Recurrent Spiking Neural Networks

A quick overview of the e-prop online learning framework

Florent Pollet
ENS Paris-Saclay
florent.pollet@ens-paris-saclay.fr

March 24, 2024

Contents

List of Figures

List of Tables

1 Introduction

Spiking neural networks (SNN) could represent the next major revolution in Artificial Intelligence (AI). In the last decade, AI advances were made possible thanks to the latest breakthroughs of Deep Learning [21]. The increase of parallelization and GPU computing allowed the training and the use of Deep Artificial Neural Networks (ANN) for a wide-range of applications. However, nowadays, such approaches show their limits: for instance, a training of a Large Language Model (LLM) like GPT-3.5 can cost several millions of dollars[1] and it is estimated to consume every day more than 1 GWh[2].

Therefore, there is currently extensive research to increase performance but primarily energy-efficiency of the existing AI systems [8]. In this regard, the idea of mimicking more closely brain mechanisms with neural networks which include biologically-inspired neuron models has emerged at several levels [24]: regarding hardware, switching from Von Neumann architectures in which memory is separated from processing (Von Neumann bottleneck) to novel ones is being considered; for software, new neural network frameworks are being designed like Spiking Neural Networks. They bring a new paradigm which includes temporality and sparsity, unlike ANNs.

Spiking Neural Networks are part of the larger Neuromorphic Computing/Brain Inspired Computing community [22], which should become a big market in the coming years, and famous tech companies like Intel are already investing in these fields. As of today, unlike "traditional" deep learning, hardware and software are less mature, much less standardized and there is still heavy exploration of different methods and applications [26]. One of the main challenges consist in the training of these neural networks [28]. Indeed, due to the non-differentiability of the activation functions, ANN techniques cannot be easily reproduced.

In this regard, the studied paper [1] suggests a new online learning method called *e-prop*, for eligibility traces propagation, which is both efficient and biologically-plausible for SNN training, and they compare it with the offline state-of-the-art BackPropagation-Through-Time (BPTT) method, which has shown to give the best results despite its computational complexity and its artificial paradigm. In the paper, they focus on one simple architecture, namely a single layer recurrent spiking neural network (RSNN).

In this work, I start by presenting the general spiking neural network framework considered in the paper [1] in section 2, before quickly presenting the e-prop learning method in section 3, without the additional reward e-prop technique for Reinforcement Learning (RL). Then, I implement e-prop with the recent *jax* Python library [5] and finally I perform some supervised learning experiments on another dataset in section 4. The code is available on Github `https://github.com/florian6973/btt-spyx`. Tests were made using a Nvidia P100 (Kaggle) and a Nvidia V100S (OVH Cloud), resorting to GPUs to speed up computation times.

[1] https://www.cnbc.com/2023/03/13/chatgpt-and-generative-ai-are-booming-but-at-a-very-expensive-price.html
[2] https://medium.com/@zodhyatech/how-much-energy-does-chatgpt-consume-4cba1a7aef85

2 Recurrent Spiking Neural Networks (RSNN)

Spiking neural networks represent a vibrant field in constant evolution, as some papers like [10] highlights. Therefore, it is important to clearly define the different models and architectures that are being used in the studied paper to avoid any potential confusion.

2.1 Neuron models

Several neuron models can be considered for spiking neural networks. Choosing one type instead of the other is often a trade-off between bio-plausibility and computational efficiency. If the neuron models have been known since the early nineteenth century and are often continuous, they are being discretized for SNNs.

Leaky Integrate-and-Fire (LIF) The LIF neuron is a very simple model for spike neural networks. It has been originally described with the following ordinary differential equation 1, and modeled as an electronic circuit, with U the membrane potential, I the input current, and $\tau = RC$ the membrane time constant of the neuron, U_{rest} the resting potential:

$$\tau \frac{dU(t)}{dt} = -[U(t) - U_{rest}] + I_{in}(t)R \tag{1}$$

For SNN, we use an equivalent discretized version. Each neuron has an internal/hidden state v^t at time t, an observable state $z^t \in \{0, 1\}$, has inputs $(x_i^t)_i$ such that $x_i^t = 1$ if the neuron i fired at time t, 0 otherwise, and w^{in} the input synapse weights. A neuron fires at t if $v^t > v_{th}$, with v_{th} the threshold potential. With δt the discrete time step size (1 ms for instance), $\alpha = \exp(-\delta t/\tau)$ the decay time constant, and H the Heaviside step function, we can write:

$$\begin{cases} v^{t+1} &= \alpha v^t + \sum_i w_i^{in} x_i^{t+1} - z^t v_{th} \\ z^{t+1} &= H(v^{t+1} - v_{th}) \end{cases} \tag{2}$$

The $-z^t v_{th}$ term shows that the potential is decreased after the emission of a spike. In addition, to add refractoriness to this model, z_t can be fixed to 0 for a few times after (usually 3 for 3 ms, to match biology).

In the studied paper [1], they consider a slightly modified version of the classical LIF model [10] compared to most neuromorphic software: they add "recurrent" connections between the neurons of the same layer, indexed by j, so the potential can be rewritten as, with W^{rec} the associated recurrent weights:

$$v_j^{t+1} = \alpha v_j^t + \sum_{i \neq j} W_{j,i}^{rec} z_i^t + \sum_i W_{j,i}^{in} x_i^{t+1} - z_j^t v_{th} \tag{3}$$

As a result, the main parameters to consider for this model are the decay factor α (accounting for the leaky behavior), the threshold potential v_{th} and the refractory period. We do not follow the Lapicque parametrization (1907) which is less convenient and common.

Adaptive LIF (ALIF) The LIF model is known to limit the performance of spiking neural networks built using it [3]. Therefore, modifying the LIF model to add spike-frequency adaptation (SFA), which is self-inhibitory mechanism of biological neurons, can improve the expressivity of the neural network. Introduced by [3], the neuron has now an additional hidden state variable a^t, which represents the variable component of its firing rate, and the equations can be rewritten, with β and ρ:

$$\begin{cases} v^{t+1} &= \alpha v^t + \sum_i w_i^{in} x_i^{t+1} - z^t v_{th} \\ a^{t+1} &= \rho a^t + z^t \\ z^{t+1} &= H(v^{t+1} - v_{th} - \beta a^t) \end{cases} \tag{4}$$

As for LIF, [1] considers recurrent ALIF neurons. Moreover, there are two new parameters β and ρ to tune in this model.

Note that [11] suggests a simpler alternative to overcome the LIF limitations: increasing the leakage time constant τ of the LIF neurons allows to show similar learning performance as networks with ALIF neurons. Therefore, the additional computation complexity of the ALIF model is not needed when proper constant tuning is done.

A short overview and comparison of these two models is shown in Figure 1, with a neuron having a single input. We can clearly see that, for a same input spike train, SFA makes the ALIF neuron response differentiate from the LIF neuronover time.

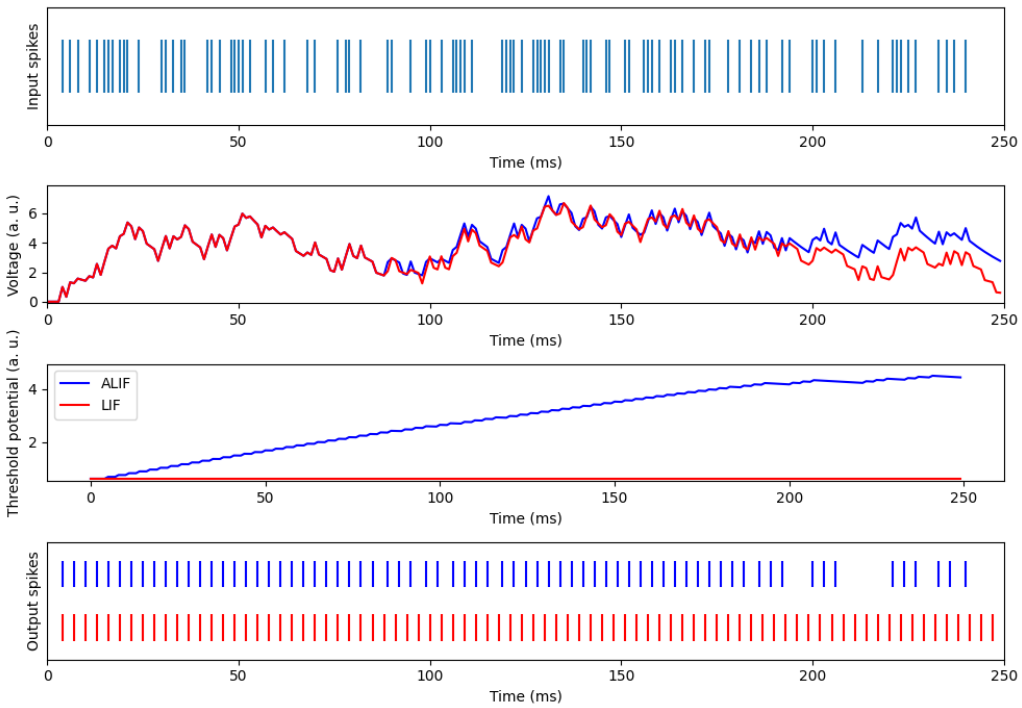

Figure 1: Comparison of the LIF and ALIF neuron models on a short time scale (without learning)

Izhikevich Even though not mentioned in the original paper [1] but in later work [30], the Izhikevich model could be interesting because it properly models the refractory period of a neuron. However, [30] has shown that this type of neuron is not adapted to e-prop, because the eligibility vectors (defined in the section 3) tend to diverge. As a result, we will not consider them in this report.

Hodgkin-Huxley The best theoretical model to explain biological observations is considered to be the one from Hodgin-Huxley, introduced in 1952 [16]. Indeed, it takes into account the potential of different ion channels. However, it has never been used in spiking neural networks due to its computational cost.

Leaky neurons This category is not standard, it is used in the paper [1] for output neurons with temporal smoothing. These leaky neurons are not spiking neurons, they are real valued but they also include a decay factor κ in their dynamics that can be computed with $\exp(-\delta_t/\tau_{out})$ by analogy to the LIF neurons. With b the bias, it can be described by the equation:

$$y^t = \kappa y^{t-1} + \sum_j w_j z_j^t + b \tag{5}$$

Note that in their code[3], they use a slightly different equation $y^t = \kappa y^{t-1} + (1-\kappa)\sum_j w_j z_j^t$ that can be easily implemented with an exponential filter.

2.2 Coding

One of the first questions which arises when dealing with spiking neural networks is how to represent data as spike trains, that is to say how to transform an image or numbers, continuous data, into a series of 0 and 1 corresponding to activations of some input neurons. Hopefully that is already what is happening in our retina for instance, so it is possible to get inspiration from the brain, as always. Decoding can follow the same logic or use classical techniques like one-hot encoding for regression or classification tasks.

Rate coding Rate coding may be the simplest coding method: it consists in converting a continuous input value to a firing rate. It has been widely used, people today opt for alternatives, since this type of coding is not the sparsest, and therefore it is not the most energy efficient. However, it has been shown to bring the best results so far.

[3] Available `https://github.com/IGITUGraz/eligibility_propagation`

Temporal coding Temporal coding is a very popular technique, also known as latency coding, which uses a fixed total number of spikes: information comes from the timing of the spike (time-to-first-spike mechanism). The main advantage of this method is the sparsity of the coding. One potential downside could be the required time window to allow enough information transfer. However, logarithmic temporal encoding has proven to be both biologically and computationally practical to overcome this issue.

Delta modulated coding Delta modulated coding is a rarer coding scheme, inspired by neurophysiology. With this method, there is firing when there is a significant change in intensity. It is used in some recent datasets, like [7], for which the data has been converted using a simulated cochlear model.

2.3 Architectures

Spiking neural networks cover a wide range of architectures, most of them being inspired by artificial neural networks. However, it is worth noticing that bringing architectures from ANN is not always adapted to this framework and it may lead to suboptimal performance [20]. Note that some hybrid architectures between ANN and SNN are also studied.

Feed-forward Therefore, classical Deep-Learning architectures such as Multi-Layer Perceptron (MLP), deep feed-forward networks, and Convolutional Neural Networks (CNN) have been transposed to SNNs [17]. However, without ANN-to-SNN conversion (see section 2.4), they tend to be less deep and they do not perform as well as their ANN counterpart.

Long short-term memory Spiking Neural Networks (LSNN) In the studied paper [1], the authors focus on Recurrent Spiking Neural Networks (RSNN), which is one of the most promising architecture to date with SNN. The network has an initial state, and then input is fed into the network producing a new output and modifying the initial state. Then, a new input is fed into the network, alongside the modified state and it produces a new output. Therefore, inference is recursively done via unfolding of the network.

It is worth noticing that the authors of [3] underline the fact that RSNN with SFA can be equivalent to LSTM in traditional Deep Learning, hence the name LSNN for the architecture. However, they study LSNN with only one fully-connected layer, which could potentially limit the expressivity of the network for complex tasks. Thus, in addition to biological claims from the Allen Institute, the considered RSNNs in [1] contain a mix of LIF and ALIF neurons.

Transformer Some recent papers like [33] have introduced spiking transformers, even though it is not very widespread yet.

2.4 Training

Finally, the last step to get a useful framework is to be able to train the spiking neural networks so they can learn. And it is precisely the core of the studied paper [**bellec2020**] to introduce a new method in this regard. However, let's start by reviewing the most common paradigms used to train SNN.

ANN-to-SNN conversion One of the easiest strategy to get a trained SNN is to train an ANN and then convert the network and its weights to an SNN [9]. It is mainly used for Deep SNN, for which learning can be really hard, and it only uses very simple neuron models like Integrate-and-Fire (IF), with often rate coding.

Spike-timing-dependent plasticity (STDP) To train a SNN from scratch, one of the first ideas is to apply biological mechanisms that have been known for learning in the brain. In this regard, STDP can be seen as a timing dependent variant of the Hebbian learning rule *Neurons who fire together, wire together; and those who fire out of sync, lose their link*, for which weights (or neural links) increase when two spikes happen in the expected causal temporal order, otherwise they decrease.

This online and local learning method is much more computationally-efficient than BPTT but it often leads to poor accuracy results. Therefore, some hybrid approaches are studied, and with a step back, it is possible to consider that [1] could be one of them. However, it is a bit that the paper does not explicitly mention the relationship between STDP and e-prop, which is done hopefully covered by [30]: e-prop shows STDP-like properties when using large synaptic delays, or ALIF models can be slightly add STDP behavior [29].

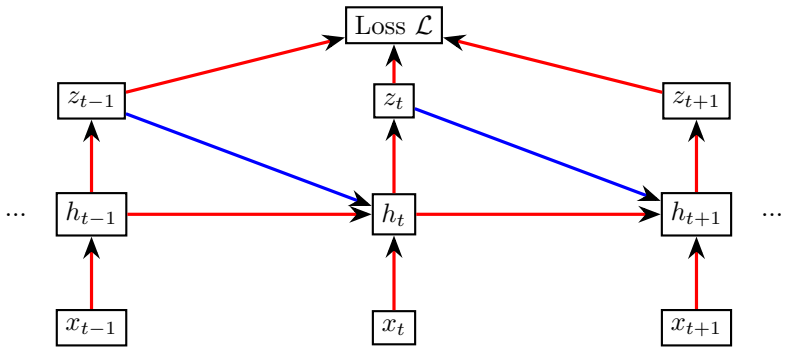

Figure 2: Example of a computational graph for a Spiking Neural Network. Inputs are written $(x_t)_t$, outputs $(z_t)_t$, and hidden states of the network $(h_t)_t$. Blue arrows are only present for Recurrent Neural Networks.

BackPropagation Through Time (BPTT) The BPTT method comes from an optimization standpoint: it is an offline gradient-based learning method to train spiking neural networks. It is much more efficient and popular than the former backpropagation using spike times [4]. The idea of BPTT was inspired by RNNs, since SNNs inherently come with a time dimension, and it consists in computing the gradient of the unrolled computational graph of the network, as shown in Figure 2. This method applies both to Feed-Forward SNN and RSNN.

In this framework, one of the main hurdles is to handle the non-differentiable activation functions (like Heaviside). In this regard, surrogate gradients are used. Different strategies can be considered, like approximating it with Gaussian functions [27] or other estimators like arctan, most of them have been validated experimentally. The only thing to remember is that the neuron must have fired to get a non-zero gradient.

In the most extreme cases, for microchips, simple Straight-Through Estimators (STE), introduced by Hinton, which consist in simply the identity function, have demonstrated reasonable performance [11]. It should be reminded that this method does not prevent vanishing or exploding gradients for RSNNs or Deep SNNs.

3 The e-prop learning framework

e-prop, also known as eligibility propagation, has been introduced in [1]. e-prop is a type of forward-mode learning algorithms, trying to approximate the standard Real-Time Recurrent Learning (RTRL) approach [34] from 1989, which is much more memory-intensive but more accurate. Its main advantages are biologically-plausible rules, good complexity and performance for an online (temporal locality) method. In [11], they also suggest a spatially local variant.

e-prop is not the only RTRL approximation: Decolle [19] or Forward Propagation Through Time (FPTT) [32] are other interesting alternatives, with which e-prop should be compared, but it is not done in the paper.

3.1 Biological considerations

The learning dilemma that the paper [1] claims to solve is that the method showing the best results, BPTT, is not biologically plausible at all, because of the need of storing the intermediate state and a backward gradient computation. Therefore, the design of a "good" online algorithm could revolutionize the neuromorphic hardware. In any case, any analogy with brain computations must always be done carefully, as highlighted in [6].

However, two neuroscience discoveries suggest new mechanisms for learning, that will be the foundations of the e-prop framework:

- Preceding activity leaves traces at the molecular level (calcium ions or CAMKII enzymes), which are called eligibility traces. They flow from left to right in Figure 2.
- Top-down signals exist in the brain, thanks to dopamine or acetylcholine, to inform the neurons of behavioral results. They are called learning signals. They flow vertically from top to bottom in Figure 2.

These two elements, eligibility traces and learning signals, lay the biological foundation for the e-prop framework.

3.2 e-prop definition

The e-prop derivation roughly consists in approximating the gradients in an online setting with eligibility traces and learning signals. To understand the following calculations better, we will use the notations and the full computational graph of Figure 2, and consider the weights (W_{ji}), each weight representing the link from the neuron i to the neuron j.

In most papers, very little attention is put to properly distinguish total and partial derivatives unfortunately, so here are the used definitions:

- $\frac{d.}{d.}$ indicates the total derivative
- $\frac{\partial.}{\partial.}$ indicates the partial derivative, which only makes sense between directly connected nodes of the computational graph
- $[\frac{d.}{d.}]_{local}$ consists in the total derivative in an altered computational graph, for which links from z_t to z_{t+1} (blue) are removed: it corresponds to the red graph in Figure 2

Please note that L_j^t is an approximation of the total derivative and it does not take into account indirect influences (from the future - unavailable for an online method). Moveover, in the following computations, derivatives can be vectors or matrices, depending on the variables.

We define the ideal learning signal for neuron j as L_j^t and the eligibility trace for synapse of indices ji as e_{ji}^t:

$$L_j^t = \frac{\partial \mathcal{L}}{\partial z_j^t} \quad (\approx \frac{d\mathcal{L}}{dz_j^t}) \qquad\qquad e_{ji}^t = \left[\frac{dz_j^t}{dW_{ji}}\right]_{local} \tag{6}$$

$$= \frac{\partial z_j^t}{\partial h_j^t} \cdot \sum_{t' \leq t} \frac{\partial h_j^t}{\partial h_j^{t-1}} ... \frac{\partial h_j^{t+1}}{\partial h_j^{t'}} \cdot \frac{\partial h_j^{t'}}{\partial W_{ji}} \tag{7}$$

From there, the main e-prop equation, derived from BPTT in [2] and [1], can be written as:

$$\frac{d\mathcal{L}}{dW_{ji}} = \sum_t \frac{d\mathcal{L}}{dz_j^t} e_{ji}^t \approx \sum_t L_j^t e_{ji}^t \tag{8}$$

3.2.1 e-prop derivations from BPTT

e-prop main equation 8 can be derived from the BPTT classical factorization of [31]:

$$\frac{d\mathcal{L}}{dW_{ji}} = \sum_{t'} \frac{d\mathcal{L}}{dh_j^{t'}} \cdot \frac{\partial h_j^{t'}}{\partial W_{ji}} \tag{9}$$

We note $(1, ..., t_m)$ the timestep sequence, we can expand the first term of the product using the chain rule, helped by the computational graph of Figure 2 and then proceed by induction:

$$\begin{cases} \frac{d\mathcal{L}}{dh_j^t} = \frac{d\mathcal{L}}{dz_j^t} \cdot \frac{\partial z_j^t}{\partial h_j^t} + \frac{d\mathcal{L}}{dh_j^{t+1}} \cdot \frac{\partial h_j^{t+1}}{\partial h_j^t} \text{ for } 1 \leq t \leq t_m \\ \frac{d\mathcal{L}}{dh_j^t} = 0 \text{ for } t > t_m \end{cases} \tag{10}$$

After a full recursion, we get a triangular sum for which we can switch the summation order:

$$\frac{d\mathcal{L}}{dW_{ji}} = \sum_{1 \leq t' \leq t_m} \left(\sum_{t' \leq t \leq t_m} \frac{d\mathcal{L}}{dz_j^t} \cdot \frac{\partial z_j^t}{\partial h_j^t} \cdot \left[\prod_{t'+1 \leq t'' \leq t} \frac{\partial h_j^{t''}}{\partial h_j^{t''-1}} \right] \right) \cdot \frac{\partial h_j^{t'}}{\partial W_{ji}} \tag{11}$$

$$= \sum_{1 \leq t \leq t_m} \frac{d\mathcal{L}}{dz_j^t} \cdot \frac{\partial z_j^t}{\partial h_j^t} \cdot \left(\sum_{1 \leq t' \leq t} \left[\prod_{t'+1 \leq t'' \leq t} \frac{\partial h_j^{t''}}{\partial h_j^{t''-1}} \right] \cdot \frac{\partial h_j^{t'}}{\partial W_{ji}} \right) \tag{12}$$

$$= \sum_t \frac{d\mathcal{L}}{dz_j^t} e_{ji}^t \tag{13}$$

To facilitate intermediate computations, we can also define the eligibility vector ϵ_{ji}^t recursively and rewrite the eligibility trace:

$$\epsilon_{ji}^t = \begin{cases} 0 \text{ for } t = 0 \\ \frac{\partial h_j^t}{\partial h_j^{t-1}} \cdot \epsilon_{ji}^{t-1} + \frac{\partial h_j^t}{\partial W_{ji}} \text{ for } 1 \leq t \leq t_m \end{cases} \quad \implies e_{ji}^t = \frac{\partial z_j^t}{\partial h_j^t} \cdot \epsilon_{ji}^t \tag{14}$$

Last but not least, as in BPTT, $\partial z_t^j / \partial h_j^t$ is approximated with a surrogate gradient.

3.2.2 Broadcast weights

The section 3.2.1 presents the BPTT derivations. The e-prop framework introduces different approximations of the exact formulas for the biologically-plausible learning signals, so that the computation of equation 8 can be done online in a memory efficient manner. In this regard, the authors of [1] introduce broadcast weights B_{jk} from output neuron k to network neuron j. Depending on how the broadcast weights are defined, different e-prop algorithms can be distinguished.

Two of the losses given as examples in the paper are the Mean Square Error (MSE) for regression and the Cross-Entropy error (CE), with K output neurons and $\pi_k^t = \text{softmax}(y_1^t, ..., y_K^t)$, for classification:

$$\mathcal{L}_{\mathcal{MSE}} = \frac{1}{2} \sum_{t,k} (y_k^t - y_k^{*,t})^2 \qquad \mathcal{L}_{\mathcal{CE}} = -\sum_{t,k} \pi_k^{*,t} \log \pi_k^t \tag{15}$$

From there, we can define ζ as a common variable, equal to y for regression and to π for classification, before considering the family of learning signals $(\tilde{L}_j^t)$ below:

$$\tilde{L}_j^t = \sum_k B_{jk} \sum_{t' \geq t} (\zeta_k^{t'} - \zeta_k^{*,t'}) \kappa^{t'-t} \tag{16}$$

We can therefore plug equations 16 and 8 together and switch the order of summation:

$$\frac{d\mathcal{L}}{dW_{ji}} \approx \sum_{k,t} B_{jk} \sum_{t' \geq t} (\zeta_k^{t'} - \zeta_k^{*,t'}) \kappa^{t'-t} e_{ji}^{t'} \tag{17}$$

$$\approx \sum_{k,t} B_{jk} (\zeta_k^t - \zeta_k^{*,t}) \sum_{t' \leq t} \kappa^{t'-t} e_{ji}^{t'} \tag{18}$$

Equation 18 shows that the gradient can be computed online and weights updated in that way, since the approximation only depends on past events. Moreover, unlike RTRL, e-prop complexity is only $\mathcal{O}(n^2)$, instead of $\mathcal{O}(n^4)$ [18], with n the number of neurons.

Symmetric e-prop For symmetric e-prop, we select the best approximation $\tilde{L}^t_{i,j} = L^t_{i,j}$ (ideal learning signal), resulting in $B_{jk} = W^{out}_{kj}$. Assuming that we have leaky neurons for outputs (see section 2.1), we obtain by derivation:

$$\frac{d\mathcal{L}}{dW_{ji}} \approx \sum_{k,t} W^{out}_{kj}(\zeta^t_k - \zeta^{*,t}_k) \sum_{t' \leq t} \kappa^{t'-t} e^{t'}_{ji} \tag{19}$$

Symmetric e-prop have been shown to be the most efficient method, and therefore the two other methods suggested below have not been reused in the literature [11] [30] [27] after this paper.

Random e-prop The broadcast weights (B_{jk}) have a constant random value, inspired by the theory of Broadcast Alignment of Feed-Forward Network [23].

Adaptive-prop Like random e-prop, the broadcast weights (B_{jk}) are initialized randomly, but a plasticity rule mirroring the update of the symmetric e-prop is implemented.

3.3 e-prop applications

Two main supervised learning applications are presented in [1]. All these experiments are conducted in an offline manner, so the online aspects of the method are not really evaluated, and a firing rate regularization term is also included in the losses.

Learning phoneme recognition Using the TIMIT dataset [12], the task is to extract phonemes from audio. Two classification subtasks are implemented: frame-wise recognition (one phoneme per defined fixed frame) and sequence recognition (sequence of phonemes). e-prop gives very good results, almost on-par with BPTT.

Difficult temporal credit assignment To deny the claim that e-prop cannot make the network learn long temporal dependencies, a simple experiment of integration of cues before asking for a binary decision has been designed, and the paper [1] shows that the network is able to successfully remember over more than 2 seconds. This is made possible thanks to the eligibility traces and the ALIF neurons.

Hardware In addition to successful software applications, some researchers were able to implement the e-prop algorithm on neuromorphic hardware, such as on a custom microchip ReckOn [11] or on SpiNNaker 2 [27]. This shows the wide relevance of this approach.

3.4 Detailed calculations

The detailed e-prop derivations for LIF and ALIF neurons, with surrogate gradients can be found in [2], [1] and [30]. In the paper [1], only derivations for the recurrent weights are presented, but not for the weights related to the input and output layers.

	e-prop hardcoded (EH) vs e-prop autodiff (EA)	EH vs BPTT autodiff
5 timesteps	e-08	0.1
25 timesteps	e-07	0.26
100 timesteps	e-07	0.57

Table 1: Norm of the differences between different gradients

4 Experiments with the e-prop framework

The e-prop framework has been introduced in 2019, however, despite the interesting results of e-prop, no current spiking neural network Python libraries offer an implementation of the method. The code shared by the author is now obsolete, programmed with Tensorflow 1 [25].

Therefore, I decided to implement it again, using the latest machine learning frameworks Jax [5] and Spyx [13], so that it can benefit from the latest GPU-acceleration, such XLA JIT-compiled routines on GPU.

4.1 Implementation of symmetry e-prop with *Jax*

An important question for the implementation is the way of computing the e-prop gradients. Of course, a hard-coded method is necessary to check the implementation, but all current deep-learning Python libraries offer auto-differentiation. However, it is needed to properly instruct the autodiff system to compute the e-prop gradients and not the exact BPTT gradients. In this regard, one can notice that removing the blue edges from the computational graph shown in Figure 2 does the trick. Indeed, by definition of e-prop, and without these links, we get the following equations:

$$L_j^t = \frac{\partial \mathcal{L}}{\partial z_j^t} = \frac{d\mathcal{L}}{dz_j^t} \qquad e_{ji}^t = \frac{dz_j^t}{dW_{ji}} \qquad \frac{d\mathcal{L}}{dW_{ji}} = \sum_t L_j^t e_{ji}^t \tag{20}$$

So the exact gradient in the altered computational graph is exactly the e-prop one of the full computational graph.

Therefore, we get almost for free the BPTT gradients with autodiff and the e-prop gradients when altering the computational graph. As done in the original source code, I first checked my Jax implementation by comparing hardcoded and autodiff gradients, and I also reinstalled an old Tensorflow environment to compare with their code. This can be found in the notebook *autodiff.ipynb*. Results are presented in Table 1. We clearly see that the gradient error between BPTT and e-prop increases as there are more timesteps, hopefully sublinearly it seems. In any case, the e-prop approximation is better on short durations and e-prop hardcoded perfectly coincides with e-prop autodiff.

To integrate the e-prop framework better with spyx, I implemented the LIF/ALIF and Leaky Output neuron models with the same interface as spyx, so that simple RNNs can be built with spyx and haiku support [15] (haiku should be updated to flax soon [14]).

I plan to do a pull request to the spyx github repository with my changes, to share this work with the community, and compare the execution time between jax and tensorflow.

4.2 Experiments with another dataset

Now that the e-prop framework is integrated in spyx, I tested the learning method with another classical benchmark dataset, called SHD [7]. It consists in digit recognition. Using a batch size of 256, 400 ALIF neurons, 60 iterations and 128 timesteps, I compared the BPTT and e-prop results in Figures 3 and 4, after an optimization with Adam. The code can be found in *shd.ipynb*.

The two networks have very close performance, which is also equivalent to the spyx baseline example of 0.76 with a feed-forward network. We can notice a bit of overfitting. In any case, symmetric e-prop does not alter the training or the generalization capability of the network.

4.3 Limitations and future work

This new implementation with Jax works well and it is very fast on GPU, but it can still be improved: a major feature would be to be able to test e-prop in an online setting, to reveal all the advantages of this method, instead of keeping an offline approach easy to compare with BPTT. Moreover, hyperparameter tuning could be done with the SHD dataset, to increase performance as well.

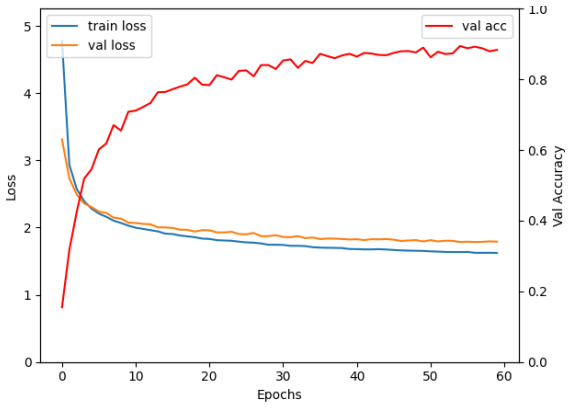

Figure 3: BPTT - train loss: 1.62, test accuracy: 0.76

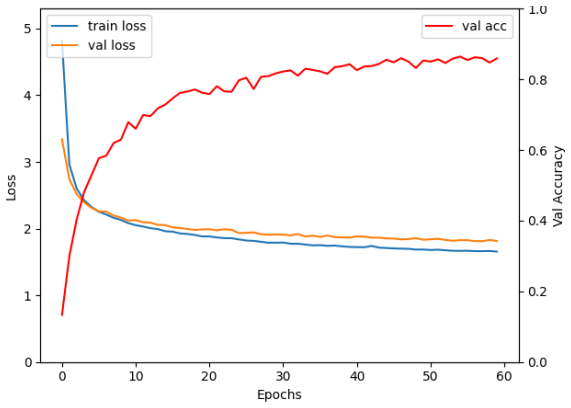

Figure 4: e-prop - train loss: 1.66, test accuracy: 0.75

To conclude, the e-prop framework is a very promising biologically-plausible online learning framework for spiking neural networks. It is useful for many applications in supervised learning and RL [1], in addition to being implementable for energy-efficient hardware [11]

However, additional research is still needed, as regards neuron models compatible with e-prop, the effect of deeper/multi-layer architectures, and an in-depth comparisons with other online SNN training methods which have emerged recently like decolle [19] or FPTT [32]. Compared to the classical deep learning literature, I have been surprised by the number of different techniques and standards I could find regarding SNNs: my hopes would be to help the community structure and unify this fascinating field.

References

[1] Guillaume Bellec et al. "A solution to the learning dilemma for recurrent networks of spiking neurons". In: *Nature Communications* 11.1 (July 2020). ISSN: 2041-1723. DOI: 10.1038/s41467-020-17236-y. URL: http://dx.doi.org/10.1038/s41467-020-17236-y.

[2] Guillaume Bellec et al. *Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets*. 2019. arXiv: 1901.09049 [cs.NE].

[3] Guillaume Bellec et al. *Long short-term memory and learning-to-learn in networks of spiking neurons*. 2018. arXiv: 1803.09574 [cs.NE].

[4] Sander M. Bohté, Joost N. Kok, and Han La Poutré. "SpikeProp: backpropagation for networks of spiking neurons". In: *The European Symposium on Artificial Neural Networks*. 2000. URL: https://api.semanticscholar.org/CorpusID:14069916.

[5] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018. URL: http://github.com/google/jax.

[6] Romain Brette. "Brains as Computers: Metaphor, Analogy, Theory or Fact?" In: *Frontiers in Ecology and Evolution* 10 (Apr. 2022). ISSN: 2296-701X. DOI: 10.3389/fevo.2022.878729. URL: http://dx.doi.org/10.3389/fevo.2022.878729.

[7] B. Cramer et al. "The Heidelberg Spiking Data Sets for the Systematic Evaluation of Spiking Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* (2020), pp. 1–14. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2020.3044364.

[8] Manon Dampfhoffer et al. "Are SNNs really more energy-efficient than ANNs? An in-depth hardware-aware study". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 2022 (2022), pp. 1–11. DOI: 10.1109/TETCI.2022.3214509. URL: https://cea.hal.science/cea-03852141.

[9] Jianhao Ding et al. *Optimal ANN-SNN Conversion for Fast and Accurate Inference in Deep Spiking Neural Networks*. 2021. arXiv: 2105.11654 [cs.NE].

[10] Jason K. Eshraghian et al. *Training Spiking Neural Networks Using Lessons From Deep Learning*. 2023. arXiv: 2109.12894 [cs.NE].

[11] Charlotte Frenkel and Giacomo Indiveri. "ReckOn: A 28nm Sub-mm2 Task-Agnostic Spiking Recurrent Neural Network Processor Enabling On-Chip Learning over Second-Long Timescales". In: *2022 IEEE International Solid- State Circuits Conference (ISSCC)*. IEEE, Feb. 2022. DOI: 10.1109/isscc42614.2022.9731734. URL: http://dx.doi.org/10.1109/ISSCC42614.2022.9731734.

[12] Garofolo, John S. et al. *TIMIT Acoustic-Phonetic Continuous Speech Corpus*. 1993. DOI: 10.35111/17GK-BN40. URL: https://catalog.ldc.upenn.edu/LDC93S1.

[13] Kade M. Heckel and Thomas Nowotny. *Spyx: A Library for Just-In-Time Compiled Optimization of Spiking Neural Networks*. 2024. arXiv: 2402.18994 [cs.NE].

[14] Jonathan Heek et al. *Flax: A neural network library and ecosystem for JAX*. Version 0.8.2. 2023. URL: http://github.com/google/flax.

[15] Tom Hennigan et al. *Haiku: Sonnet for JAX*. Version 0.0.10. 2020. URL: http://github.com/deepmind/dm-haiku.

[16] A. L. Hodgkin and A. F. Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of Physiology* 117.4 (Aug. 1952), pp. 500–544. ISSN: 1469-7793. DOI: 10.1113/jphysiol.1952.sp004764. URL: http://dx.doi.org/10.1113/jphysiol.1952.sp004764.

[17] Yangfan Hu, Huajin Tang, and Gang Pan. *Spiking Deep Residual Network*. 2020. arXiv: 1805.01352 [cs.NE].

[18] Kazuki Irie, Anand Gopalakrishnan, and Jürgen Schmidhuber. *Exploring the Promise and Limits of Real-Time Recurrent Learning*. 2024. arXiv: 2305.19044 [cs.LG].

[19] Jacques Kaiser, Hesham Mostafa, and Emre Neftci. "Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE)". In: *Frontiers in Neuroscience* 14 (May 2020). ISSN: 1662-453X. DOI: 10.3389/fnins.2020.00424. URL: http://dx.doi.org/10.3389/fnins.2020.00424.

[20] Youngeun Kim et al. *Neural Architecture Search for Spiking Neural Networks*. 2022. arXiv: 2201.10355 [cs.NE].

[21] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539. URL: http://dx.doi.org/10.1038/nature14539.

[22] G. Li et al. "Brain inspired computing: a systematic survey and future trends". In: (2023). DOI: 10.36227/techrxiv.21837027.v1.

[23] Timothy P. Lillicrap et al. *Random feedback weights support learning in deep neural networks*. 2014. arXiv: 1411.0247 [q-bio.NC].

[24] Wolfgang Maass. "Networks of spiking neurons: The third generation of neural network models". In: *Neural Networks* 10.9 (1997), pp. 1659–1671. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(97)00011-7. URL: https://www.sciencedirect.com/science/article/pii/S0893608097000117.

[25] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[26] Fabrizio Ottati et al. *To Spike or Not To Spike: A Digital Hardware Perspective on Deep Learning Acceleration*. 2024. arXiv: 2306.15749 [cs.NE].

[27] Amirhossein Rostami et al. "E-prop on SpiNNaker 2: Exploring online learning in spiking RNNs on neuromorphic hardware". In: *Frontiers in Neuroscience* 16 (Nov. 2022). ISSN: 1662-453X. DOI: 10.3389/fnins.2022.1018006. URL: http://dx.doi.org/10.3389/fnins.2022.1018006.

[28] Amirhossein Tavanaei et al. "Deep learning in spiking neural networks". In: *Neural Networks* 111 (Mar. 2019), pp. 47–63. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2018.12.002. URL: http://dx.doi.org/10.1016/j.neunet.2018.12.002.

[29] Manuel Traub et al. *Learning Precise Spike Timings with Eligibility Traces*. 2020. arXiv: 2006.09988 [cs.NE].

[30] Werner van der Veen. *Including STDP to eligibility propagation in multi-layer recurrent spiking neural networks*. 2022. arXiv: 2201.07602 [cs.NE].

[31] P.J. Werbos. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560. DOI: 10.1109/5.58337.

[32] Bojian Yin, Federico Corradi, and Sander M. Bohte. *Accurate online training of dynamical spiking neural networks through Forward Propagation Through Time*. 2022. arXiv: 2112.11231 [cs.NE].

[33] Zhaokun Zhou et al. *Spikformer: When Spiking Neural Network Meets Transformer*. 2022. arXiv: 2209.15425 [cs.NE].

[34] David Zipser. "A Subgrouping Strategy that Reduces Complexity and Speeds Up Learning in Recurrent Networks". In: *Neural Computation* 1.4 (Dec. 1989), pp. 552–558. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.552. eprint: https://direct.mit.edu/neco/article-pdf/1/4/552/811939/neco.1989.1.4.552.pdf. URL: https://doi.org/10.1162/neco.1989.1.4.552.