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

Yin, B.; Corradi, F.; Bohté, S.M.

DOI
10.48550/arXiv.2112.11231
10.1038/S42256-023-00650-4

Publication date
2023

Document Version
Submitted manuscript

Published in
Nature Machine Intelligence

ACCURATE ONLINE TRAINING OF DYNAMICAL SPIKING NEURAL NETWORKS THROUGH FORWARD PROPAGATION THROUGH TIME

A PREPRINT

Bojian Yin
CWI
Bojian.Yin@cwi.nl

Federico Corradi
IMEC
Federico.Corradi@imec.nl

Sander M. Bohté
CWI
S.M.Bohte@cwi.nl

November 14, 2022

ABSTRACT

The event-driven and sparse nature of communication between spiking neurons in the brain holds great promise for flexible and energy-efficient AI. Recent advances in learning algorithms have demonstrated that recurrent networks of spiking neurons can be effectively trained to achieve competitive performance compared to standard recurrent neural networks. Still, as these learning algorithms use error-backpropagation through time (BPTT), they suffer from high memory requirements, are slow to train, and are incompatible with online learning. This limits the application of these learning algorithms to relatively small networks and to limited temporal sequence lengths. Online approximations to BPTT with lower computational and memory complexity have been proposed (e-prop, OSTL), but in practice also suffer from memory limitations and, as approximations, do not outperform standard BPTT training. Here, we show how a recently developed alternative to BPTT, Forward Propagation Through Time (FPTT) can be applied in spiking neural networks. Different from BPTT, FPTT attempts to minimize an ongoing dynamically regularized risk on the loss. As a result, FPTT can be computed in an online fashion and has fixed complexity with respect to the sequence length. When combined with a novel dynamic spiking neuron model, the Liquid-Time-Constant neuron, we show that SNNs trained with FPTT outperform online BPTT approximations, and approach or exceed offline BPTT accuracy on temporal classification tasks. This approach thus makes it feasible to train SNNs in a memory-friendly online fashion on long sequences and scale up SNNs to novel and complex neural architectures.

Keywords Spiking neural network · FPTT · online learning · Liquid Time-Constant

1 Introduction

Recent work has demonstrated effective and efficient performance from spiking neural networks [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).

However, the imprecision of the surrogate gradient approach 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 can suffer from vanishing or exploding backpropagating gradients, limiting its applicability on long time sequences [8]. Alternative approaches like real-time recurrent learning (RTRL)[9] similarly exhibit excessive data-complexity, and low time-complexity approximations to BPTT like e-prop [10] or OSTL [11] 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 neuromorphic hardware [2]: as there is no notion of discrete-time steps in clock-less event-driven neuromorphic devices and time is continuous, off-chip SNNs need to be

Table 1: 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].

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	$\Omega(c(1)T)$	$\Omega(T)$	$\Omega(1)$
FPTT-K	$\Omega(c(K)T)$	$\Omega(K)$	$\Omega(T/K)$

trained on temporal sequences with extremely short time steps to mimic continuous-time characteristics and guarantee corresponding performance in real-life applications [12].

A novel learning algorithm, Forward Propagation Through Time (FPTT), was recently introduced based on minimizing an instantaneous risk function using dynamic regularization. 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. The latter also enables FPTT to learn in an online fashion. As we show, a straightforward application of FPTT to SNNs fails, and we found it similarly failed with standard RNNs: we deduce that FPTT particularly benefits from the gating-structure inherent in LSTMs and GRUs which is lacking in standard RNNs and SRNNs.

Here, and inspired by the concept of Liquid Time Constants (LTCs) [13], we introduce a novel class of spiking neurons, the Liquid Spiking Neuron, where internal time-constants are dynamic and input-driven in a learned fashion, resulting in functionality similar to the gating operation in LSTMs. We then integrate these Liquid Spiking Neurons in SNNs that are trained with FPTT.

We demonstrate that LTC-SNNs networks 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 a number of classical benchmarks that can easily be varied in duration, like the adding task and the DVS gesture benchmark [14, 15]. We also show how LTC-SNNs trained with FPTT can be applied to large-scale convolutional SNNs, where we demonstrate novel state-of-the-art for online learning in recurrent SNNs on several 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 state-of-the-art performance as obtained with offline BPTT (Fashion-MNIST, DVS-CIFAR10).

2 Related Work

The problem of training recurrent neural networks has an extensive history, including early work by Werbos [16], Elman [17] and Mozer [18]. 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 [16]. 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) [9] and mixes of both RTRL and BPTT approaches [19]; they however exhibit prohibitive time and memory complexity [11]. See Table 1 for 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, 20], so-called "surrogate gradients " [6]. Various SNNs trained with such surrogate gradients and BPTT now achieve competitive performance compared to classical RNNs [1, 15, 21]. While effective, the application of BPTT in SNNs has several drawbacks: in particular, BPTT accumulates the approximation error of surrogate gradients along time. 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 [10] achieve linear time complexity and have proven effective for many small scale benchmark problems and also large scale networks like cortical microcircuits [22]. Online Spatio-Temporal Learning (OSTL) [11] 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. Still, in terms of trained accuracy, none of these approximations have been shown to outperform standard BPTT.

FPTT. Forward Propagation through Time (FPTT) [8] updates the network parameters by optimising the instantaneous risk function ℓ_t^{dyn}, which includes the ordinary objective $\mathcal{L}_t$ and also a dynamic regularisation penalty $\mathcal{R}_t$ based on

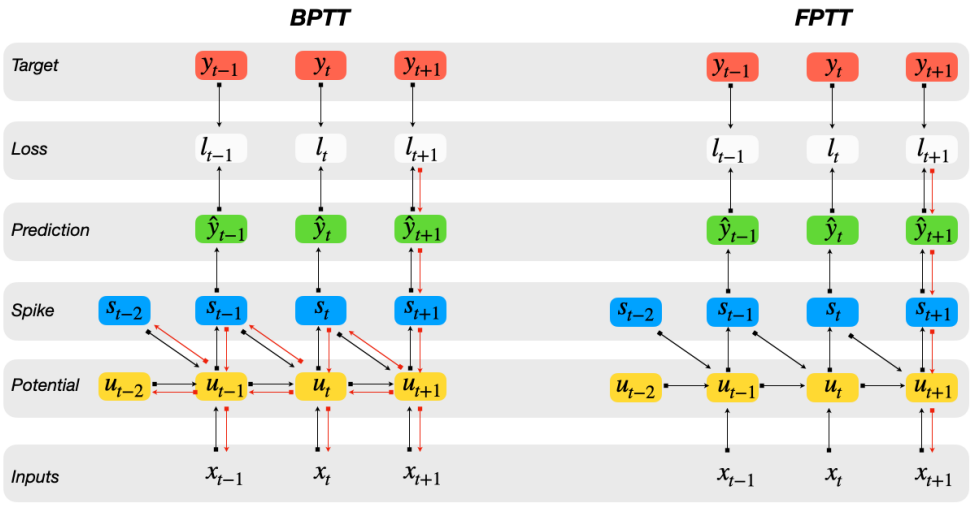


Figure 1: Roll-out of the computational graph of a spiking neuron as used for BPTT (left) and that of FPTT (right).

previously observed losses $\ell_t^{dyn} = \mathcal{L}_t + \mathcal{R}_t$ (see Appendix A for details). By adding this dynamically time-evolving regularizer, FPTT optimizes RNNs similar to feedforward networks, as shown in the computational diagram of Fig.1. FPTT thus eliminates the dependence of the gradient calculation on the sum of products of partial gradients along the time dimension in BPTT.

In detail, first, the empirical objective $\mathcal{L}(y_t, \hat{y}_t)$ is the same as that of BPTT, representing the gap between target values y_t and real time predictions $\hat{y}_t$). Second, and most important, the novel dynamic regularization part is controlled by the "running average" of all the weights seen so far. The update schema of this regularizer is as follows:

$$\begin{aligned} \mathcal{R}(\bar{\Phi}_t) &= \frac{\alpha}{2} \parallel \Phi - \bar{\Phi}_t - \frac{1}{2\alpha}\nabla l_{t-1}(\Phi_t) \parallel \\ \Phi_{t+1} &= \Phi_t - \eta \nabla_\Phi l(\Phi)|_{\Phi=\Phi_t} \\ \bar{\Phi}_{t+1} &= \frac{1}{2}(\bar{\Phi}_t + \Phi_{t+1}) - \frac{1}{2\alpha}\nabla l_t(\Phi_{t+1}) \end{aligned} \tag{1}$$

In FPTT, a state vector $\bar{\Phi}_t$ is introduced which summarises past losses: the first update is a normal update of parameters Φ_t based on gradient optimization with fixed $\bar{\Phi}_t$; after the update, we optimize $\bar{\Phi}_t$ with fixed Φ_t. This approach allows 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 modelling 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, Kag & Saligrama [8] introduced a divergence term 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.

3 Training networks of spiking neurons

To apply FPTT to SNNs, we first define the spiking neuron model and explain how BPTT is 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:

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

where τ is the set of internal time constants and Φ is the set of associated parameters like synaptic weights. The membrane potential evolves along time based on previous neuronal states (e.g. potential u_{t-1} and spike-state s_{t-1}) and current inputs x_t.

(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 it emits a spike ($s_t = 1$), the membrane potential will reset to resting potential u_r. In all experiments, we set $u_r = 0$:

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

For optimal performance, the various time constants in the spiking neurons can be learned to match the temporal dynamics of the task [15, 21].

BPTT for SNNs. As outlined in Algorithm 1, 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

$$\frac{\partial \ell_{t+1}}{\partial \Phi} = \frac{\partial l_{t+1}}{\partial \hat{y}_{t+1}} \frac{\partial \hat{y}_{t+1}}{\partial s_{t+1}} \frac{\partial s_{t+1}}{\partial u_{t+1}} \sum_{j=1}^{t+1} \left(\prod_{m=j}^{t+1} \frac{\partial u_m}{\partial u_{m-1}} \right) \frac{\partial s_{m-1}}{\partial \Phi} \tag{6}$$

where the partial derivative of spike $\frac{\partial s_t}{\partial u_t}$ 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 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.

FPTT for SNN FPTT can be used for training SNNs as described in Algorithm 2: we optimize the network by minimizing the instantaneous loss with the dynamic regularizer $\ell_t^{dyn} = \mathcal{L}(y_t, \hat{y}_t) + \mathcal{R}(\bar{\Phi}_t)$. For FPTT, the update function Equation (6) then becomes:

$$\frac{\partial \ell_{t+1}^{dyn}}{\partial \Phi} = \frac{\partial l_{t+1}}{\partial \hat{y}_{t+1}} \frac{\partial \hat{y}_{t+1}}{\partial s_{t+1}} \frac{\partial s_{t+1}}{\partial u_{t+1}} \frac{\partial u_{t+1}}{\partial \Phi} \tag{7}$$

Compared to Equation (6), Equation (7) has no dependence on a chain of past states, and can thus be computed in an online manner. Theoretically, FPTT provides a more robust and efficient gradient approximation for recurrent neural networks to avoid gradient vanishing or explosion. For SNNs, FPTT simplifies the complex gradient computation path in BPTT, potentially weakening the effect of surrogate gradients and providing a better gradient approximation for the network.

4 The Liquid Spiking Neuron

We here introduce the Liquid Spiking Neuron (LSN) model, which, as we will show, enables the application of FPTT to 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 [13], we introduce a spiking neuron model where some of the time-constants are learned functions of the inputs and hidden states of the network, as illustrated in Fig. 2.

Mathematically, we describe a Liquid Time Constant as:

$$\tau^{-1} = \sigma(x_t, u_{t-1} | W_\tau), \tag{8}$$

where, to ensure smooth changes when learning, we use a sigmoid function to scale the inverse of the time-constant to a range of 0 to 1. In detail, the liquid time-constants in standard adaptive spiking neurons [1, 10] are either calculated

Algorithm 1 Training SNN with BPTT

Require: $B = \{x_t, y_t\}_{t=0}^{T}$, # Epochs E
Require: Optimizer and learning rate η
Initialize Weight W, v,
for each $e \in E$ **do**
  Initialize Neuron states u_t, s_t
  Randomly Shuffle B
  for each $t \in T$ **do**
    Update: $s_{h,t}, u_{h,t} = \hat{f}_s(x_{t-1}, [u_{h,t-1}, s_{h,t-1}] \| W)$
    Predict: $\hat{y}_t = \hat{f}_s(s_{h,t}, [u_{o,t}, s_{o,t}] \| v)$
  end for
  $Loss$: $\ell(W) = \sum_{t=1}^{T} \ell(y_t, \hat{y}_t)$
  $Update$: $W = W - \eta \nabla_W \ell(W)|_W$
end for

Algorithm 2 Training SNN with FPTT

Require: $B = \{x_t, y_t\}_{t=0}^{T}$, # Epochs E
Require: Optimizer and learning rate η
Initialize Weight W, and $\bar{W} = W$
for each $e \in E$ **do**
  Initialize Neuron states u_t, s_t
  Randomly Shuffle B
  for each $t \in T$ **do**
    Update: $s_{h,t}, u_{h,t} = \hat{f}_s(x_{t-1}, [u_{h,t-1}, s_{h,t-1}] \| W)$
    Predict: $\hat{y}_t = \hat{f}_s(s_{h,t}, [u_{o,t}, s_{o,t}] \| v)$
    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 $\bar{W}$: $\bar{W}_{t+1} = \frac{1}{2}(\bar{W}_t + W_{t+1}) - \frac{1}{2\alpha} \nabla \ell_t(W_{t+1})$
  end for
end for

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 Spiking Neuron defined by the following equations:

$$\begin{aligned}
\tau_{adp}\, update :& \rho = \tau_{adp}^{-1} = \sigma([x_t, b_{t-1}] \parallel W_{\tau_{adp}}) \\
\tau_m\, update :& \tau_m^{-1} = \sigma([x_t, u_{t-1}] \parallel W_{\tau_m}) \\
\theta_t\, update :& b_t = \rho b_{t-1} + (1-\rho) s_{t-1} \\
& \theta_t = 0.1 + 1.8 b_t \\
u_t\, update :& du = (-u_{t-1} + x_t)/\tau_m \\
& u_t = u_{t-1} + du \\
spike\, s_t :& s_t = f_s(u_t, \theta) \\
resting :& u_t = u_t(1 - s_t) + u_{rest} s_t,
\end{aligned} \tag{9}$$

where the neuron uses an adaptive threshold θ_t as in the Adaptive Spiking Neurons [10], and τ_m and τ_m are computed as liquid time-constants.

5 Experiments

Datasets. We demonstrate the effectiveness of the FPTT algorithm with LTC-SNNs on a number of classical benchmarks either to compare to [8] (the Add-task), or to compare to established SNN benchmarks (the DVS Gesture and DVS-CIFAR10 classification tasks, and the Sequential, Sequential-Permuted, rate-based and Fashion MNIST classification tasks).

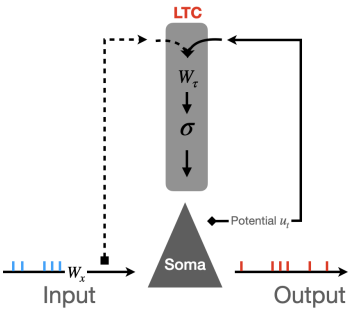


Figure 2: Circuit of Liquid time-constant spiking neuron

The **Add-Task** [23] 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 trained networks consist of 128 recurrently connected neurons of respective types LTC-SNN (LTC-SRNN), LSTM, or Adaptive Spiking Neuron [10] (ASRNN), and a dense output layer with only 1 neuron.

The **IBM DVS Gesture** dataset [14] consists of 11 kinds of hand and arm movements of 29 individuals under three different lighting condition captured using a DVS128 camera. Each frame is a 128-by-128 size image with 2 channels. **DVS-CIFAR10** is a widely used neuromorphic vision dataset where the event stream obtained by displaying the moving images of the CIFAR-10 dataset [24]. As in [15], for both DVS-Gesture and DVS-CIFAR10, we cluster the event flow into frames. Since sequence length depends on sampling frequency, we sampled such as to yield various sequence lengths from 20 to 500 frames.

In the DVS-Gesture dataset, we apply either a shallow spiking recurrent network (SRNN) or a deep spiking convolutional network (SCNN) to test the training effectiveness of FPTT on longer sequences as well as larger and deeper networks. 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. Then, the 2D image at each channel is 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 fused information is then fed to a recurrently connected layer with 512 hidden neurons. Finally, a leaky integrator is applied to generate predictions: [1024,1024]-[512D,512D]-512R-11I.

To achieve high performance on the DVS Gestureand DVS-CIFAR10 datasets, we follow [15] 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. The network was optimized through Adamax [25] with a batch size of 16 and initial learning rate of 1e-3.

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 more difficult than S-MNIST because it lacks temporally correlated patterns. 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 [25] 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.

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 [26]. As in [11], 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.

We also tested FPTT-trained LTC-SCNNs on the traditional static **MNIST** and **Fashion-MNIST** datasets for comparison with other models trained offline. Pixel values are directly injected as current into the first spiking layer of the network, repeated 20 times to mimic a constant input stream. We apply an SCNN with 3 convolutional layers, 1 Dense layer and 1 leaky Integrator output layer: ConvK3C32S1P1-MPK2S2-ConvK3C128S1P1-MPK2S2-ConvK3C256S1P1-

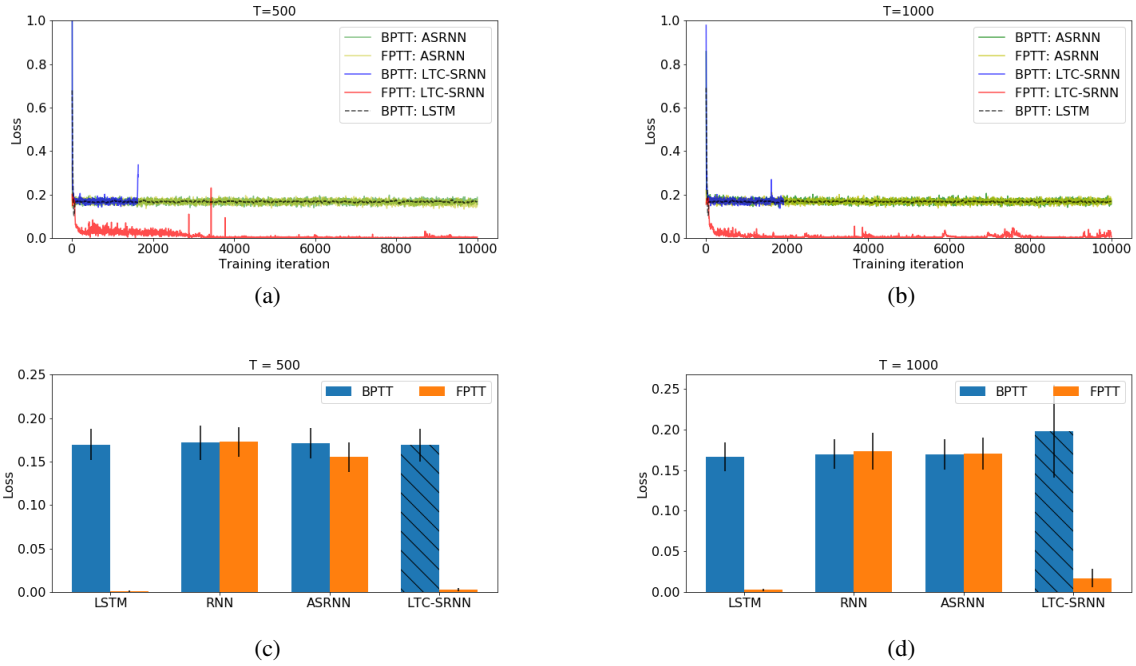


Figure 3: Add Task. **The top row** plots the loss curve for the Add Task for different sequence lengths (a) $T = 500$ and (b) $T = 1000$. We take as baseline performance a non-spiking LSTM with the same number of neural units trained with BPTT. **The second row** plots the loss of the last 100 training iterations averaged over 5 runs for sequence lengths (c) $T = 500$ and (d) $T = 1000$. The loss of LTC-SRNN becomes NaN after 2000 iterations when training with BPTT. We then report the loss right before divergence, indicated by a hatched texture in the respective bars.

MPK2S2-512D-10I. The network was optimized by Adamax with a batch size of 64 and an initial learning rate of 1e-3.

5.1 FPTT-SNN requires Liquid Spiking Neurons

To illustrate the effectiveness of the FPTT-LTC-SNN framework and the need for liquid time-constant spiking neurons when applying FPTT learning, we apply both BPTT and FPTT to the Add Task. This has been done using various networks, including non-spiking LSTMs as a baseline as in [8], adaptive SRNNs (ASRNNs) as in [1], and LTC-SRNNs.

Example loss-curves are shown in Fig. 3(a,b), for adding sequences of length 500 and 1000. As reported in [8], standard LSTMs trained with BPTT do not converge, while they do when applying FPTT training. For SNNs, we find that ASRNNs as in [1] trained with either FPTT or BPTT do not converge; for LTC-SNNs trained with BPTT, we find that the loss initially decreases to values similar to that of standard SRNNs, but then learning diverges due to exploding gradients. Finally, LTC-SNNs trained with FPTT successfully minimize the loss similar to the FPTT-LSTM networks. The final losses averaged over 5 networks are shown in Fig 3(c,d).

5.2 FPTT allows for longer sequence training

We next study the ability of FPTT-style training of LTC-SNNs to learn increasingly long sequences. With the DVS Gesture dataset, we systematically investigate the performance of shallow SRNN models on increasingly many frames of high frequency sampled signals: we converte the entire event stream into sequences of differing lengths, ranging from 20 to 500 frames. The longer sequence lengths pose a serious challenge to a network's ability to memorize relevant information at different time scales. Furthermore, it also places increasingly high demands on memory and training time for training via BPTT.

We examine the performance of various network architectures, training methods and loss functions. In particular, we trained a set of networks with an identical number of neural units: LSTM networks with BPTT, with and without the auxiliary loss, and similarly trained ASRNNs and recurrent LTC-SNNs (LTC-SRNNs). This we compared to

LTC-SRNNs trained with FPTT with the auxiliary loss. The results for different sampling frequencies are listed in Table 2.

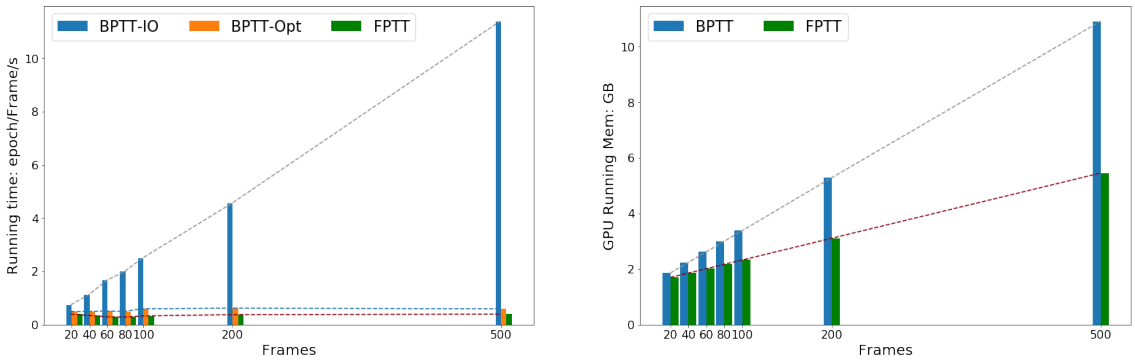
Table 2: **Performance comparison between BPTT and FPTT on the DVS gesture dataset.** Each number in the table is the average of three runs. All networks have equal number of neural units.(*): training diverged; reported accuracy is best accuracy before divergence.

Frames	BPTT					FPTT
	LSTM+Aux	LSTM	LTC-SRNN+Aux	LTC-SRNN	ASRNN+Aux	LTC-SNN
20	86.69±0.43	82.29±2.46	83.42±1.35	84.37±2.27	79.16±1.98	**88.31**±0.59
40	88.77±1.71	84.95±0.71	85.96±1.16	84.37±1.24	80.78±1.40	**90.39**±0.71
60	87.61±0.86	85.15±0.75	85.62±1.18	83.91±0.71	80.55±0.49	**90.74**±0.16
80	87.97±0.14	84.83±1.42	85.30±0.71	80.44±3.6	76.04±0.85	**91.31**±0.98
100	88.89±0.49	83.79±0.71	83.21±0.43	78.70±0.91	74.3±0.84	**91.89**±0.16
200	85.76±0.49	81.87±2.58	51.39±6.0	43.98±2.35	64.87±0.78	**90.16**±1.43
500	82.52±1.82	78.81±1.5	38.89±3.22	36.46±1.5	48.32±2.0(*)	**90.64**±1.56

Table 3: **Firing rate (fr) and training-time-per-frame comparison between BPTT and FPTT on the DVS gesture dataset.** Each number in the table is the average of three runs.

Frames	Fr			LTC-SNN Time (s)	
	BPTT		FPTT	FPTT	BPTT
	SRNN+Aux	LTC-SRNN	SRNN		
20	0.242	0.255	0.206	0.4	0.75
40	0.202	0.220	0.163	0.36	1.13
60	0.173	0.198	0.138	0.32	1.67
80	0.142	0.171	0.126	0.29	2.00
100	0.120	0.144	0.119	0.33	2.50
200	0.089	0.088	0.091	0.38	4.56
500	0.040	0.076	0.071	0.4	11.4

From the Table 2, we first observe that the LTC-SRNN trained using FPTT achieved the best performance in all cases, also outperforming standard (BPTT-trained) LSTMs. The FPTT-trained LTC-SRNN moreover exhibits essentially constant performance over the whole range of sequence lengths. In contrast, the accuracy of both LTC-SRNNs and ASRNNs quickly deteriorate as sequence length increases, from 85.5% for 60 frames to 38.9% for 500 frames for the best performing LTC-SRNN. For the baseline standard LSTM this effect is also there, albeit more moderate (decreasing 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.



(a) Time Efficiency (b) Memory Efficiency

Figure 4: (a) Time Efficiency per frame: training speed of the network using BPTT and FPTT on the DVS-Gesture dataset with different sampling frequencies, where for BPTT both standard (BPTT-Opt) and FPTT-like IO (BPTT-IO) is plotted. (b) Memory Efficiency: Memory cost required for network training of the DVS-Gesture dataset at different sampling frequencies. We set the batch size to 64.

Table 4: Performance of deep SRNNs/SCNNs on various tasks.

Task	**Online**			**Offline** [BPTT]		this work
	Algorithm	Network	test Acc	Network	Test Acc	LTC-SRNN
S-MNIST	e-prop[27]	LSNN	94.32%	ASRNN[1]	98.7%	97.37%
PS-MNIST	-	-	-	ASRNN[1]	94.3%	94.77%
R-MNIST	OSTL[11]	SNU[28]	95.54%	SNU[28]	97.72%	98.63%
MNIST	-	-	-	LISNN[29]	99.5%	99.62%
				PLIF[15]	99.72%	
Fashion-MNIST	-	-	-	LISNN[29]	92.07%	93.58%
				PLIF[15]	94.38%	
DVS-Gesture	DECOLLE[30]	SNN	95.54%	PLIF[15]	97.57%	97.22%
DVS-CIFAR10	-	-	-	SNN[31]	60.5%	72.3%
				PLIF[15]	74.8%	

We also noted the sparsity (average firing rate) and training time per frame in Table 3. For sparsity, we find no meaningful differences between BPTT-SNNs and the FPTT-LTC-SNN; in terms of training time, we find that, when using identical sample-memory-retrieval IO, FPTT-trained LTC-SRNNs train increasingly faster than the BPTT-SNNs; when using the Pytorch optimized BPTT training routine, FPTT was faster consistently by an approximately constant factor (Fig. 4a).

For on-GPU memory consumption, FPTT-LTC-SNNs require increasingly less memory as sequence length increases (Fig. 4b). Additionally, for the DVS-Gesture dataset trained on a sequence length of 500 frames, for a batch size of 64 the LSTM-BPTT implementation uses 10.8GB of GPU-memory, the LTC-SNN-BPTT version uses 13.2GB, while the LTC-SNN-FPTT uses 5.6GB. Increasing the batch size to 128 then causes the BPTT versions to no longer fit in our GPU memory (24GB) while the FPTT-LTC-SNN uses 9.6GB.

For both training time and memory usage, we note that the FPTT implementation is unoptimized in the used version of PyTorch, where for instance the memory allocated to historical hidden states is not de-allocated for FPTT, resulting in unnecessary large memory use, and low-level optimized FPTT implementations should further reduce memory to near constant.

5.3 FPTT with LTC Spiking Neurons improves over Online BPTT

We further compare large and deep LTC-SCNNs trained with FPTT on standard benchmarks. As shown in Table 4, we find that LTC-SCNNs trained with FPTT consistently outperform SNNs trained with online BPTT approximations like OSTL and e-Prop. Compared to offline BPTT approaches, the online FPTT-trained LTC-SRNNs achieve new SoTa for SNNs (PS-MNIST, R-MNIST) or achieve close to similar performance (S-MNIST, DVS-Gesture, DVS-Cifar10).

For these large networks, we also find that the memory requirements for FPTT is substantially lower than for BPTT training by a factor of 4 to 5 (Table 5), and training time is substantially reduced, typically by a factor of 3 to 4 (Table 6).

Table 5: Memory efficiency. (*): Models using the same batch size cannot be trained on a single GPU, the reported number is obtained using a halved batch size.

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

Table 6: Total training time.

	S-MNIST	rate-MNIST	MNIST	DVS gesture
BPTT-IO	-	18min	25min	-
BPTT-opt	40min	192s	362s	108s
FPTT	737s	204s	384s	112s

6 Discussion

We showed how a novel training approach, FPTT, can be successfully applied to long sequence learning with recurrent SNNs using novel Liquid Spiking Neurons. Compared to BPTT, FPTT is compatible with online training, has constant memory requirements, trains substantially faster even without optimizations in the software framework and can learn longer sequences with a constant network architecture. In terms of accuracy, FPTT substantially outperforms online approximations to BPTT like OSTL and eProp. Additionally, when training large deep SCNNs with FPTT, excellent performance is achieved approaching or exceeding not-online BPTT-based solutions, including a first demonstration of online learning of tasks like DVS-CIFAR10.

To achieve these results, we introduced Liquid Time-Constant Spiking Neurons (LTC-SN), where the time-constants in the neuron are computed as a learned dynamic function of the current state and input. The LTC-SN is inspired by the functioning of pyramidal neurons in brains, where the apical tuft is coupled to somatic processing [32, 33]. Pyramidal neurons are known to have complex non-linear interactions between different morphological parts far exceeding the simple dynamics of LIF-style neurons [34], where the apical tuft may calculate a modulating term acting on the computation in the soma [35], 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 [36, 37] and are central to biologically plausible theories of learning working memory [38].

In terms of improvements in training time and memory use, the benefits of FPTT versus BPTT were substantial, however they were less than theoretically should be the case. We believe that here, the principal cause are the low-level optimizations in frameworks like Pytorch that are implemented for BPTT but not (yet) for FPTT: when we rearranged BPTT to have a similar main memory access pattern as our FPTT implementation, BPTT training time showed a quadratic increase, as expected.

The use of FPTT may also hold 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 [39], FPTT-training of LTC-SNNs can also likely be implemented fully locally and online on neuromorphic hardware.

Together, we believe this work suggests that FPTT may be an excellent training paradigm for SNNs, particularly for LTC-SNNs, which introduce the idea of local online updates necessary in biologically constrained neural information processing systems.

Acknowledgement 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").

References

[1] Bojian Yin, Federico Corradi, and Sander M. Bohté. "Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks". In: *Nat Mach Intell* 3 (2021), pp. 905–913.

[2] Jan Stuijt et al. "μBrain: An Event-Driven and Fully Synthesizable Architecture for Spiking Neural Networks". In: *Frontiers in neuroscience* 15 (2021), p. 538.

[3] Nicolas Perez-Nieves et al. "Neural heterogeneity promotes robust learning". en. In: *Nat. Commun.* 12.1 (Oct. 2021), p. 5791.

[4] Joram Keijser and Henning Sprekeler. "Interneuron diversity is required for compartment-specific feedback inhibition". In: *bioRxiv* (2020).

[5] Sander M Bohte. "Error-backpropagation in networks of fractionally predictive spiking neurons". In: *International Conference on Artificial Neural Networks*. Springer. 2011, pp. 60–68.

[6] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks". In: *IEEE Signal Processing Magazine* 36.6 (2019), pp. 51–63.

[7] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems: NeurIPS* 32 (2019), pp. 8026–8037.

[8] Anil Kag and Venkatesh Saligrama. "Training Recurrent Neural Networks via Forward Propagation Through Time". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 5189–5200.

[9] Ronald J Williams and David Zipser. "A learning algorithm for continually running fully recurrent neural networks". In: *Neural computation* 1.2 (1989), pp. 270–280.

[10] Guillaume Bellec et al. "A solution to the learning dilemma for recurrent networks of spiking neurons". In: *Nature communications* 11.1 (2020), pp. 1–15.

[11] Thomas Bohnstingl et al. "Online spatio-temporal learning in deep neural networks". In: *arXiv preprint arXiv:2007.12723* (2020).

[12] Yuming He 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)*. 2021, pp. 1–3. DOI: 10.1109/A-SSCC53895.2021.9634787.

[13] Ramin Hasani et al. "Liquid Time-constant Networks". In: (June 2020). arXiv: 2006.04439 [cs.LG].

[14] Arnon Amir et al. "A low power, fully event-based gesture recognition system". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 7243–7252.

[15] Wei Fang 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*. 2021, pp. 2661–2671.

[16] Paul J Werbos. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[17] Jeffrey L Elman. "Finding structure in time". In: *Cognitive science* 14.2 (1990), pp. 179–211.

[18] Michael C Mozer. "Neural net architectures for temporal sequence processing". In: *Santa Fe Institute Studies in the Sciences of Complexity-Proceedings Volume-*. Vol. 15. ADDISON-WESLEY PUBLISHING CO. 1993, pp. 243–243.

[19] James M Murray. "Local online learning in recurrent networks with random feedback". In: *ELife* 8 (2019), e43299.

[20] Sander M Bohte, Joost N Kok, and Han La Poutre. "Error-backpropagation in temporally encoded networks of spiking neurons". In: *Neurocomputing* 48.1-4 (2002), pp. 17–37.

[21] Bojian Yin, Federico Corradi, and Sander M Bohté. "Effective and efficient computation with multiple-timescale spiking recurrent neural networks". In: *International Conference on Neuromorphic Systems 2020*. 2020, pp. 1–8.

[22] Franz Scherr and Wolfgang Maass. "Analysis of the computational strategy of a detailed laminar cortical microcircuit model for solving the image-change-detection task". In: *bioRxiv* (2021). DOI: 10.1101/2021.11.17.469025. eprint: https://www.biorxiv.org/content/early/2021/11/19/2021.11.17.469025.full.pdf.

[23] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[24] Hongmin Li et al. "Cifar10-dvs: an event-stream dataset for object classification". In: *Frontiers in neuroscience* 11 (2017), p. 309.

[25] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[26] Wulfram Gerstner et al. "Neural codes: firing rates and beyond". In: *Proceedings of the National Academy of Sciences* 94.24 (1997), pp. 12740–12741.

[27] Guillaume Emmanuel Fernand Bellec et al. "Long short-term memory and learning-to-learn in networks of spiking neurons". In: *Advances in Neural Information Processing Systems: NeurIPS*. 2018.

[28] Stanisław Woźniak et al. "Deep learning incorporating biologically inspired neural dynamics and in-memory computing". In: *Nature Machine Intelligence* 2.6 (2020), pp. 325–336.

[29] Xiang Cheng et al. "LISNN: Improving Spiking Neural Networks with Lateral Interactions for Robust Object Recognition." In: *IJCAI*. 2020, pp. 1519–1525.

[30] Jacques Kaiser, Hesham Mostafa, and Emre Neftci. "Synaptic plasticity dynamics for deep continuous local learning (DECOLLE)". In: *Frontiers in Neuroscience* 14 (2020), p. 424.

[31] Yujie Wu et al. "Direct training for spiking neural networks: Faster, larger, better". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 1311–1318.

[32] João Sacramento et al. "Dendritic cortical microcircuits approximate the backpropagation algorithm". In: *Advances in Neural Information Processing Systems: NeurIPS* 31 (2018), pp. 8721–8732.

[33] Albert Gidon et al. "Dendritic action potentials and computation in human layer 2/3 cortical neurons". en. In: *Science* 367.6473 (Jan. 2020), pp. 83–87.

[34] David Beniaguev, Idan Segev, and Michael London. "Single cortical neurons as deep artificial neural networks". In: *Neuron* 109.17 (2021), pp. 2727–2739.

[35] Matthew E Larkum, Walter Senn, and Hans-R Lüscher. "Top-down dendritic input increases the gain of layer 5 pyramidal neurons". en. In: *Cereb. Cortex* 14.10 (Oct. 2004), pp. 1059–1070.

[36] Uwe Frey and Richard GM Morris. "Synaptic tagging and long-term potentiation". In: *Nature* 385.6616 (1997), pp. 533–536.

[37] Diego Moncada et al. "Identification of transmitter systems and learning tag molecules involved in behavioral tagging during memory formation". In: *Proceedings of the National Academy of Sciences* 108.31 (2011), pp. 12931–12936.

[38] Jaldert O Rombouts, Sander M Bohte, and Pieter R Roelfsema. "How attention can create synaptic tags for the learning of working memories in sequential tasks". In: *PLoS computational biology* 11.3 (2015), e1004060.

[39] Isabella Pozzi, Sander Bohte, and Pieter Roelfsema. "Attention-Gated Brain Propagation: How the brain can implement reward-based error backpropagation". In: *Advances in Neural Information Processing Systems: NeurIPS* 33 (2020).

Appendix A: FPTT theory

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

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

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

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

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

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

and, for one step optimization:

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

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

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

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

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

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

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

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

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

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

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

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

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