nature machine intelligence

Article

https://doi.org/10.1038/s42256-023-00650-4

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

Received: 5 May 2022

Accepted: 30 March 2023

Published online: 8 May 2023

Check for updates

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

With recent advances in learning algorithms, recurrent networks of spiking neurons are achieving performance that is competitive with vanilla recurrent neural networks. However, 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 the 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. The efficiency and robustness of FPTT enable 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.

The binary, event-driven and sparse nature of communication between spiking neurons in the brain holds great promise for flexible and energy-efficient artifical intelligence (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] exhibit similarly excessive computational complexity, and low complexity approximations to BPTT like e-prop[11] or Online Spatio-Temporal Learning (OSTL)[12] at best approach the performance of BPTT. 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].

Recently, an algorithm was introduced for online learning in recurrent networks, Forward-Propagation Through Time (FPTT), demonstrating better generalization on many temporal classification benchmark tasks compared with BPTT[8]. 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,

[1]CWI, Machine Learning Group, Amsterdam, The Netherlands. [2]Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands. [3]Stichting IMEC Netherlands, Eindhoven, The Netherlands. [4]Faculty of Science, University of Amsterdam, Amsterdam, The Netherlands. [5]Faculty of Science and Engineering, Rijksuniversiteit Groningen, Groningen, The Netherlands. ✉e-mail: byin@cwi.nl

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 ref. 8, and we also observed this 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-constants (LTCs)[14], we introduce a class of spiking neurons, liquid time-constant spiking neurons (LTC-SNs), 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 (P/S-MNIST, R-MNIST, DVS-GESTURE) and to near (Fashion-MNIST, DVS-CIFAR10) or exceeding (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 were previously infeasible. We demonstrate this in a large-scale you-only-look-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 conversion of artificial neural networks (ANNs) to SNNs[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 (ref. 22) implementation—SPYv4—uses 21 layers, 6.2 million LTC spiking neurons and 14 million 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 comprising 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 computed along the paths of the unrolled network. The direct application of error-backpropagation to this unrolled graph is known as BPTT[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 RTRL[10] and approximations thereof[26]. RTRL, however, is prohibitive in time and memory complexity, and while approximations improve complexity (see ref. 12 and Table 1), they yield variable and task-dependent accuracy deficits compared with exact gradients[11,27].

SNNs are neural networks composed of spiking neurons: stateful neural units that communicate using binary values, that is 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 neurons' 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 (for example, 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 (Supplementary Table 5). Furthermore, because the SNN training accuracy heavily depends on hyperparameters, obtaining convergence using BPTT in SNNs is non-trivial.

For SNNs, approximations to BPTT like e-prop[11] and 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].

Table 1 | Computational complexity of gradients, parameter updates and memory storage per sample

Algorithm	Gradient update	Parameter update	Memory storage
BPTT	$\Omega(c(T)T)$	$\Omega(1)$	$\Omega(T)$
RTRL	$\Omega(c(T)T^2)$	$\Omega(T)$	$\Omega(T)$
e-prop/OSTL	$\Omega(c(1)T)$	$\Omega(T)$	$\Omega(1)$
FPTT	$\Omega(c(1)T)$	$\Omega(T)$	$\Omega(1)$

Computational expense increases as the length T of the sequence grows, that is $c(1) < c(T)$.

FPTT in SNNs

As introduced in ref. 8, FPTT can be implemented directly on the computational graph of SNNs, similar to BPTT approaches. This is illustrated in Fig. 1a; the gradient calculation and corresponding algorithms are given in Methods and Supplementary Algorithm. FPTT intuitively provides a more robust and efficient gradient approximation for SNNs 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 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. Also, inspired by ref. 14, the LTC-SN's internal time-constants are a learned function of the inputs and hidden states of the network (Fig. 1b, Methods). That is, 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

https://doi.org/10.1038/s42256-023-00650-4

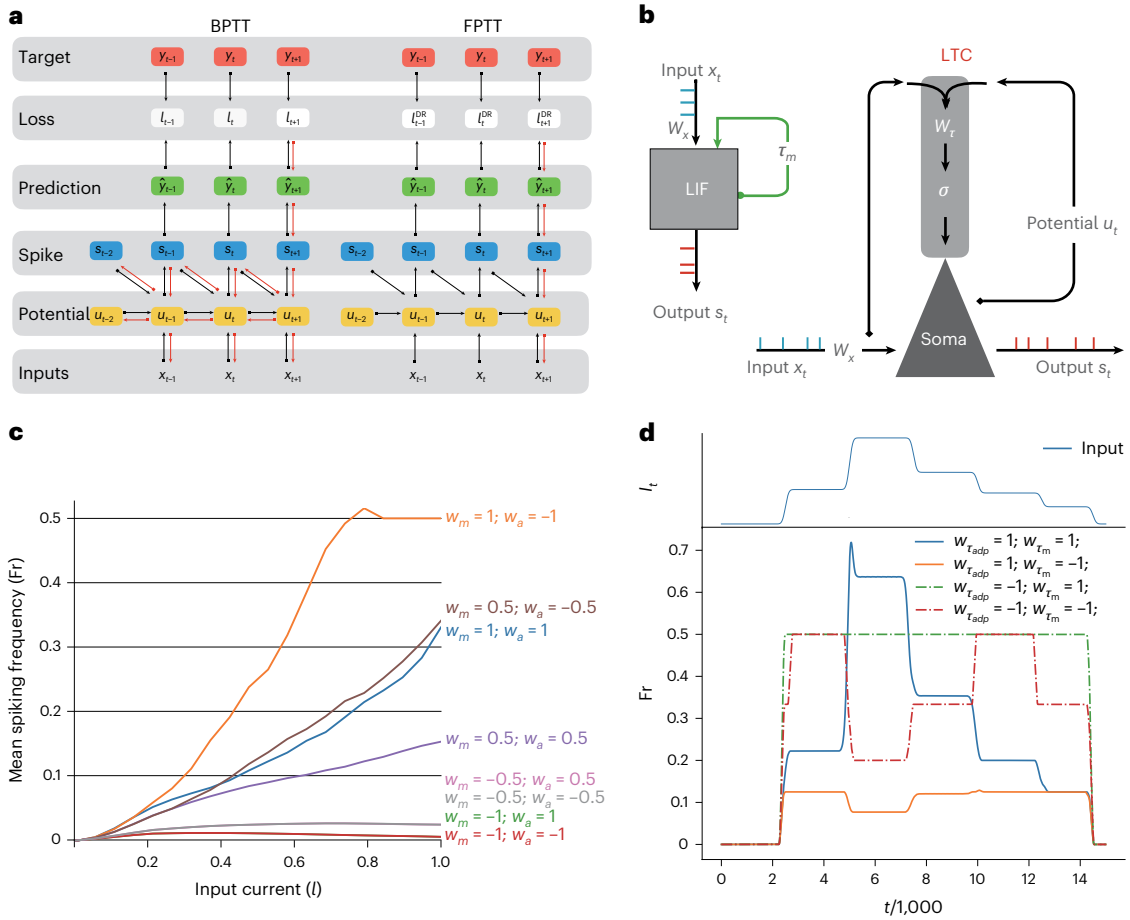

Fig. 1 | FPTT and LTC spiking neurons. a, Roll-out of the computational graph of a spiking neuron as used for BPTT (left) and FPTT (right). **b**, Illustration of LIF (left) and LTC (right) spiking neurons as recurrent network structures. **c**, *F*–*I* curve of an LTC spiking neuron for combinations of effective LTC weights $W_\tau = \{W_{\tau_m}, W_{\tau_{\text{adp}}}\}$: $W_{\tau_m} = \{-1, -0.5, 0.5, 1\}$ and $W_{\tau_{\text{adp}}} = \{-1, -0.5, 0.5, 1\}$ associated with respective dynamic time-constant functions σ subject to input current I. **d**, 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 $W_{\tau_{\text{adp}}} = \{-1, 1\}$.

currents: as shown in Fig. 1c, 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. 1d, samples of trained behaviour are shown in Supplementary Fig. 5).

Experiments

We demonstrate the effectiveness of FPTT training for LTC-SNNs on several well-known temporal classification benchmarks, including the Add-task as in ref. 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). Moreover, we 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 28 × 28 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.

https://doi.org/10.1038/s42256-023-00650-4

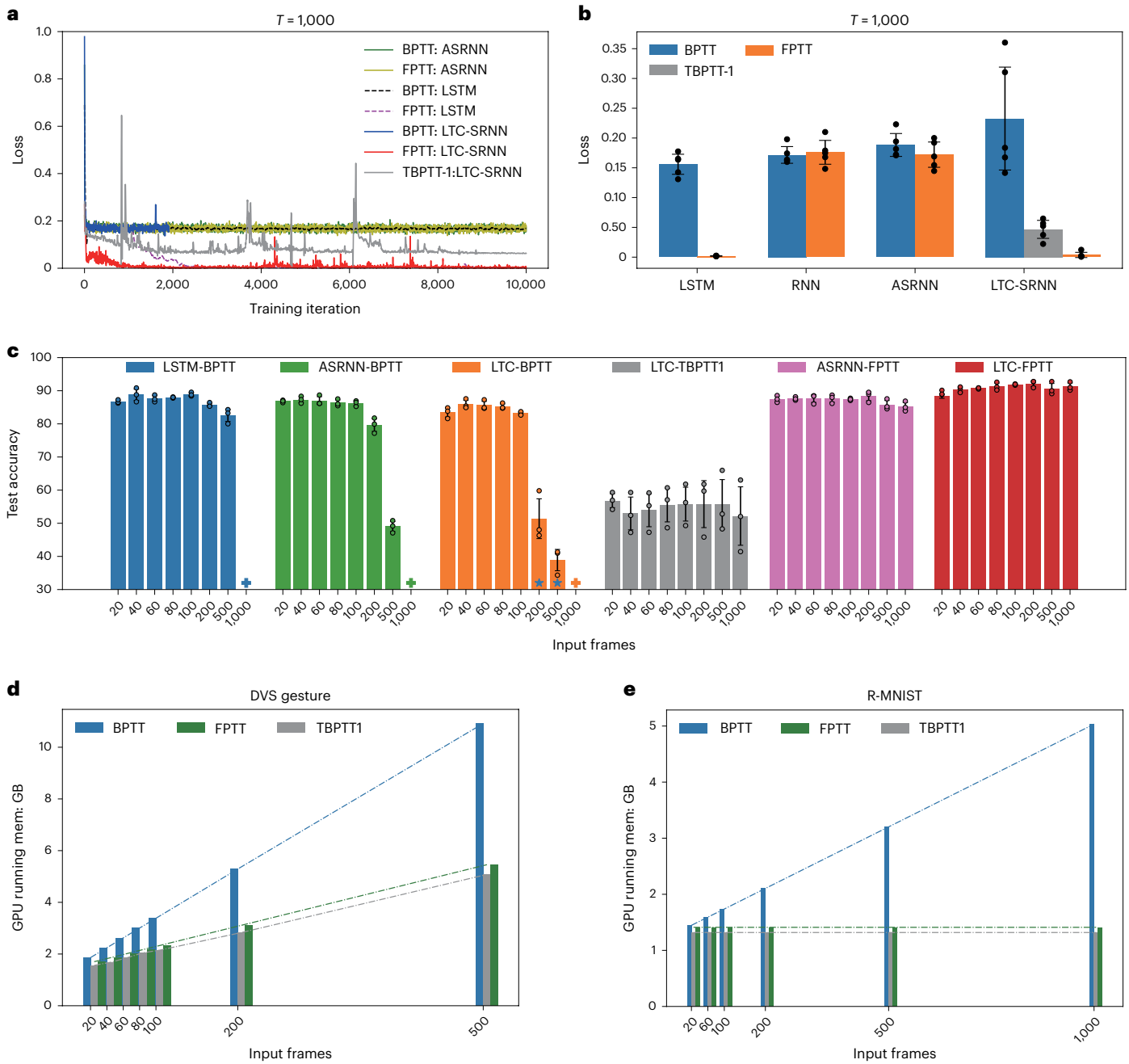

Fig. 2 | Performance evaluation of SNNs. a, Example loss-curves for networks trained on the Add-task with sequence length $T = 1000$ for BPTT, Truncated-BPTT for truncation length 1 (TBPTT-1, equivalent to BP at each time step) and FPTT. The loss of BPTT-trained LTC-SNN becomes NaN after 2,000 iterations. **b**, Corresponding average loss and s.d. of the last 100 training iterations (obtained from five runs). **c**, Plot of test accuracy for BPTT, T-BPPT1 and FPTT trained shallow networks on the DVS-GESTURE dataset. Accuracy bar average and s.d. over three 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, TBPTT-1 and FPTT.

FPTT-SNN requires LTC-SNs

We apply both BPTT and FPTT to the Add-task as originally studied in ref. 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 1,000, example loss-curves are plotted in Fig. 2a and averaged converged losses in Fig. 2b. We find that as in ref. 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—BPTT truncated to one timestep (TBPTT-1), corresponding to applying plain back-propagation (BP) at every timestep, does not diverge but also does not fully converge. 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 (Supplementary Table 4), 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 (Supplementary Fig. 4).

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 ref. 16, ranging from 20 to 1,000 frames. For each frame-encoded dataset, we train various network 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 (Supplementary Fig. 1a,b).

The results for different sampling frequencies are shown in Fig. 2c. 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.

By 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. 2c. 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. Applying plain BP at every timestep (TBPPT1) does converge for large frame-lengths but, as expected, achieves poor accuracy. Together, 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 1,000 frame-length data, graphical processing unit (GPU) memory (24 GB) was insufficient for training LSTM, ASRNN and LTC-SNN.

Comparing sparsity (average firing rate) for the different SNN models, we find no meaningful differences (Supplementary Fig. 1c); parameter-matched networks showed similar performance as unit-matched networks, and the inclusion of an auxiliary loss[8] only aided BPTT-trained LSTMs but not BPTT-trained SRNNs. Making only the membrane-decay time-constant dynamic has a small negative impact (ASRNN; Supplementary Table 1).

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. 2d), 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 1,000 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 increases linearly (Fig. 2e).

FPTT with LTC-SNs 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 2 we compare the LTC-SNNs to existing state-of-the-art online and offline SNNs. We find that LTC-SCNNs trained with FPTT consistently and substantially outperform SNNs trained with online BPTT approximations like OSTL and e-prop. Compared with 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 versus BTTP trained networks were lower by up to a factor of 5 (Supplementary Table 3) while the training time was only slightly longer (Supplementary Table 4).

Large-scale object detection. 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[22,34]. The 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 6.2 million spiking neurons, two convolutional output layers and 14 million parameters in total, illustrated in Supplementary Fig. 2a. 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 14 GB 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, for example four or eight 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 four reads and 53.25% at eight reads (Fig. 3a) on the VOC dataset (Methods); Fig. 3b shows examples of the detected and classified objects from author-acquired outside image captures. 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. Figure 3c shows example activity from neurons from respectively a shallow (near input) and deep network layer while the network processes a stream of images from webcam pointing either to a monitor running an author-collected video or the office environment: 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 8,000 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 RNNs, FPTT, can be successfully applied to long sequence learning with recurrent SNNs using LTC-SNs. Compared with 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 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 (Supplementary Fig. 4), the optimal degree of network recurrency remains to be determined and may allow for further efficiency and accuracy improvements[35]. LTC-SNs maintain binary communication

https://doi.org/10.1038/s42256-023-00650-4

Table 2 | Test accuracy of deep SRNNs/SCNNs on various tasks

Task	Online baseline		This work		Offline state of the art	
S-MNIST	-		FPTT+LTC	**97.37%**	BPTT+ASRNN[1]	***98.7%***
PS-MNIST	-		FPTT+LTC	**93.23%**	BPTT+ASRNN[1]	***94.3%***
R-MNIST	OSTL + SNU[12,46]	95.34%	FPTT+LTC	***98.63%***	BPTT+SNU[46]	97.72%
MNIST	DECOLLE+SNN[47]	98.0%	FPTT+LTC	**99.62%**	BPTT+PLIF[16]	***99.72%***
Fashion MNIST	EMSTDP+SNN[48]	85.3%	FPTT+LTC	**93.58%**	BPTT+PLIF[16]	***94.38%***
DVS-GESTURE	DECOLLE+SNN[49]	95.54%	FPTT+LTC	**97.22%**	BPTT+PLIF[16]	***97.57%***
DVS-CIFAR10	-		FPTT+LTC	**73.2%**	BPTT+PLIF[16]	***74.8%***

We apply FPTT-trained LTC-SNNs with one recurrent layer comprising 512 neurons on the (P)S-MNIST tasks. For RMNIST, an identical network structure as in ref. 12 is used, and for the remaining benchmarks the network structures match ref. 16. Bold denotes state-of-the-art online performance; italics denotes overall SNN state-of-the-art.

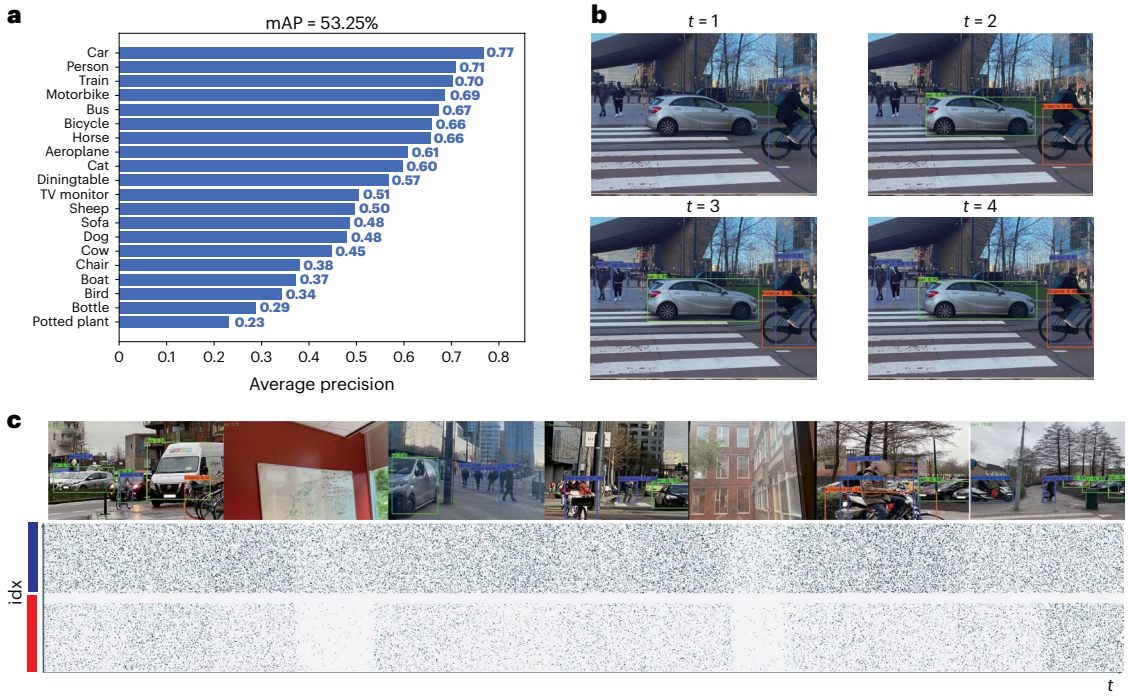

Fig. 3 | Spiking-YOLO-v4 (SPYv4). a, The mAP and components for classification and recognition of 20 different kinds of objects in SPYv4 on the PASCAL VOC07 dataset. **b**, An example of object recognition with Spiking-YOLO on a sequence of author-collected cell-phone images in Rotterdam, the Netherlands. Objects are localized and identified for each image independently. **c**, SPYv4 applied to streaming video as obtained from a webcam moving between filming an author-collected video clip and the surrounding environment. Top: example images, bottom: raster plot of spiking activity for 100 spiking neurons randomly drawn from the shallow and deep layer.

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 modelling 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[36,37], where the neuron's apical tuft may calculate a modulating term acting on the computation in the soma[38] that could act similar to the trainable LTCs used in this work. In a similar vein, learning rules derived from weight-specific traces may relate to synaptic tags[39,40] and are central to biologically plausible theories of learning working memory[41]. 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 (for example, for the DVS-GESTURE task; Supplementary Table 5). These considerations suggest that 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 back-propagation 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[42,43] 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 comprising 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 $\mathcal{L}_t$ and also a dynamic regularization penalty $\mathcal{R}_t$ based on previously observed losses $\ell_t^{\text{dyn}} = \mathcal{L}_t + \mathcal{R}_t$ (Supplementary Appendix A). In FPTT, the empirical objective $\mathcal{L}(y_t, \hat{y}_t)$ 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 ref. 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 $\mathcal{R}_t$ is as follows:

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

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

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

Here, the state vector $\bar{W}_t$ summarizes 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 equation (1c), the loss $\nabla l_t(W_{t+1})$ is estimated as in ref. 8, avoiding propagating the gradient through equation (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 modelling 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 P and target label distribution Q:

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

where $\beta \in [0, 1]$; l_t^{CE} is the classical cross-entropy for a classification loss and $l_t^{\text{div}} = -\sum_{\bar{y}} Q(\bar{y}) \log \hat{P}(\bar{y})$ is the divergence term. We use the auxiliary loss in all experiments as in ref. 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 BP without BPTT—the absence of these regularization terms drastically reduces accuracy, both for plain BP as well as for truncated BPTT (Supplementary Fig. 6).

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 ref. 8 to exploit GPU parallelism; reduction to batch-size = 1 yielded similar results (for example, Supplementary Fig. 3).

An SNN comprising spiking neurons that 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:

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

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 (for example 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 $\ell_t = \mathcal{L}(y_t, \hat{y}_t)$. 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 $\hat{f}'_s(u_t, \theta)$ to approximate this partial term[1].

The computational graph of BPTT is illustrated in Table 1 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 $\ell_t^{\text{dyn}} = \mathcal{L}(y_t, \hat{y}_t) + \mathcal{R}(\bar{W}_t)$. The update function, equation (6), then becomes:

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

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 modelled 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 $\alpha = \exp(-\text{d}t/\tau_m) = \sigma(\text{Dense}[x_t, u_{t-1}])$, for non-convolutional networks, or using a 2D convolution for spiking convolutional networks, $\alpha = \exp(-\text{d}t/\tau_m) = \sigma(\text{Conv}([x_t, u_{t-1}]))$, 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 LTC-SN i defined as:

$$\tau_{\text{adp}}\text{ update}: \rho^i = \exp(-dt/\tau^i_{\text{adp}}) = \sigma\left(\text{Dense}_{\text{adp}}\left[x_t, b^i_{t-1}\right]\right) \tag{8a}$$

$$\tau_m\text{ update}: \alpha^i = \exp(-\text{d}t/\tau^i_m) = \sigma\left(\text{Dense}_m\left[x_t, u^i_{t-1}\right]\right) \tag{8b}$$

$$\theta_t \text{ update}: b_t^i = \rho^i b_{t-1}^i + (1-\rho^i)s_{t-1}^i;\ \theta_t^i = 0.1 + 1.8b_t^i \quad (8c)$$

$$u_t \text{ update}: \mathrm{d}u^i = -u_{t-1}^i + x_t;\ u_t^i = \alpha^i u_{t-1}^i + (1-\alpha^i)\mathrm{d}u^i \quad (8d)$$

$$\text{spike } s_t: s_t^i = f_s(u_t^i, \theta^i) \quad (8e)$$

$$\text{resetting}: u_t^i = u_t^i(1-s_t^i) + u_{\text{rest}}s_t^i, \quad (8f)$$

where the neuron uses an adaptive threshold θ_t as in the Adaptive Spiking Neurons[11], resting potential $u_{\text{rest}} = 0$, and time-constants τ_m^i and τ_{adp}^i are computed as LTCs 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 one neuron.

For the DVS-GESTURE dataset, each frame is a 128 × 128 size image with two channels. Each sample in the DVS-GESTURE dataset was split into fixed-duration blocks as in ref. 16, where each block is averaged to a single frame. This conversion results in sequences of up to 1,000 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 × 128 image into a 32 × 32 image by averaging each 4 × 4 pixel block. The 2D image at each channel is then flattened into a 1D vector of length 1,024. 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. All networks were trained for 30 epochs using the Adam optimizer with initial learning rate 3×10^{-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 ref. 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 ref. 29 and ref. 8 for FPTT training. Specifically, in (P)S-MNIST, we used a shallow network with one recurrent layer comprising 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[44] with a batch size of 128 using 200 training epochs. We set the initial learning rate to 3×10^{-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 87 ms ± 13 ms (S-MNIST) and 65 ms ± 12 ms (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 ref. 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 ref. 16: an SCNN with three convolutional layers, one dense layer and one 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 1×10^{-3}. For the DVS-GESTURE and DVS-CIFAR10 datasets, we also follow the high-performance architecture from ref. 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[44] with a batch size of 16 and initial learning rate of 1×10^{-3}.

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

Spiking Tiny YOLOv4

The SPYv4 network follows the Tiny YOLO-v4 architecture[22,34]. 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 the Pascal VOC dataset[17] as was done in ref. 18: the network was trained using a combination of VOC2007 and VOC2012 (16,551 images), and evaluated on the validation dataset of VOC2007 (4,952 images).

For target detection, we use a threshold on the intersection over union (IOU), that is, 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[34] and label smoothing during training to obtain better performance. In the mosaic enhancement, the network uses a mosaic of four 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 1×10^{-3}.

Reporting summary

Further information on research design is available in the Nature Portfolio Reporting Summary linked to this article.

Data availability

This paper utilized publicly available datasets. The MNIST dataset can be accessed at http://yann.lecun.com/exdb/mnist/, while the Fashion MNIST dataset is available at https://github.com/zalandoresearch/fashion-mnist. The DVS-GESTURE dataset can be download at https://research.ibm.com/interactive/dvsgesture/, and the DVS-Cifar10 dataset can be obtained from https://figshare.com/articles/dataset/CIFAR10-DVS_New/4724671/2. The PASCAL VOC07 and VOC2012 datasets are publicly available at http://host.robots.ox.ac.uk/pascal/VOC/index.html. It is important to note that no software was used in the collection of data. Source data are provided with this paper.

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 and Zenodo[45] (https://zenodo.org/record/7498559).

References

1. Yin, B., Corradi, F. & Bohte, 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. Neurosci.* **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. Preprint at *bioRxiv* https://doi.org/10.1101/2020.11.17.386920 (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 Inf. Process. Syst.* **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 Comput.* **1**, 270–280 (1989).
11. Bellec, G. et al. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nat. Commun.* **11**, 3625 (2020).
12. Bohnstingl, T., Woźniak, S., Pantazi, A. & Eleftheriou, E. Online spatio-temporal learning in deep neural networks. In *IEEE Transactions on Neural Networks and Learning Systems* (IEEE, 2022).
13. He, Y. et al. A 28.2 μC 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* (IEEE, 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 (AAAI, 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 (IEEE, 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 (IEEE, 2021).
17. Everingham, M., Van Gool, L., Williams, C. K., Winn, J. & Zisserman, A. The PASCAL visual object classes (VOC) challenge. *Int. J. Comput. 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 (AAAI, 2020).
19. Chakraborty, B., She, X. & Mukhopadhyay, S. A fully spiking hybrid neural network for energy-efficient object detection. *IEEE Trans. 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 and Machine Intelligence* (IEEE, 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. Preprint at https://arxiv.org/abs/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. *Cognit. Sci.* **14**, 179–211 (1990).
25. Mozer, M. C. Neural net architectures for temporal sequence processing. In *Santa Fe Institute Studies on the Sciences of Complexity Proceedings* Vol. 15, 243 (Addison-Wesley, 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* 8–10 (Association for Computing Machinery, 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* (Association for Computing Machinery, 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. Preprint at *bioRxiv* https://doi.org/10.1101/2021.11.17.469025 (2021).
31. Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural Comput.* **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. Neurosci.* **11**, 309 (2017).
33. Gerstner, W., Kreiter, A. K., Markram, H. & Herz, A. V. Neural codes: firing rates and beyond. *Proc. Natl Acad. Sci. USA* **94**, 12740–12741 (1997).
34. Bochkovskiy, A., Wang, C.-Y. & Liao, H.-Y. M. YOLOv4: optimal speed and accuracy of object detection. Preprint at https://arxiv.org/abs/2004.10934 (2020).
35. Kalchbrenner, N. et al. Efficient neural audio synthesis. In *International Conference on Machine Learning* 2410–2419 (PMLR, 2018).
36. Sacramento, J., Ponte Costa, R., Bengio, Y. & Senn, W. Dendritic cortical microcircuits approximate the backpropagation algorithm. *Adv. Neural Inf. Process. Syst.* **31**, 8721–8732 (2018).
37. Beniaguev, D., Segev, I. & London, M. Single cortical neurons as deep artificial neural networks. *Neuron* **109**, 2727–2739 (2021).
38. 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).
39. Frey, U. & Morris, R. G. Synaptic tagging and long-term potentiation. *Nature* **385**, 533–536 (1997).
40. 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. USA* **108**, 12931–12936 (2011).
41. 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 Comput. Biol.* **11**, e1004060 (2015).

42. Pozzi, I., Bohte, S. & Roelfsema, P. Attention-gated brain propagation: how the brain can implement reward-based error backpropagation. *In Adv. Neural Inf. Process. Syst.* **33**, 2516–2526 (2020).
43. Scellier, B. & Bengio, Y. Equilibrium propagation: bridging the gap between energy-based models and backpropagation. *Front. Comput. Neurosci.* **11**, 24 (2017).
44. Kingma, D. P. & Ba, J. Adam: a method for stochastic optimization. In *3rd International Conference on Learning Representations (ICLR)* 1–15 (2015).
45. Yin, B. byin-cwi/sFPTT: Training SNN via FPTT. *Zenodo* https://doi.org/10.5281/ZENODO.7498559 (2023).
46. 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).
47. Zou, Z. et al. Memory-inspired spiking hyperdimensional network for robust online learning. *Sci. Rep.* **12**, 7641 (2022).
48. 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* (Association for Computing Machinery, 2019).
49. Kaiser, J., Mostafa, H. & Neftci, E. Synaptic plasticity dynamics for deep continuous local learning (DECOLLE). *Front. Neurosci.* **14**, 424 (2020).

Acknowledgements

B.Y. is supported by the Nederlandse Organisatie voor Wetenschappelijk Onderzoek, Toegepaste en Technische Wetenschappen (NWO-TTW) Programme 'Efficient Deep Learning' (EDL) P16-25. S.M.B. is supported by the European Union (grant agreement 7202070 'Human Brain Project'). The authors are grateful to H. Corporaal for reading the manuscript and providing constructive remarks.

Author contributions

B.Y., F.C. and S.M.B. conceived the experiments. B.Y. conducted the experiments. B.Y., F.C. and S.M.B. analysed the results. All authors reviewed the manuscript.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s42256-023-00650-4.

Correspondence and requests for materials should be addressed to Bojian Yin.

Peer review information *Nature Machine Intelligence* thanks Catherine Schuman and the other, anonymous, reviewer(s) for their contribution to the peer review of this work.

nature research

Corresponding author(s): Bojian Yin

Last updated by author(s): Jan 3, 2023

Reporting Summary

Nature Research wishes to improve the reproducibility of the work that we publish. This form provides structure for consistency and transparency in reporting. For further information on Nature Research policies, see our Editorial Policies and the Editorial Policy Checklist.

Statistics

For all statistical analyses, confirm that the following items are present in the figure legend, table legend, main text, or Methods section.

n/a	Confirmed	
☐	☒	The exact sample size (*n*) for each experimental group/condition, given as a discrete number and unit of measurement
☐	☒	A statement on whether measurements were taken from distinct samples or whether the same sample was measured repeatedly
☒	☐	The statistical test(s) used AND whether they are one- or two-sided *Only common tests should be described solely by name; describe more complex techniques in the Methods section.*
☒	☐	A description of all covariates tested
☒	☐	A description of any assumptions or corrections, such as tests of normality and adjustment for multiple comparisons
☐	☒	A full description of the statistical parameters including central tendency (e.g. means) or other basic estimates (e.g. regression coefficient) AND variation (e.g. standard deviation) or associated estimates of uncertainty (e.g. confidence intervals)
☒	☐	For null hypothesis testing, the test statistic (e.g. *F*, *t*, *r*) with confidence intervals, effect sizes, degrees of freedom and *P* value noted *Give P values as exact values whenever suitable.*
☒	☐	For Bayesian analysis, information on the choice of priors and Markov chain Monte Carlo settings
☒	☐	For hierarchical and complex designs, identification of the appropriate level for tests and full reporting of outcomes
☒	☐	Estimates of effect sizes (e.g. Cohen's *d*, Pearson's *r*), indicating how they were calculated

Our web collection on statistics for biologists contains articles on many of the points above.

Software and code

Policy information about availability of computer code

Data collection	All datasets in the paper are public available. No software was used for data collection
Data analysis	The datasets analysed during the current study are publicly available in the https://github.com/byin-cwi/sFPTT

For manuscripts utilizing custom algorithms or software that are central to the research but not yet described in published literature, software must be made available to editors and reviewers. We strongly encourage code deposition in a community repository (e.g. GitHub). See the Nature Research guidelines for submitting code & software for further information.

Data

Policy information about availability of data

All manuscripts must include a data availability statement. This statement should provide the following information, where applicable:

- Accession codes, unique identifiers, or web links for publicly available datasets
- A list of figures that have associated raw data
- A description of any restrictions on data availability

All datasets in the paper are public available.

Field-specific reporting

Please select the one below that is the best fit for your research. If you are not sure, read the appropriate sections before making your selection.

☒ Life sciences ☐ Behavioural & social sciences ☐ Ecological, evolutionary & environmental sciences

For a reference copy of the document with all sections, see nature.com/documents/nr-reporting-summary-flat.pdf

Life sciences study design

All studies must disclose on these points even when the disclosure is negative.

Sample size	*Describe how sample size was determined, detailing any statistical methods used to predetermine sample size OR if no sample-size calculation was performed, describe how sample sizes were chosen and provide a rationale for why these sample sizes are sufficient.*
Data exclusions	*Describe any data exclusions. If no data were excluded from the analyses, state so OR if data were excluded, describe the exclusions and the rationale behind them, indicating whether exclusion criteria were pre-established.*
Replication	*Describe the measures taken to verify the reproducibility of the experimental findings. If all attempts at replication were successful, confirm this OR if there are any findings that were not replicated or cannot be reproduced, note this and describe why.*
Randomization	*Describe how samples/organisms/participants were allocated into experimental groups. If allocation was not random, describe how covariates were controlled OR if this is not relevant to your study, explain why.*
Blinding	*Describe whether the investigators were blinded to group allocation during data collection and/or analysis. If blinding was not possible, describe why OR explain why blinding was not relevant to your study.*

Reporting for specific materials, systems and methods

We require information from authors about some types of materials, experimental systems and methods used in many studies. Here, indicate whether each material, system or method listed is relevant to your study. If you are not sure if a list item applies to your research, read the appropriate section before selecting a response.

Materials & experimental systems

n/a	Involved in the study
☒	☐ Antibodies
☒	☐ Eukaryotic cell lines
☒	☐ Palaeontology and archaeology
☒	☐ Animals and other organisms
☒	☐ Human research participants
☒	☐ Clinical data
☒	☐ Dual use research of concern

Methods

n/a	Involved in the study
☒	☐ ChIP-seq
☒	☐ Flow cytometry
☒	☐ MRI-based neuroimaging