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

Bojian Yin (byin@cwi.nl)
CWI https://orcid.org/0000-0002-5074-4337
Federico Corradi
Eindhoven University of Technology
Sander Bohte
CWI https://orcid.org/0000-0002-7866-278X

Article

Keywords:

Posted Date: June 15th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1625930/v1

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 standard recurrent neural networks. Still, these learning 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 '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 train in an end-to-end fashion the first deep and performant spiking neural network for object localization and recognition, demonstrating for the first time the possibility of training large-scale complex spiking neural network architectures online and on long temporal sequences.

Introduction

The 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, limiting 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 data complexity, and low time-complexity approximations to BPTT like e-prop[11] or OSTL[12] at best approach BPTT performance. In addition, 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 a novel online learning algorithm, Forward Propagation Through Time (FPTT), for online learning in recurrent networks. FPTT differs from BPTT in that it does not calculate a gradient through time. Instead, it derives from federated learning approaches and considers learning-through-time as a coordinated consensus problem[14]. FPTT enables immediate, online learning by updating the network parameters via the optimization of an instantaneous risk function which includes a dynamically time-evolving regularizer captured by local synapse-specific traces. This allows FPTT to optimize RNNs similar to feedforward networks and thus eliminates the dependence of the gradient calculation on the sum of products of partial gradients along the time dimension in BPTT. FPTT was demonstrated to improve long sequence training in Long Short-Term Memory networks (LSTMs) compared to BPTT while exhibiting linear $\Omega(T)$ computational cost per sample. However, as we demonstrate, a straightforward application of FPTT to recurrent SNNs on long sequence training does not improve performance on the same long sequence task, and we also observed this with standard RNNs. Therefore, we deduce that FPTT particularly benefits from the gating structure inherent in LSTM-style gated RNNs, which is lacking in standard RNNs and SRNNs.

We take inspiration from the concept of Liquid Time-Constant (LTCs)[15] and we introduce a novel 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 then integrate these neurons in SNNs that are trained with FPTT. We demonstrate that these LTC-SNNs trained with FPTT 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 adding task and the DVS gesture benchmark[16,17]. We also show how LTC-SNNs trained with FPTT can be applied to large convolutional SNNs, where we demonstrate novel state-of-the-art for online learning in recurrent SNNs on a number of standard benchmarks (S-MNIST, R-MNIST, DVS-GESTURE) and also show that large feedforward SNNs can be trained successfully in an online manner 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 new You Look Only Once (YOLO) LTC-SNN architecture for object detection on the Pascal Visual Object Classes (Pascal VOC) dataset[18]. Unlike classification tasks, object detection is a significantly more 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[19–21], requiring many thousands of time-steps at inference time, or small scale and inefficient SNNs with performance far removed from that of modern ANNs [22]. Our FPTT-trained YOLOv4[23] implementation – SPYv4 – uses 21 layers, 6.2M spiking neurons, and 14M parameters to achieve a new state-of-the-art (SoTa) for SNNs, exceeding the performance of converted ANNs with 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, including early work by Werbos[24], Elman[25] and Mozer[26]. In a recurrent network, to account for past influences on current activations, the network is 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[24]. 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 mixes of both RTRL and BPTT approaches[27]; they however exhibit prohibitive time and memory complexity[12], see Table 1**a** for an overview.

For networks of spiking neurons, 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]. Various SNNs trained with such surrogate gradients and BPTT now achieve competitive performance compared to classical RNNs[1,17,29]. In these and other studies, more intricate spiking neuron models, like those including adaptation, outperformed less complex models like standard Leaky-Integrate-and-Fire neurons[11]. Additionally, learning 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. We found this to be the case in particular for training networks with complex and more biologically detailed neuron models like Izhikevich and Hodgkin-Huxley models[30]. Furthermore, because the SNN performance heavily depends on hyperparameters related to the surrogate gradients, obtaining convergence in SNN networks is non-trivial. Moreover, the spike-triggered reset of the membrane potential due to refraction causes a vanishing gradient.

Approximations to BPTT like e-prop[11] achieve linear time complexity and have proven effective for many small scale benchmark problems and also large scale networks like cortical microcircuits[31]. Online Spatio-Temporal Learning (OSTL)[12] separates the spatial and temporal gradient calculations to derive weight updates in an online manner, but suffers from very high computational and memory costs for generic RNNs. In terms of trained accuracy, however, none of these approximations have been shown to outperform standard BPTT.

FPTT in SNNs

FPTT can be implemented directly on the computational graph of SNNs, similar to BPTT approaches. This is illustrated in Fig. 1**b**, and the corresponding algorithms are given in 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 weakens surrogate gradients' cumulative effect of avoiding or reducing the gradient vanishing or explosion problem.

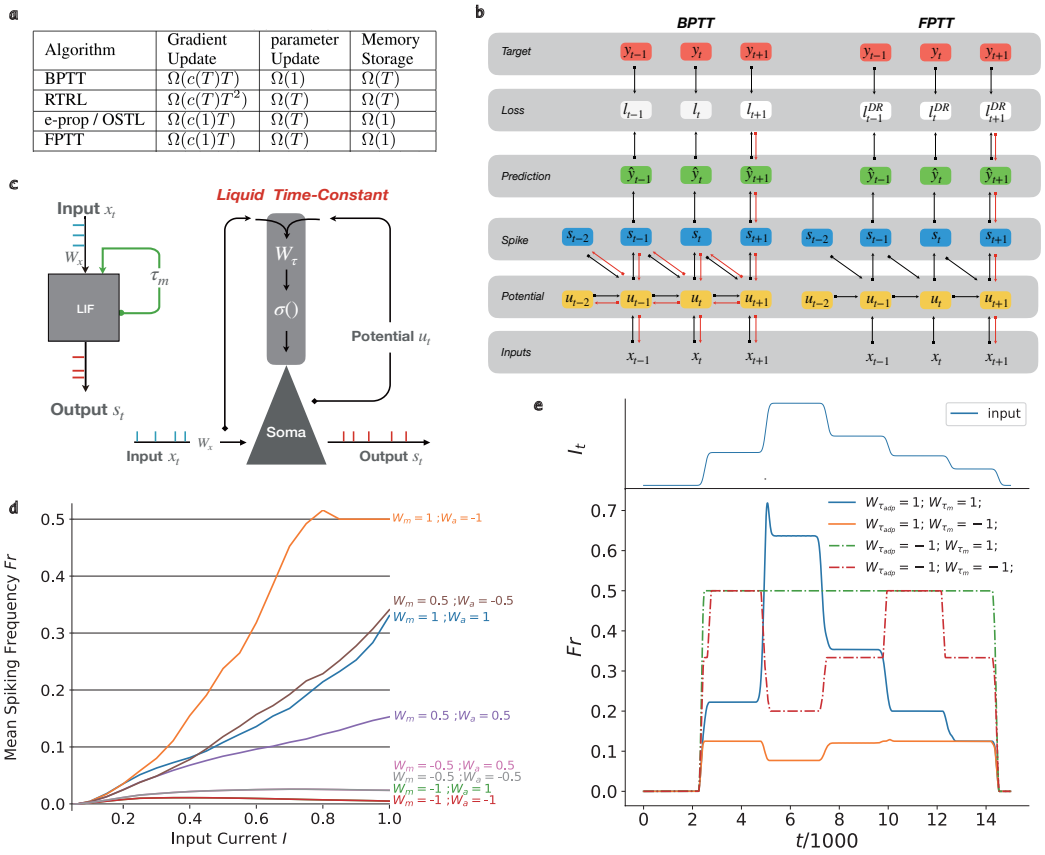

Figure 1. **a**, Computational complexity of gradients, parameter updates and memory storage per sample. The computational expense increases as the length of the sequence grows. i.e. $c(1) < c(T)$ after[8]. **b**, Roll-out of the computational graph of a spiking neuron as used for BPTT (left) and that of FPTT(right); **c**, LIF neuron and Liquid Time-Constant spiking neuron (LTC-SN) as recurrent network structures. **d**, F-I curve of a LTC-SN spiking neuron model for different combinations of effective LTC weights $W_m = \{-1 - 0.5, 0.5, 1\}$ and $W_a = \{-1 - 0.5, 0.5, 1\}$ associated with respective dynamic time constant functions σ subject to input current I. **e**, Response Fr of LTC-SN to varying current input I (top) for different combinations of effective LTC weights $W_{\tau_m} = \{-1, 1\}$ and $W_{\tau_{adp}} = \{-1, 1\}$.

As we will show, FPTT applied directly to SNNs like the Adaptive Spiking Recurrent Neural Networks (ASRNNs) from[1] was insufficient to obtain the improvements reported in [8] – and we found this observation also held for standard non-spiking RNNs (not shown). Noting that FPTT was successfully applied to RNNs with gating structures (LSTM), we introduce the Liquid Time-Constant Spiking Neuron (LTC-SN) model to generate a spiking neural unit with a similar gating structure: LTC-SNNs. We observe that to some degree, 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 inputs. Inspired also by the work by Hasani et al.[15], the LTC-SN unit's internal time constants are a learned function of the inputs and hidden states of the network (illustrated in Fig. 1**c**). For spiking neurons that included adaptation, both the time-constant of the membrane potential decay and the time-constant of the adaptation decay become 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. Moreover, 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**).

Experiments

We demonstrate the effectiveness of the FPTT algorithm with LTC-SNNs on several classical benchmarks, comparing to[8] (the Add-task) and 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-trained LTC-SNNs allows for large SNN applications like object localization.

The **Add-Task**[32] is used to evaluate the ability of RNNs to maintain long-term 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 real-valued items sampled uniformly from [0, 1], x_2 is a binary sequence of 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[16] 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[33]. The **Sequential and Permuted-Sequential** MNIST (S-MNIST, PS-MNIST) datasets were developed to measure sequence recognition and memory capabilities of learning algorithms: a grey input image of shape 28-by-28 is reshaped into a one-dimensional sequence consisting of 784 time steps. At each time step, only one pixel entered the network as an input. The permuted-MNIST dataset is generated by performing a fixed permutation on the sequential 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[34]. 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 first spiking layer of the network, repeated 20 times to mimic a constant input stream. For object localization, we used the **PASCAL VOC** benchmark[18] 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 trained with BPTT fails to converge to zero loss, while the same LSTM does converge when trained with FPTT. For SNNs, we find that ASRNNs trained with either FPTT or BPTT do not fully converge, similar to the BPTT-trained LSTM: for the more complex LTC-SNNs trained with BPTT, we find that learning rapidly diverges due to exploding gradients. LTC-SNNs trained with FPTT however successfully minimize the loss, similar to the FPTT-trained LSTM network. Of note, we observed that FPTT enabled convergence when training networks of other complex spiking neurons, such as Izhikevich and Hodgkin-Huxley spiking models, which failed to converge with BPTT. However, we did not observe performance improvements over simple spiking neurons models as the Leaky-Integrate-and-Fire model (not shown).

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[17], ranging from 20 to 1000 frames, and examine the performance of various neural network models, training methods and loss functions. For each frame-encoded dataset, we train networks for a fixed number of epochs with an identical number of neural units and report the best performance. The results for different sampling frequencies are shown in Fig. 2**c**; networks either converged before the final epoch or diverged (Fig S1a,b).

Of all methods and networks, the LTC-SNN trained using FPTT achieved the best performance 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 substantially outperform the ASRNNs.

In contrast, the accuracy of both LTC-SNNs and ASRNNs trained with BPTT quickly deteriorates as sequence length increases, from 85.5% for 60 frames to 38.9% for 500 frames for the best performing LTC-SNN. For the LTC-SNNs, the networks did not converge for frame-lengths 200 and 500 (denoted by the star) and best validation accuracy is reported in Fig. 2a. 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 training with BPTT: when applied to the 1000 frame-length data, it places such high demands on GPU memory that it did not fit in memory (24GB) for LSTM, ASRNN and LTC-SNN (denoted by the + in the plot of Fig. 2c).

Comparing sparsity (average firing rate) for the different SNN models, we find no meaningful differences (Fig. S1c); parameter-matched networks showed similar performance as unit-matched networks, and the inclusion of the auxiliary loss only aided BPTT-trained LSTMs but not BPTT-trained SRNNs. Performance of BPTT-approximations like eProp proved highly memory intensive and was accuracy was less than exact BPTT. Making only the membrane-decay timeconstant dynamic has a small negative impact (ASRNN− (Table S1).

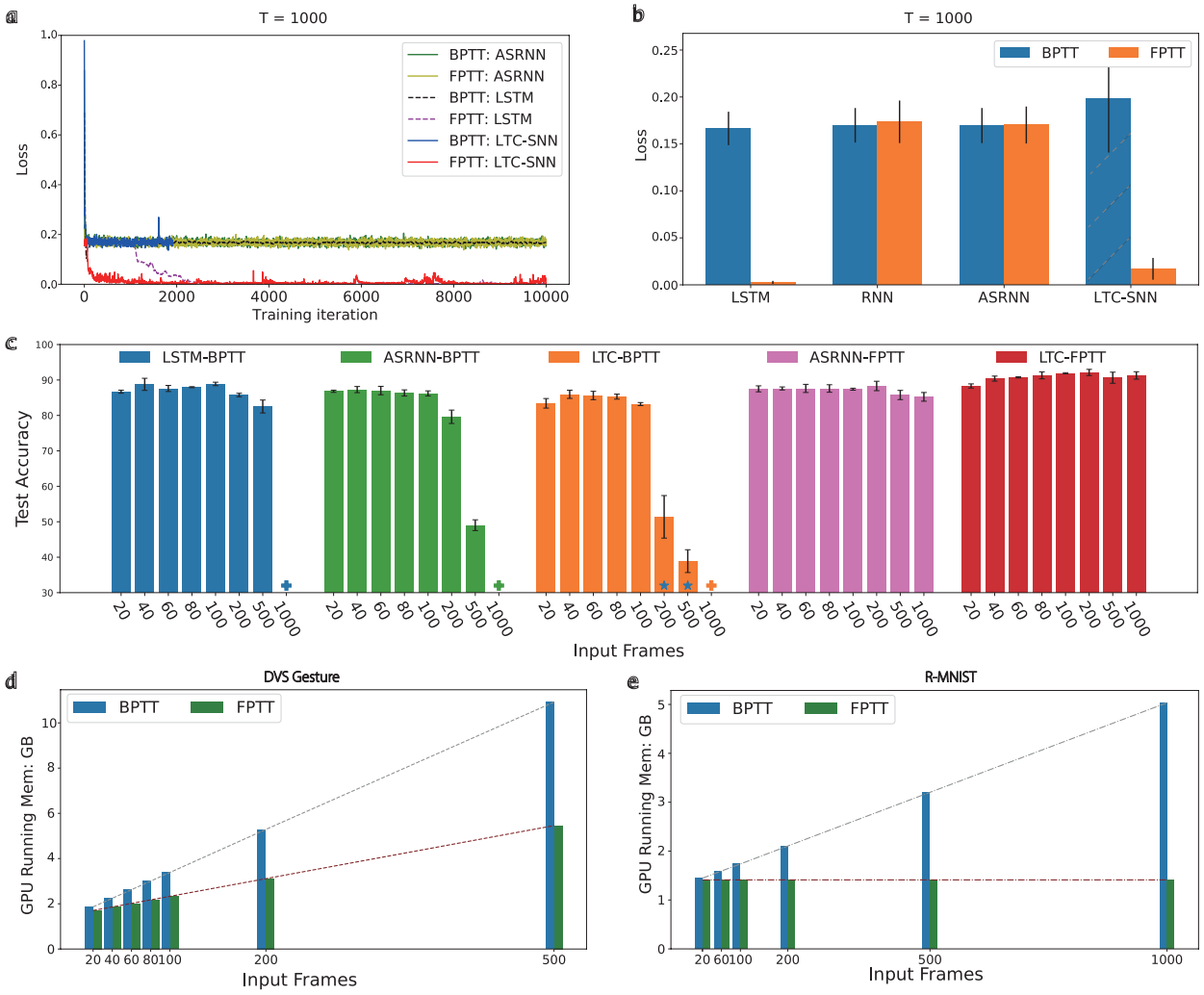

Figure 2. **a**, Example loss-curves for networks trained on the Add Task for a sequence of length 1000 time steps. The loss of LTC-SNN becomes *NaN* after 2000 iterations when training with BPTT. **b**, plots the loss of the last 100 training iterations averaged over 5 runs for sequence lengths. **c**, The Bar chart of Performance comparison between BPTT and FPTT on the DVS gesture dataset. **d**, Memory Efficiency: GPU Memory required for network training of the DVS Gesture dataset for different sequence length. **e** GPU Memory cost required for training of the R-MNIST dataset.

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 to encode each data sample. FPTT's increasing memory use can be attributed to the rapidly increasing size of the frame-encoded dataset, as it increases 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 a the R-MNIST classification problem, where longer sequence durations 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 BPTT. To demonstrate the power of FPTT as an online training method, we trained a number of deep spiking convolutional networks (SCNNs) comprised of LTC-SN neurons with FPTT for standard sequential benchmarks and compared them to the existing state-of-the-art. As shown in Table 1, we find that LTC-SCNNs 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 new SoTa for SNNs (PS-MNIST, R-MNIST) or achieve close to similar performance (S-MNIST, DVS-Gesture, DVS-Cifar10); the memory requirements

Table 1. Performance of deep SRNNs/SCNNs on various tasks.

Task	**Online**			This work	**Offline** [BPTT]	
	Algorithm	Network	Test Acc	LTC-SNN	Network	Test Acc
S-MNIST	e-prop[35]	LSNN	94.32%	**97.37%**	ASRNN[1]	98.7%
PS-MNIST	-	-	-	**94.77%**	ASRNN[1]	94.3%
R-MNIST	OSTL[12]	SNU[36]	95.54%	**98.63%**	SNU[36]	97.72%
MNIST	-	-	-	**99.62%**	PLIF[17]	99.72%
Fashion-MNIST	-	-	-	**93.58%**	PLIF[17]	94.38%
DVS-Gesture	DECOLLE[37]	SNN	95.54%	**97.22%**	PLIF[17]	97.57%
DVS-CIFAR10	-	-	-	**73.2%**	PLIF[17]	74.8%

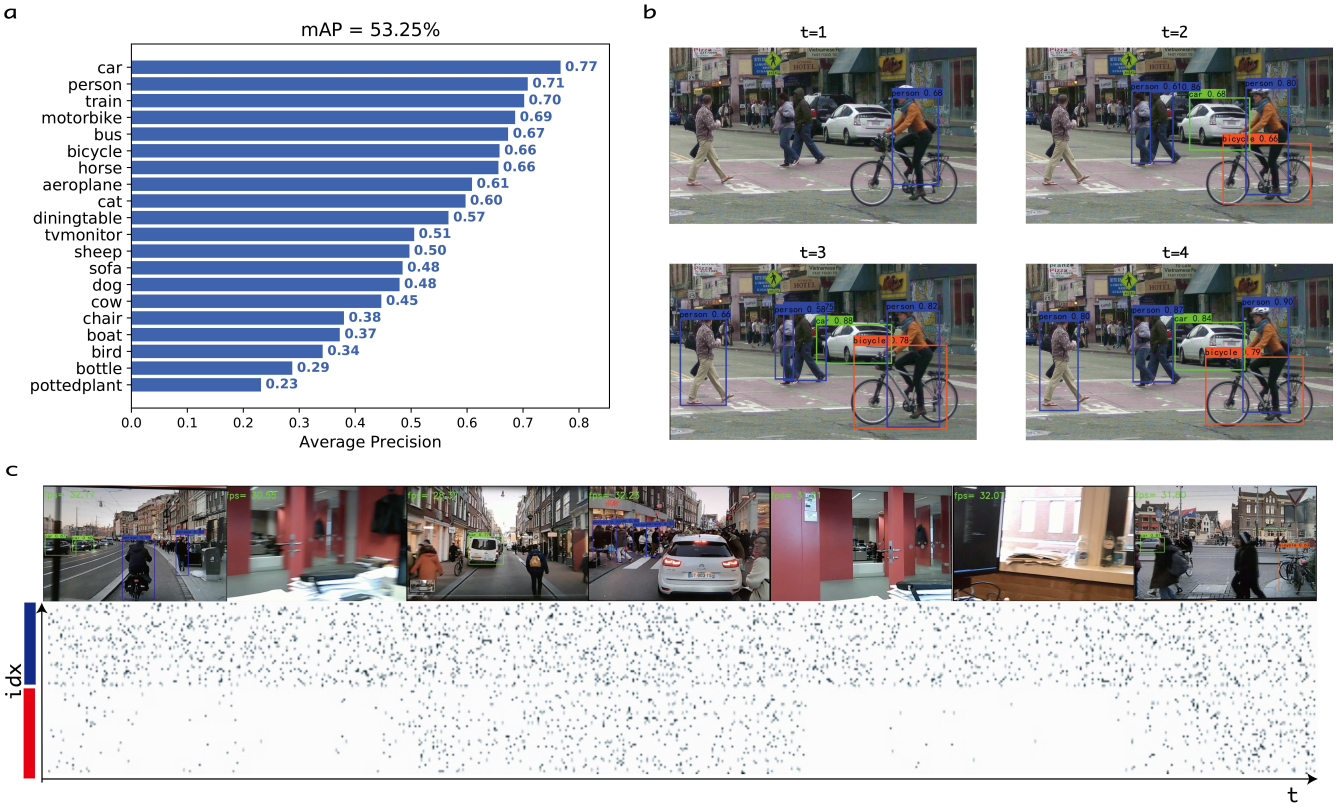

Figure 3. **a**, The mAP%.5 calculation in the classification and recognition of 20 different kinds of objects in SPYv4 on the PASCAL VOC07 dataset. **b**, An example of object recognition process of Spiking-YOLO on a sequence of images. Objects are localized and identified for each image independently. **c**, SPYv4 applied to streaming video (example images on top), with raster plot of spiking activity below. Spikes are shown for 100 random select neurons from the in first and last layer.

for FPTT vs. BTTP trained networks were lower by up to a factor of 5 (Table S2).

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 the power of FPTT-trained LTC-SNNs by directly training a large spike-based object-detection model based on the Tiny YOLO-v4 architecture[23,38].

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. This makes it both larger and deeper than previous end-to-end trained spiking models – the network architecture is illustrated in Fig.S2**a**. The network uses multiple reads of the input image, e.g. 4 or 8 times, as timesteps in the network to obtain the final result. Training a single timestep in the network requires less than 14GB of GPU memory, and as BPTT scales linearly with the number of timesteps, learning with BPTT in such a large network over multiple timesteps is infeasible.

Trained with FPTT, our network achieved a MAP@.5 of 51.38% at 4 reads, and 53.25% at 8 reads (Fig. 3**a**) on the VOC dataset. Examples of the detected and classified objects are shown in Fig. 3**b**. Neural activation in the network is highly sparse, as only about 10% of neurons are activated on average in each step of the network. When receiving direct camera inputs images, can perform inference at about 60 steps per second, processing 7 or 15 pictures per second on a workstation equipped with an NVIDIA RTX 3090 GPU; example activity from two different layers in the network is shown in Fig. 3**c**. The network activity also demonstrates how neurons deeper in the network fall silent when irrelevant stimuli are shown, while neurons closer to the inputs remain active. Earlier work like Spiking-YOLO[19] achieved MAP@.5 51.83% with 8000 simulation time steps. Our SPYv4 network thus outperforms these previous spike-based YOLO networks in performance, sparseness, and latency.

Discussion

We showed how a novel training approach, FPTT, can be successfully applied to long sequence learning with recurrent SNNs using novel 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 substantially outperforms online approximations of BPTT like OSTL and eProp, including a first demonstration of online learning of tasks like DVS-CIFAR10. When training large convolutional LTC-SNNs with FPTT, excellent performance is achieved, approaching or exceeding offline BPTT-based solutions.

To achieve effective online learning with FPTT, we introduced Liquid Time-Constant Spiking Neurons (LTC-SNs), where the time-constants in the neuron are computed 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.

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[39,40], where the apical tuft may calculate a modulating term acting on the computation in the soma[41], which 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[42,43] and are central to biologically plausible theories of learning working memory[44]. 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[30]. These considerations suggest variations of FPTT may be potential candidates for temporal credit mechanisms in the brain.

FPTT also holds promise for network quantization: FPTT uses both (a form of) synaptic traces in the form of $\bar{W}$ and the actual weights W, where the traces are only used for training. One could imagine networks where the two-parameter sets are each calculated with different quantizations, where W could potentially be computed with lower precision compared to $\bar{W}$. Once trained, only the lower precision weights W are then needed for inference. When combined with local error-backpropagation solutions like BrainProp[45], FPTT-training of LTC-SNNs can also likely be implemented fully locally and online 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 neuromorphic computing and investigations of biologically plausible neural processing.

Methods

Forward Propagation through Time. 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 $\ell_t^{dyn} = \mathscr{L}_t + \mathscr{R}_t$. 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 $\hat{y}_t$.

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:

$$\begin{aligned} \mathscr{R}(W) &= \frac{\alpha}{2} \parallel W - \bar{W}_t - \frac{1}{2\alpha}\nabla l_{t-1}(W_t) \parallel^2 \\ W_{t+1} &= W_t - \eta \nabla_W l(W)|_{W=W_t} \\ \bar{W}_{t+1} &= \frac{1}{2}(\bar{W}_t + W_{t+1}) - \frac{1}{2\alpha}\nabla l_t(W_{t+1}). \end{aligned} \tag{1}$$

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].

The 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 classification tasks however, the target value is only determined after processing the entire time series. To adapt FPTT to classification tasks, or rather, to perform online classification tasks, 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 $\beta = \frac{t}{T}$.

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.

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[17,29].

(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 shown in Fig1**a** and shows that the partial derivative term depends on two pathways, $\frac{\partial u_m}{\partial u_{m-1}} = \frac{\partial u_m}{\partial u_{m-1}} + \frac{\partial u_m}{\partial s_{m-1}} \frac{\partial s_{m-1}}{\partial u_{m-1}}$. This gradient expression implies that the error of the surrogate gradient accumulates and amplifies during the training process. 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.

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 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 function of input x_t and membrane potential u_{t-1} such that $\exp(-1/\tau) = \sigma(x_t, u_{t-1}|W_\tau)$; 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. In the network, time-constants as used in standard adaptive spiking neurons[1,11] are either calculated as a function $\tau_m^{-1} = \sigma(Dense([x_t, u_{t-1}]))$, for non-convolutional networks, or using a 2D convolution for spiking convolutional networks, $\tau_m^{-1} = \sigma(Conv(x_t + u_{t-1}))$. This results in a Liquid Time-Constant Spiking Neuron defined as:

$$
\begin{aligned}
\tau_{adp}\, update &: \rho = \tau_{adp}^{-1} = \sigma([x_t, b_{t-1}] \parallel W_{\tau_{adp}}) \\
\tau_m\, update &: \tau_m^{-1} = \sigma([x_t, u_{t-1}] \parallel W_{\tau_m}) \\
\theta_t\, update &: b_t = \rho b_{t-1} + (1-\rho) s_{t-1} \\
&\quad\ \theta_t = 0.1 + 1.8 b_t \\
u_t\, update &: du = (-u_{t-1} + x_t)/\tau_m \\
&\quad\ u_t = u_{t-1} + du \\
spike\, s_t &: s_t = f_s(u_t, \theta) \\
resting &: u_t = u_t(1 - s_t) + u_{rest} s_t,
\end{aligned}
\tag{8}
$$

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_m and τ_{adp} are computed as liquid time-constants.

Datasets For the Add-task, the trained networks consist of 128 recurrently connected neurons of respective types LTC-SNN (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[17], 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, 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 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 SRNNs were taken from[1].

To achieve high performance on the DVS Gesture and DVS-CIFAR10 datasets, we follow[17] and use 20 sequential frames, where the network makes a prediction only after reading the entire sequence. We use a spiking convolutional network following the structure: ConvK7C64S1P3-MPK2S2-ConvK7C128S1P3-MPK2S2-ConvK3C128S1P1-MPK2S2-ConvK3C256S1P1-MPK2S2-ConvK3C256S1P1-MPK2S2-ConvK3C512S1P1-MPK2S2-512D-11I[1]. These networks were optimized through Adamax[46] with a batch size of 16 and initial learning rate of 1e-3.

In (P)S-MNIST, we applied a shallow network with one recurrent layer comprised of 512 hidden neurons, and the output layer consists of 10 (number of classes) leaky integrator neurons. Networks are optimized using Adam[46] 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 R-MNIST, as in[12], we apply 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 an SCNN with 3 convolutional layers, 1 Dense layer and 1 leaky Integrator output layer: ConvK3C32S1P1-MPK2S2-ConvK3C128S1P1-MPK2S2-ConvK3C256S1P1-MPK2S2-512D-10I. The network was then optimized using Adamax with a batch size of 64 and an 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[23,38]. 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 an adding operation, ensuring that only binary spikes are used for information transfer between layers. Raw pixel values are fed

[1]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 layer and recurrent layer respectively with $output\ features = 512$. 10I indicates the leaky integrator with the $output\ feature = 10$.

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[18] as was done[19]: the network was trained using a combination of VOC2007 and VOC2012 (16551 images), and evaluated on the validation dataset of VOC2007 (4952 images). 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.

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., Maass, W., Pantazi, A. & Eleftheriou, E. Online spatio-temporal learning in deep neural networks. *arXiv preprint arXiv:2007.12723* (2020).
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. Acar, D. A. E. *et al.* Federated learning based on dynamic regularization. In *ICLR* (2020).
15. 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).
16. 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).
17. 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).
18. 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).
19. 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).
20. 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).
21. 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).

22. 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).

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

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

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

26. 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).

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

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. Yin, B. *Efficient Learning in Spiking Neural Networks*. Ph.D. thesis, TU/e (in preparation).

31. 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).

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

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

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

35. Bellec, G. E. F., Salaj, D., Subramoney, A., Legenstein, R. & Maass, W. Long short-term memory and learning-to-learn in networks of spiking neurons. In *Advances in Neural Information Processing Systems: NeurIPS* (2018).

36. 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).

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. 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).

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

41. 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).

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

43. 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).

44. 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).

45. 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).

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

Acknowledgements (not compulsory)

BY is supported by the NWO-TTW Programme "Efficient Deep Learning" (EDL) P16-25, and SB is supported by the European Union (grant agreement 7202070 "Human Brain Project").

Author contributions statement

B.Y., F.C. and S.B. conceived the experiment(s), B.Y. conducted the experiment(s), B.Y., F.C. and S.B. analysed the results. All authors reviewed the manuscript.

Additional information

Competing interests. The authors declare no competing interests.

Supplementary Information

Algorithm 1: Training SNN with BPTT

// Dataset $B = \{x_t, y_t\}_{t=0}^{T}$, # Epochs E
// Set Optimizer and learning rate η
// Initialize Weight 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
    $s_{h,t}, u_{h,t} = \hat{f}_s(x_{t-1}, [u_{h,t-1}, s_{h,t-1}] \| W_1)$
    $\hat{y}_t = \hat{f}_s(s_{h,t}, [u_{o,t}, s_{o,t}] \| W_2)$
  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 η
// Initialize weight W, 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
    $s_{h,t}, u_{h,t} = \hat{f}_s(x_{t-1}, [u_{h,t-1}, s_{h,t-1}] \| W_1)$
    $\hat{y}_t = \hat{f}_s(s_{h,t}, [u_{o,t}, s_{o,t}] \| W_2)$
    Loss: $\ell_t(W)$: $\ell_t(W) = \ell(y_t, \hat{y}_t)$
    Dynamic Loss:
    $\ell^{dyn}(W) = \ell_t(W) + \frac{\alpha}{2} \| W - \bar{W}_t - \frac{1}{2\alpha} \nabla \ell_{t-1}(W_t) \|^2$
    Update W:
      $W_{t+1} = W_t - \eta \nabla_W \ell(W)|_{W=W_t}$
    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 agorithm. Adapted from[8].

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-SNN. LTC^{-} denotes an LTC-SNN network where only the membrane time constant is dynamic, and the adaptation time constant is a learned fixed parameter. For the Eprop ASRNN, we use a single layer network with 2048 neurons, matching the number of parameters in the LTC-SNN.

Frames	BPTT						FPTT			Eprop
	LSTM+Aux	LSTM	LTC-SNN+Aux	LTC-SNN	ASRNN+Aux	$ASRNN^{+}$+Aux	LTC^{-}	LTC-SNN	ASRNN	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	**90.39**±0.71	87.51±0.85	75.46±1.73
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	75.92±0.87
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	75.42±0.86
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	77.54±1.34
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	75.46±1.15
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	75.33±1.02
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	+

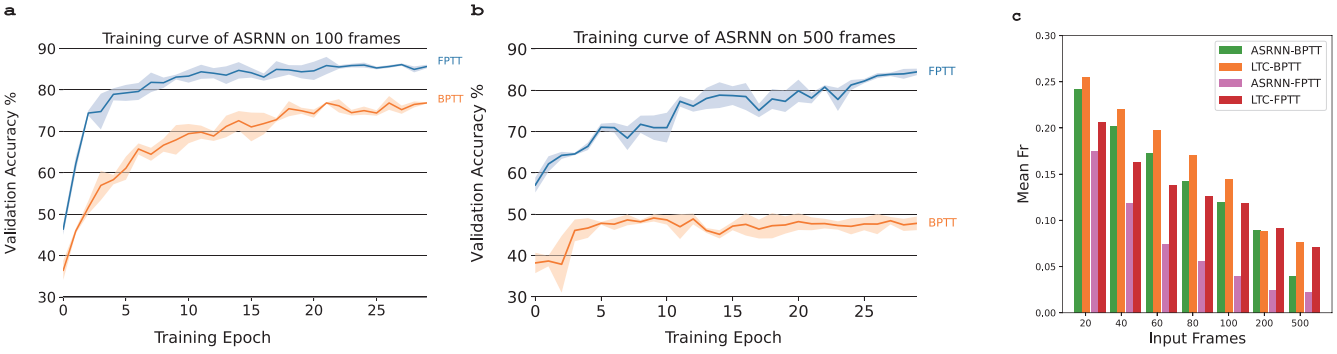

Figure S1. Learning curve of ASRNN trained via FPTT and BPTT on **a** 100 frames, and **b** 500 frames; **c**, Bar chart of mean firing rate (fr) comparison between BPTT and FPTT on the DVS gesture dataset. Each number in the table is the average of three runs.

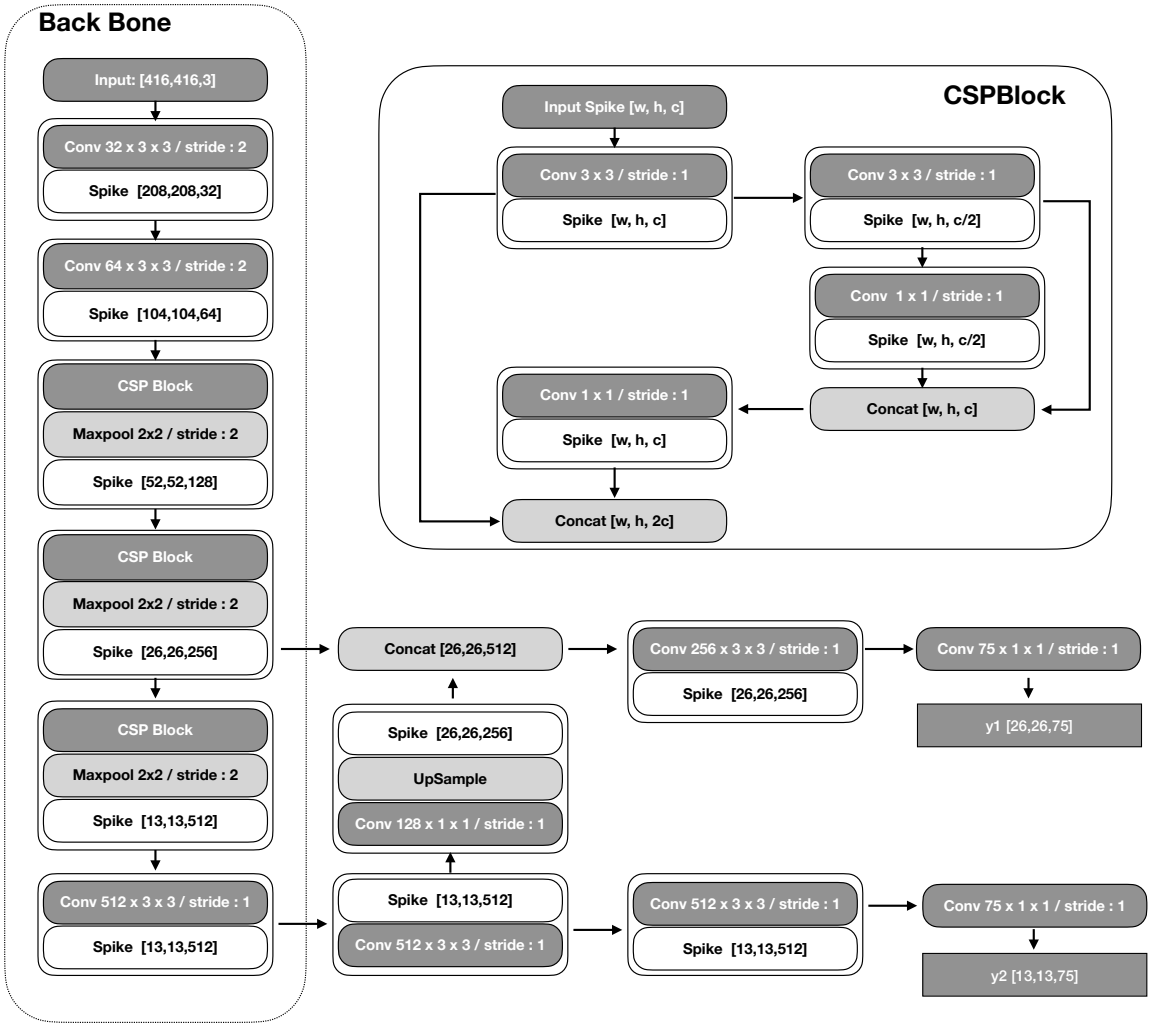

Figure S2. Spiking YOLOv4 (SPYv4) neural network architecture.

Table S2. 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

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- SupplementaryInformation.pdf