$).

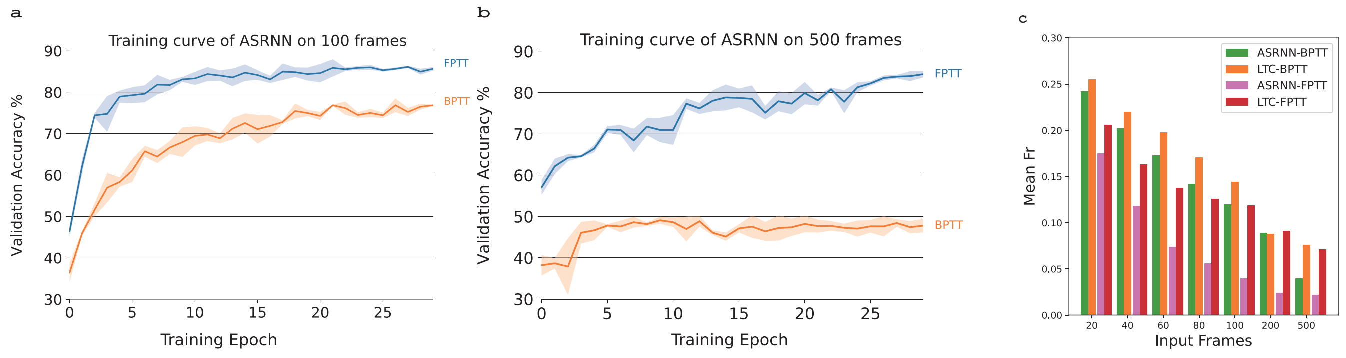


Figure S1. Learning curve of ASRNN trained via FPTT and BPTT on **a** 100 frames, and **b** 500 frames. Plotted is average and std over 3 run; **c**, Bar chart of mean firing rate (fr) comparison between BPTT and FPTT on the DVS gesture dataset. Each number in the table is the average of three runs.

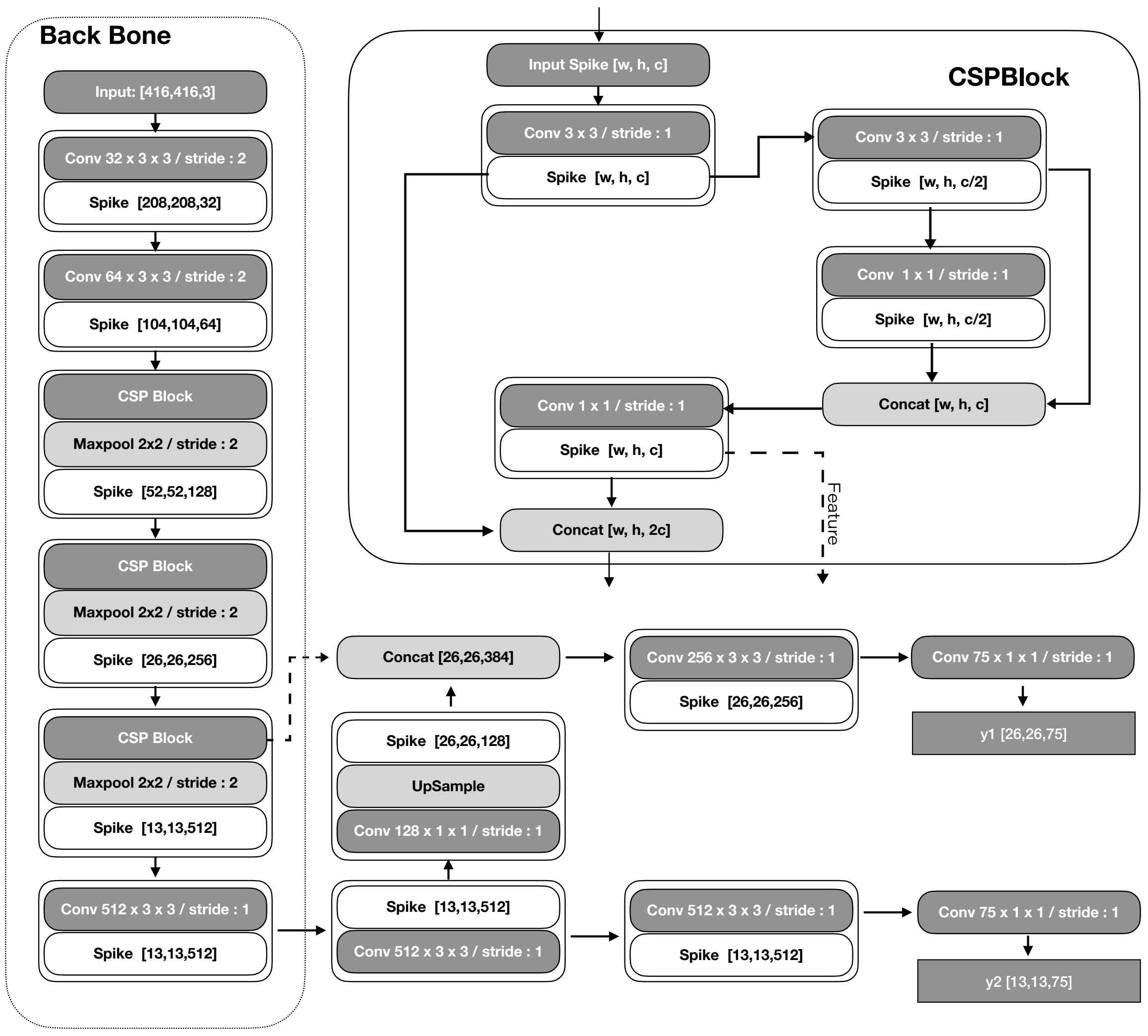


Figure S2. Spiking YOLOv4 (SPYv4) neural network architecture. The dash line read the "Feature" tensor which calculated before the last concatenation in CSP BLOCK

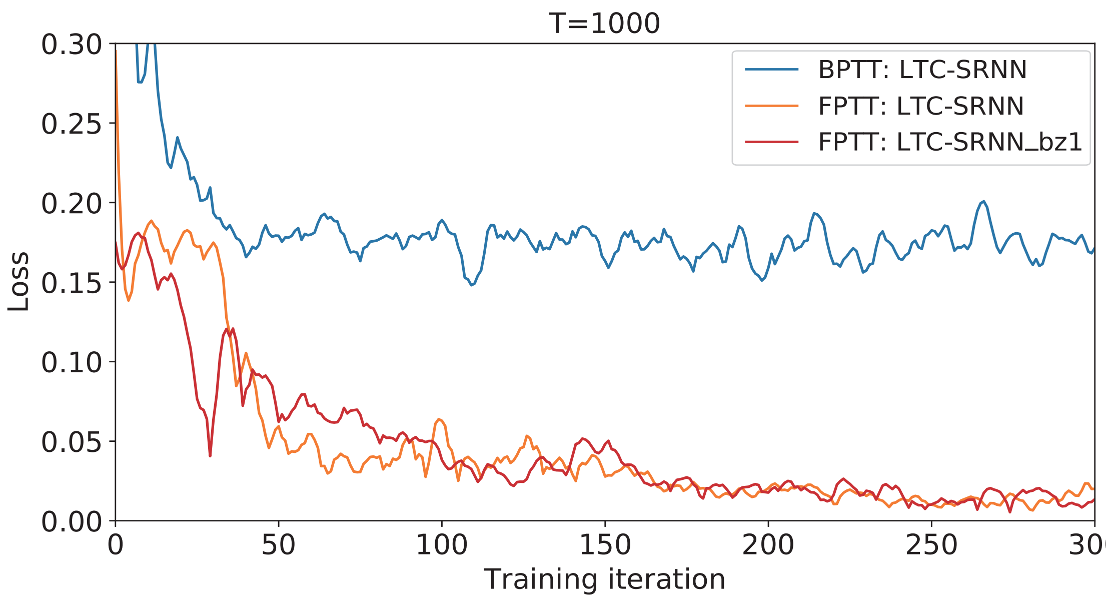


Figure S3. Example learning curves on the Add-task of LTC-SRNN trained with BPTT, FPTT, FPTT and batch-size 1 (“LTC-SRNN_bz1”, red curve).

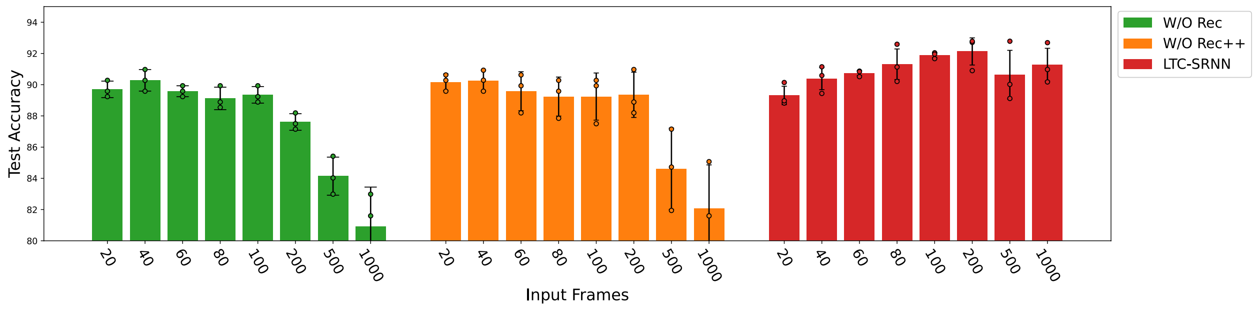


Figure S4. Ablation study of recurrent connections on the DVS Gesture dataset. Plot for test accuracy of LTC-SRNN on DVS Gesture dataset. The label “W/O Rec” denotes the LTC-SNN corresponding to the baseline recurrently connected LTC-SRNN but with the recurrent connections in the recurrent layer removed; “W/O Rec++” denotes an LTC-SNN with the same number of parameters as the LTC-SRNN (increasing the number of neural units and without recurrent connections). Test accuracy bars show mean ± std over 3 runs and individual data points are inserted.

Table S3. Memory efficiency. (*): the reported number is obtained using a halved batch size compared to the other entries to fit into GPU memory,

	S-MNIST	R-MNIST	MNIST	DVS-Gesture
BPTT	11.1GB	1.5GB	9.67GB	15.72GB(*)
FPTT	1.9GB	1.4GB	2.23GB	3.75GB

Table S4. Total training time per epoch

	S-MNIST	R-MNIST	MNIST	DVS gesture
BPTT	2518s	192s	362s	108s
FPTT	1233s	204s	384s	112s

Frames	Izhikevich	Hodgkin Huxley
100	84.03%	87.50%

Table S5. Performance of SNNs with Izhikevich and Hodgkin Huxley models on DVS-Gesture dataset. The network structure is same as the network in Table S1, where for the Izhikevich the a and b parameters are trainable and we kept c and d fixed, for the Hodgkin-Huxley model all neuron parameters were kept fixed; the network structure and the training-related hyperparameters were not further fine-tuned.

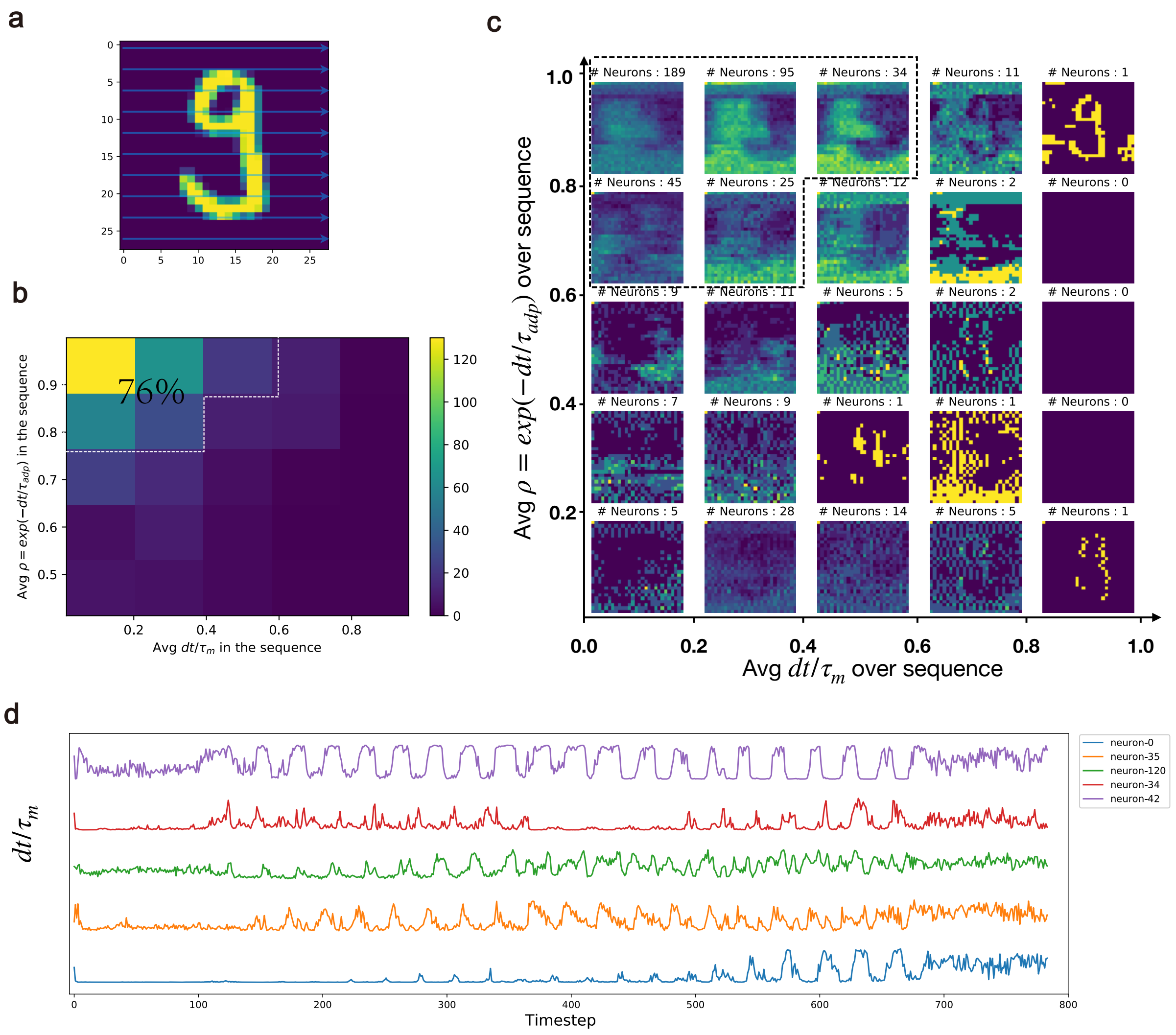


Figure S5. Dynamic neural responses of LTC-SNs. (a) an MNIST sample is sequentially fed into the LTC-SNN, pixel-by-pixel along the row-direction. (b) for illustration, we calculated the histograms of the resulting average dt/τ values after training for this single sample (c) average responses of neurons in terms of firing rate binned by $dt/\tau_m, exp(-dt/\tau_{adp})$ values (d) Example of dynamics of inverse time constant dt/τ_m for four randomly selected neurons during presentation of the sequence.

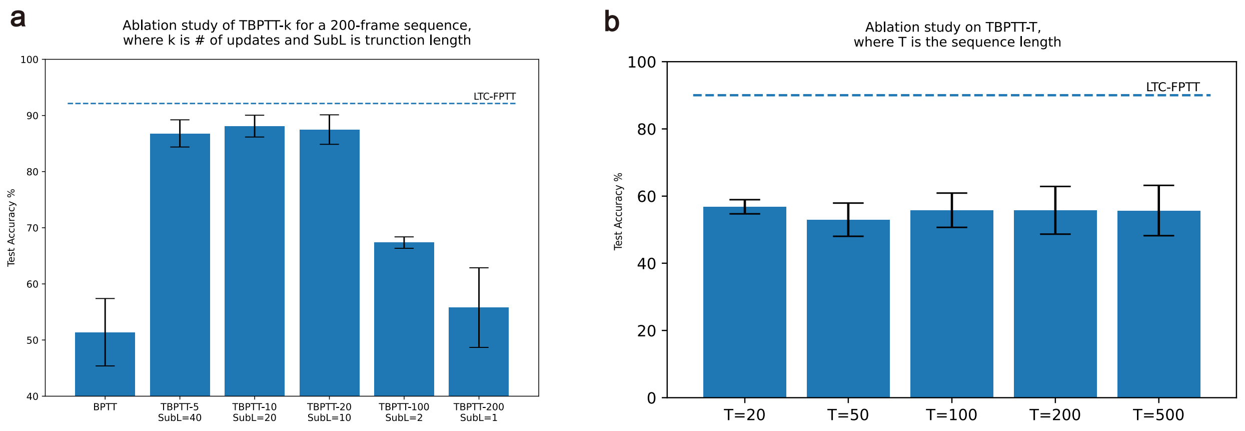


Figure S6. Ablation study of Truncated-BPTT-k ($TBPTT-k$) on DVS Gesture dataset of 200 frames where k is number of updates in the sequence resulting in truncation period $subL$. TBPTT-1, with truncation length 1, is functionally equivalent to BP. **(a)** Appropriate segmentation in time by truncation assists the network to achieve better results in long sequence classification problems. Compared to BPTT, T-BPTT has fixed consumption proportional to the truncation length. **(b)** For TBPTT-1, performance on the same network architecture is essentially constant as a function of sequence length.

Appendix A: FPTT theory

For conciseness, we briefly summarize the theory underlying FPTT as developed by Kag et al.[8].

Back-propagation-through-time Back-propagation-through-time (BPTT) uses backpropagation to calculate the gradient of the accumulated loss along the spatial-temporal dimension with respect to the parameters of the recurrent networks. Let us define a recurrent network described by differential equation $(\hat{y}^t, h_t) = NN(x_t, h_{t-1})$ where x_t is the input , $\hat{y}^t$ is the prediction and h_t is the hidden states. The gradient of time t is then computed by considering the effect of the state x_t on all future losses $l^t, l^{t+1}, ...l^T$:

$$\frac{\partial L}{\partial w} = \sum_{t=1}^{T} \frac{\partial l^t}{\partial w} = \sum_{t=1}^{T}\sum_{i=1}^{t} \frac{\partial l^t}{\partial h_i}\frac{\partial h_i}{\partial w} = \sum_{t=1}^{T}(\sum_{i=t}^{T}\frac{\partial l^i}{\partial h_t})\frac{\partial h_t}{\partial w} = \sum_{t=1}^{T}(\sum_{i=t}^{T}\frac{\partial l^i}{\partial h_t})\frac{\partial h_t}{\partial w} = \sum_{t=1}^{T}\{\sum_{i=t}^{T}(\prod_{j=i}^{T-1}\frac{\partial l^{j+1}}{\partial l^j})\frac{\partial l^i}{\partial h_t}\}\frac{\partial h_t}{\partial w}, \tag{A.1}$$

and a weight is then updated as: $w_{new} \leftarrow w_{old} - \frac{\partial L}{\partial w}$. At the end of training, the loss L will be minimized via optimal solution w^*, where $\frac{\partial}{\partial w}L(w^*) \cong 0$.

For online update based on BPTT, we will have $w_{t+1} \leftarrow w_t - \sum_{i=1}^{t}\frac{\partial}{\partial w}l^i(\hat{y}^i, y^i, w_t)$ where $l^i(\hat{y}^i, y^i, w_t)$ is the cost of time step i with parameter w_t, $\hat{y}^i$ and y^i are the prediction and target label of the time step i. When the algorithm converges to an optimal solution w^* at time step φ, we will have an optimal solution where:

$$w^* - w_t = -\nabla_w(l^t) = \frac{\partial}{\partial w}l^{\varphi}(\hat{y}^{\varphi}, y^{\varphi}, w^*) - \frac{\partial}{\partial w}l^t(\hat{y}^t, y^t, w_t) \tag{A.2}$$

and, for one step optimization:

$$w_{t+1} - w_t = \nabla_w l^{t+1} - \nabla_w l^t. \tag{A.3}$$

This demonstrates that for any timestep, the change of weight update is proportional to the change of the gradient; this observation (Equation (A.3)) is the foundation of Forward Propagation Through Time.

Forward Propagation Through Time FPTT aims to derive an online weight update mechanism with guaranteed convergence to optimal solution w^*. To have a smooth solution, FPTT learns from the historical information of weight changes by introducing a running mean $\bar{w}_t$ to summarize the historical information of weight evolution:

$$w_{t+1} - w_t = \nabla_w(l^{t+1}) - \nabla_w(l^t) \tag{A.4}$$

$$\Rightarrow \bar{w}_t - w_t \sim \nabla_w(l^{t+1}) - \nabla_w(l^t) \tag{A.5}$$

$$\Rightarrow \nabla_w(l^{t+1}) - \nabla_w(l^t) = \alpha[(\bar{w}_t - w_t) - (w_{t+1} - w_t)] = \alpha(\bar{w}_t - w_{t+1}) \tag{A.6}$$

From this, the convergence-guaranteed loss function for online update is derived, based on Eq. (A.6).

$$\nabla_w(l^{t+1}) - \nabla_w(l^t) = \alpha(\bar{w}_t - w_{t+1}) \qquad \Leftrightarrow \qquad \nabla_w(l^{t+1}) - \nabla_w(l^t) - \alpha(\bar{w}_t - w_{t+1}) = 0 \tag{A.7}$$

we define the constraint into the function $f(w_{t+1}) = \nabla_w(l^{t+1}) - \nabla_w(l^t) - \alpha(\bar{w}_t - w_{t+1})$. We now consider a convex function $F(w)$ which approached its minimum when $f(w_{t+1}) = 0$; we then have

$$F(w) = \int_w f(w)dw - \text{searching } w_{t+1} \text{ over parameter space} \tag{A.8}$$

$$= l^t(w) + \frac{\alpha}{2}\|w - \bar{w}_t - \frac{1}{2\alpha}\nabla_w(l(w_t))\|^2 \tag{A.9}$$

In this form, Eq A.7 is the first order condition for $F(w)$. So, the weight optimization is to minimize the new objective function

$$w_{t+1} = \arg\min_w l^t(w) + \frac{\alpha}{2}\|w - \bar{w}_t - \frac{1}{2\alpha}\nabla_w(l(w_t))\|^2 \tag{A.10}$$