Recurrent Spiking Neural Networks
A quick overview of the e-prop online learning framework

Florent Pollet

Modelling in neuroscience
ENS Paris-Saclay

March 26, 2024

Introduction

Review of paper *A solution to the learning dilemma for recurrent networks of spiking neurons* Bellec et al., 2020, following papers Bellec et al., 2019 and Bellec et al., 2018.

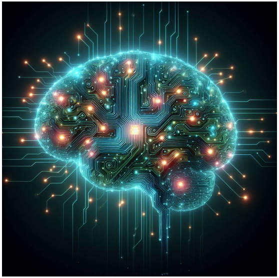
- Artificial Neural Networks (ANN) consume more and more power
- Brain energy-efficiency is much higher than current electronic systems

$\Longrightarrow$ Development of neuromorphic computing

- Spiking Neural Networks are a promising brain-inspired alternative to ANN
- However the most efficient methods to train them are not biologically plausible

$\Longrightarrow$ How to design a biologically-plausible and efficient training strategy?

e-prop



"Neuromorphic computing" by DALL-E 3

Outline

1. Recurrent Spiking Neural Networks
   - Neuron models
   - Coding
   - Architectures
   - Training
2. The e-prop learning framework
   - Biological considerations
   - e-prop definition and derivations
   - e-prop applications
3. Experiments with e-prop
   - Jax implementation for Spyx
   - e-prop on another dataset

Leaky Integrate-and-Fire (LIF) model - continuous

Analogy between ion and electron transfers, by Ramon Stein (1967):

- $V_m(t)$ membrane potential
- $I(t)$ input current
- $\tau_m = C_m R_m$ membrane time constant
- C_m membrane capacitor
- R_m membrane resistor
- Soft reset to V_{reset} after a spike emission (action potential) when V_{th} is crossed

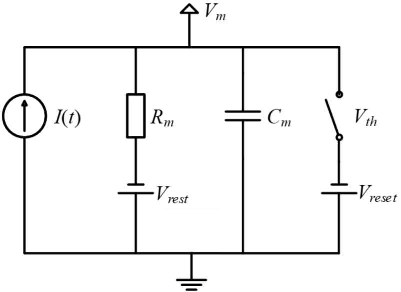
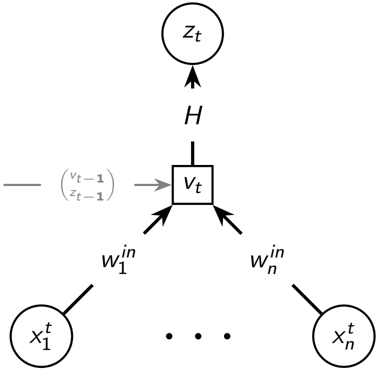


Figure: Equivalent circuit from Chen et al., 2021

$$\tau_m \frac{dV_m(t)}{dt} = -[V_m(t) - V_{rest}] + R_m I(t)$$

Leaky Integrate-and-Fire (LIF) model - discrete

- internal/hidden state v^t
- observable state $z^t \in \{0,1\}$
- n inputs $(x_i^t)_i \in \{0,1\}$
- w^{in} the input synapse weights
- δt the discrete time step size (1 ms for instance)
- $\alpha = \exp(-\delta t/\tau_m)$ the decay time constant
- H the Heaviside step function (non differentiable)



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

Adaptive Leaky Integrate-and-Fire (ALIF) model

Introduced by Bellec et al., 2018, inspired by a biological phenomenon: spike-frequency adaptation (SFA)
Neuron with SFA $\implies$ more expressivity for the neural network

- a_t the deviation in the baseline threshold voltage
- β scaling factor
- $\rho = \exp(-\delta t/\tau_a)$ the exponential decay parameter
- τ_a the adaptation time constant of the threshold decay

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

Refractory period can be enforced for both LIF and ALIF models by setting z^t to 0 for a few timesteps

LIF - ALIF comparison

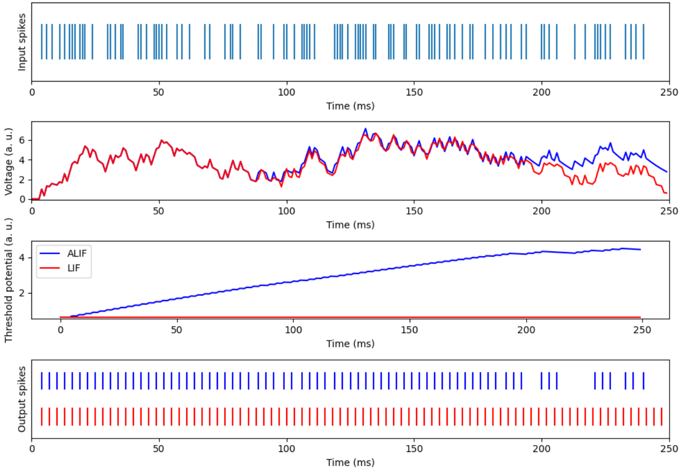


Figure: Comparison of LIF and ALIF neuron models with a random input spike train

Other famous neuron models

- Izhikevich: includes the refractory period in the model, but shown to perform badly with e-prop in van der Veen, 2022
- Hodgkin-Huxley: the most biologically accurate, conductance based instead of current input based, but more computationally-intensive
- Leaky output neurons

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

  - Real-valued, not spiking
  - Decay factor $\kappa = exp(-\delta_t/\tau_{out})$ by analogy to the LIF neurons
  - Bias b
  - Code implementation: $y^t = \kappa y^{t-1} + (1-\kappa)\sum_j w_j z_j^t$

Input data encoding

- Rate coding
  - Widespread in the community
  - Not the sparsest (not ideal for energy efficiency)
- Temporal coding
  - Very sparse
  - Can resort to logscale to increase the range
- Other coding schemes
  - Delta modulation coding, inspired from neurophysiology
- Decoding can be done in a similar fashion

Input data coding

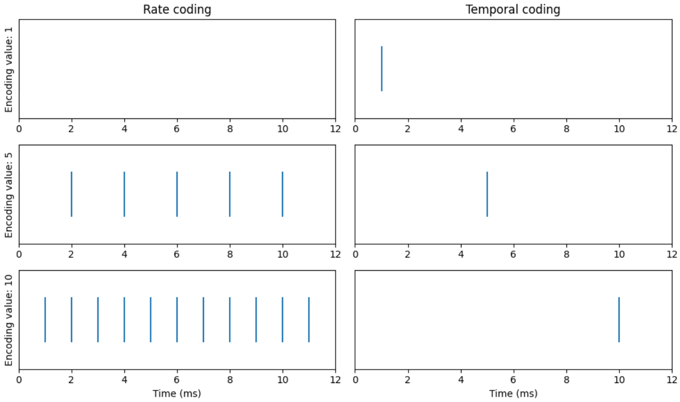


Figure: Rate and temporal coding comparison for simple integer values

SNN architectures

- Classical feed-forward architectures
  - Can replicate famous ANNs like ResNet, VGG
  - More common, mainly for Computer Vision or tasks related to Robotics
- Recurrent architectures (RSNN)
  - Closer to brain networks
  - Recurrent weights
  - The paper Bellec et al., 2018 introduces the Long short-term memory Spiking Neural Networks (LSNN), which is an RNN with a mix of LIF and ALIF neurons, thus showing SFA
- Transformers: early development Zhou et al., 2022

LSNN computational graph

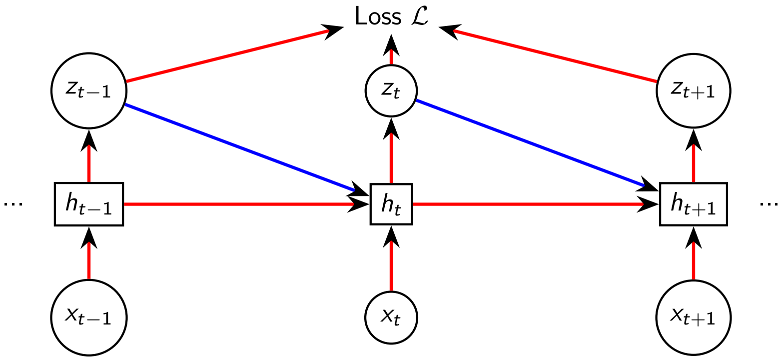


Figure: Example of a computational graph for a Spiking Neural Network. Inputs are written $(x_t)_t$, outputs $(z_t)_t$, and hidden states of the network $(h_t)_t$. Blue arrows are only present for Recurrent Spiking Neural Networks.

SNN training strategies

Main issue: activation functions are not differentiable

- ANN-to-SNN conversion: easier approach and best results for Deep SNN
- Spike-timing-dependent plasticity (STDP)
  - Most biologically-plausible method
  - Variation of Hebbian learning *Neurons who fire together, wire together*
  - Temporal order taken into account (pre- and post-synaptic)
  - Medium performance
- BackPropagation Through Time Werbos, 1990
  - Similar from RNN/LSTM training in Deep Learning
  - Uses surrogate gradients like arctan, Gaussian functions or even Straight-Through Estimators
  - Good performance
- Real-Time Recurrent Learning
  - Online method
  - Bad complexity $O(n^4)$, with n the number of neurons

Biological considerations

In the brain:

- Eligibility traces: preceding activity leaves traces at molecular level (calcium ions or CAMKII enzymes)
- Learning signals: top down signals thanks to dopamine or acetylcholine, to inform the neurons of behavioral results

Figure: Credits https://around.uoregon.edu/content/uo-neuroscientists-get-new-view-how-neurons-communicate

e-prop definitions

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:

- $\frac{d.}{d.}$ 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 h_{t+1} are removed: see next slide

$$L_j^t = \frac{\partial \mathcal{L}}{\partial z_j^t} \quad (\approx \frac{d\mathcal{L}}{dz_j^t})$$

$$\begin{aligned} e_{ji}^t &= \left[\frac{dz_j^t}{dW_{ji}}\right]_{local} \\ &= \frac{\partial z_j^t}{\partial h_j^t} \cdot \sum_{t' \leq t} \frac{\partial h_j^t}{\partial h_j^{t-1}} \cdots \frac{\partial h_j^{t+1}}{\partial h_j^{t'}} \cdot \frac{\partial h_j^{t'}}{\partial W_{ji}} \end{aligned}$$

e-prop derivations - truncated graph

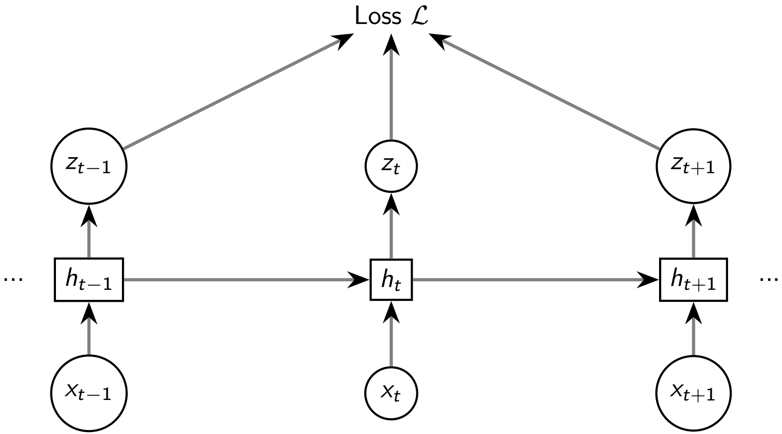


Figure: Truncated graph for local derivatives

e-prop derivations - chain rule

From the BPTT original paper Werbos, 1990, we have this factorization of the derivatives:

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

We can expand this expressive using a recursive formula:

$$\frac{d\mathcal{L}}{dh_j^t} = \frac{d\mathcal{L}}{dz_j^t} \cdot \frac{\partial z_j^t}{\partial h_j^t} + \frac{d\mathcal{L}}{dh_j^{t+1}} \cdot \frac{\partial h_j^{t+1}}{\partial h_j^t}$$

See next slide for to visualize on the graph

e-prop derivations - chain rule

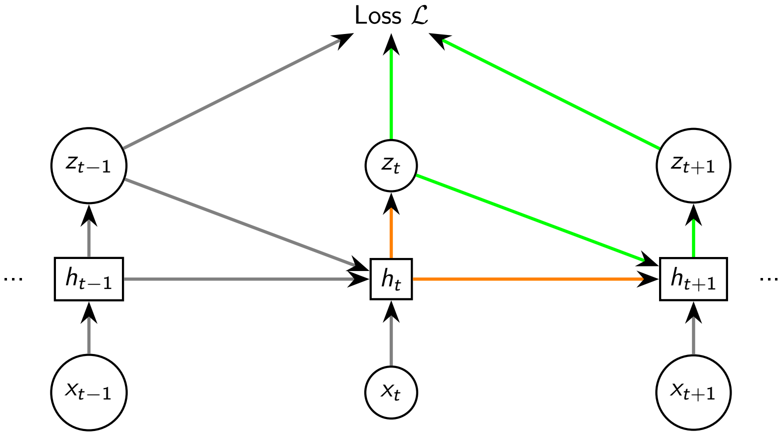


Figure: Chain rule for e-prop derivations $\frac{d\mathcal{L}}{dh_j^t}$

e-prop derivations - main equation

We can therefore write the full recursion and reorder the sum, to get the e-prop main equation:

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

To enforce an online setting, one approximate $\frac{d\mathcal{L}}{dz_j^t}$ by its partial derivative (see next slide).

e-prop derivations - learning signal approximation

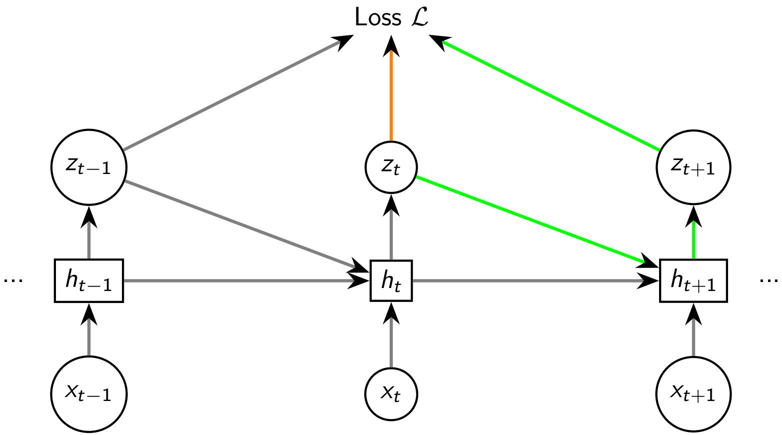


Figure: Learning signal approximation for e-prop derivations (orange: partial derivative, orange+green: total derivative)

Broadcast weights

Supervised learning here (for Reinforcement Learning: reward e-prop):

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

Family of learning signals with broadcast weights (B_{jk}):

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

Hence, the e-prop gradient:

$$\begin{aligned}\frac{d\mathcal{L}}{dW_{ji}} &\approx \sum_{k,t} B_{jk}\sum_{t'\geq t}(\zeta_k^{t'} - \zeta_k^{*,t'})\kappa^{t'-t}e_{ji}^t \\ &\approx \sum_{k,t} B_{jk}(\zeta_k^t - \zeta_k^{*,t})\sum_{t'\leq t}\kappa^{t-t'}e_{ji}^{t'}\end{aligned}$$

$\sum_{t'\leq t}\kappa^{t-t'}e_{ji}^{t'}$: temporal low-pass filter of the eligibility traces

e-prop variants

How to choose the broadcast weights?

- symmetric e-prop (ideal learning signal, best approximation): $B_{jk} = W_{kj}^{out}$
- random e-prop: constant random weights $\implies$ bad performance, except for multi-layer
- adaptive e-prop: random weights updated with the same rule as the other weights

Detailed calculations for each method can be found in Bellec et al., 2020.

Learning phoneme recognition

- TIMIT dataset from Garofolo, John S. et al., 1993
- Preprocessing to extract Mel coefficients, as inputs
- No encoding, continuous input in $[0, 1]$

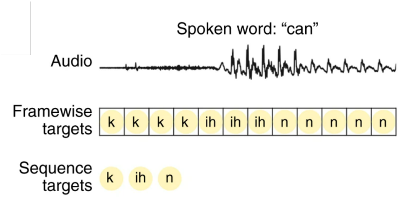


Figure: TIMIT tasks Bellec et al., 2020

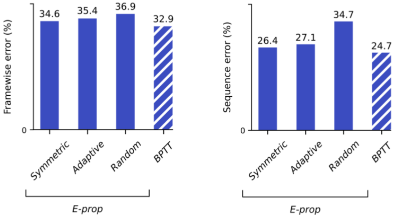


Figure: TIMIT results Bellec et al., 2020

Difficult temporal credit assignment

- Goal: choose the side with most cues
- Interest: prove the learning of long-time dependencies

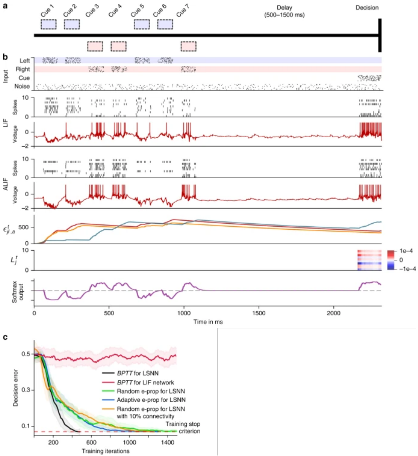


Figure: Task results Bellec et al., 2020

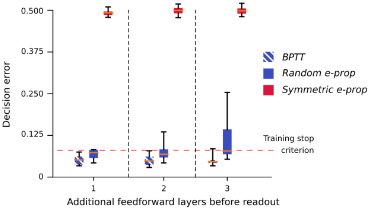


Figure: Architecture comparison Bellec et al., 2020

symmetric e-prop through auto-differentiation

Why Jax instead of PyTorch? It can be XLA JIT-compiled and the code can fully run on GPU with optimized kernels.

How to leverage auto-differentiation from modern machine learning frameworks to compute e-prop?

By stopping the gradients from z_t to h_{t+1}, we obtain the truncated computational graph shown in a previous slide, that leads to the following simplifications:

$$L_j^t = \frac{\partial \mathcal{L}}{\partial z_j^t} = \frac{d\mathcal{L}}{dz_j^t}$$

$$e_{ji}^t = \frac{dz_j^t}{dW_{ji}}$$

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

$\implies$ We can use auto-differentation to compute symmetric e-prop.

Jax implementation for Spyx: gradient computations

Figure: Jax logo - Bradbury et al., 2018

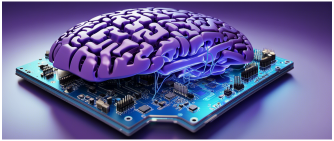


Figure: Spyx logo - Heckel and Nowotny, 2024

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

Table: Norm of the differences between different gradients
EH = e-prop hardcoded
EA = e-prop autodiff

Spiking Heidelberg Digits (SHD) dataset

- Benchmark dataset for spiking neural networks Cramer et al., 2020
- Audio digit classification in German and English
- Roughly 1000 samples
- Delta modulation encoding via synthetic cochlear

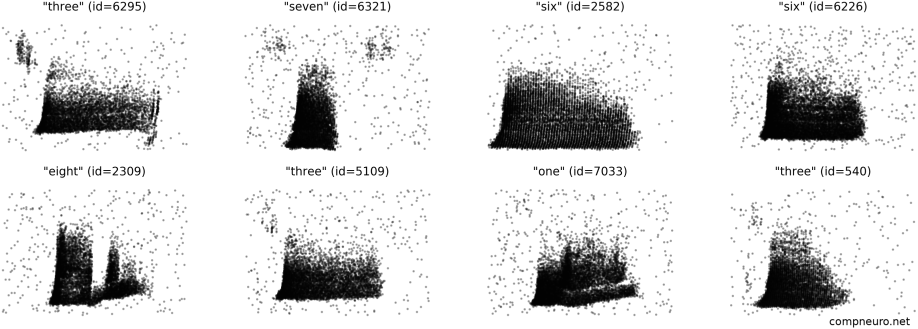


Figure: SHD dataset (x-axis: time, y-axis: channel)

LSNN with symmetric e-prop test on SHD dataset

- Network parameters: batch size of 256 (not online), 400 ALIF neurons, 60 iterations and 128 timesteps
- Training time: roughly 2 min (this is fast!)
- Baseline benchmark test accuracy: 0.76 (state of the art with SNN: 0.95)

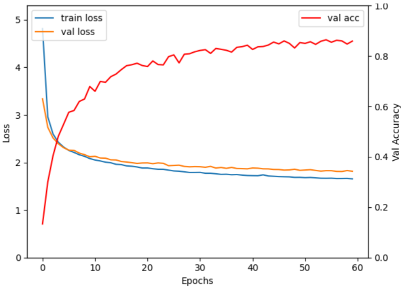


Figure: e-prop - train loss: 1.66, test accuracy: 0.75

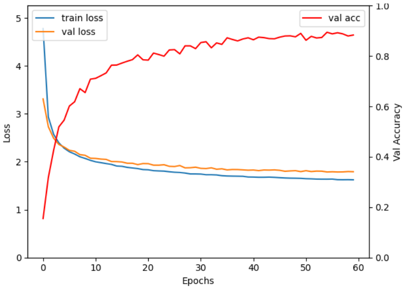


Figure: BPTT - train loss: 1.62, test accuracy: 0.76

LSNN with symmetric e-prop test on SHD dataset

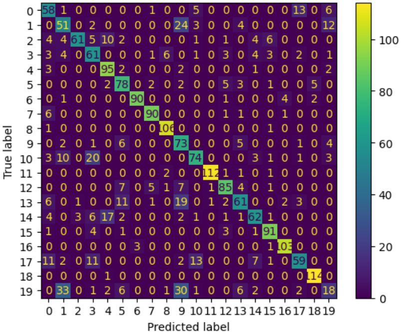


Figure: e-prop - train loss: 1.66, test accuracy: 0.75

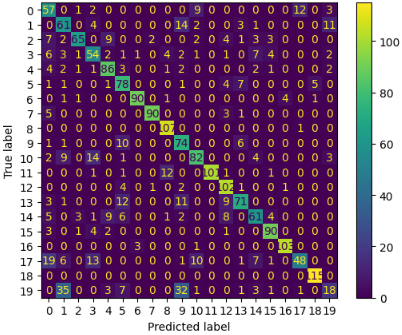


Figure: BPTT - train loss: 1.62, test accuracy: 0.76

Limitations and further work

- Connect this work to STDP, another biological learning mechanism, which is not discussed in the paper but a bit in van der Veen, 2022
- Compare e-prop to Real-Time Recurrent Learning (RTRL) and other variations, such as Online Training Through Time (OTTT) and Online SpatioTemporal Learning (OSTL) Summe, Schaefer, and Joshi, 2023 - or at least study more the online advantages of e-prop approximations
- e-prop may be more interesting for Reinforcement Learning (RL), with reward e-prop, because supervised learning is dominated by ANN

- More generally, think about good heuristics for SNN design because of a very large hyperparameter search space compared to ANN
- Another interesting dataset to test Brain-To-Text, with real human spiking data `https://eval.ai/web/challenges/challenge-page/2099/`

Thanks! Any questions?

Code: `https://github.com/florian6973/btt-spyx`

References I

- Bellec, Guillaume et al. (2018). *Long short-term memory and learning-to-learn in networks of spiking neurons*. arXiv: 1803.09574 [cs.NE].
- Bellec, Guillaume et al. (2019). *Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets*. arXiv: 1901.09049 [cs.NE].
- Bellec, Guillaume et al. (July 2020). "A solution to the learning dilemma for recurrent networks of spiking neurons". In: *Nature Communications* 11.1. ISSN: 2041-1723. DOI: 10.1038/s41467-020-17236-y. URL: http://dx.doi.org/10.1038/s41467-020-17236-y.
- Bradbury, James et al. (2018). *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. URL: http://github.com/google/jax.
- Chen, Jiankun et al. (2021). *SAR Image Classification Based on Spiking Neural Network through Spike-Time Dependent Plasticity and Gradient Descent*. arXiv: 2106.08005 [cs.CV].

References II

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

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

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

Summe, Thomas, Clemens JS Schaefer, and Siddharth Joshi (2023). *Estimating Post-Synaptic Effects for Online Training of Feed-Forward SNNs*. arXiv: 2311.16151 [cs.NE].

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

References III

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

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