Improving eligibility propagation using Izhikevich neurons in a multilayer RSNN.

Presentation 1: Formalizing the framework & initial steps

Werner van der Veen
(w.k.van.der.veen.2@umcg.nl)

October 13, 2020

Bellec's e-prop

Bellec, G., Scherr, F., Subramoney, A., Hajek, E., Salaj, D., Legenstein, R., & Maass, W. (2020). A solution to the learning dilemma for recurrent networks of spiking neurons.

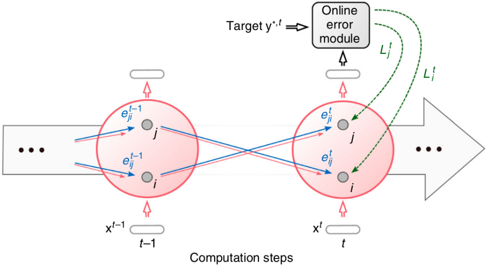
E-prop derived from BPTT:

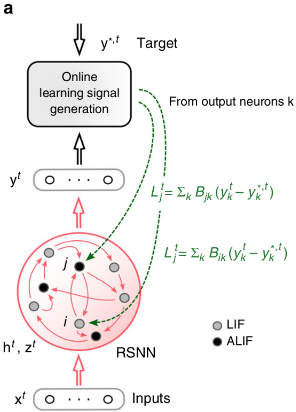
$$\frac{dE}{dW_{ji}} \stackrel{\text{def}}{=} \sum_t \frac{dE}{dz_j^t} \cdot \left[\frac{dz_j^t}{dW_{ji}}\right]_{\text{local}} \tag{1}$$

$$\stackrel{\text{def}}{=} \sum_t L_j^t \cdot e_{ji}^t \tag{2}$$

$$\stackrel{\text{def}}{=} \sum_t \sum_k B_{jk} \left(y_k^t - y_k^{*,t}\right) \cdot \frac{\partial z_j^t}{\partial \mathbf{h}_j^t} \left(\frac{\partial \mathbf{h}_j^t}{\partial \mathbf{h}_j^{t-1}} \cdot \epsilon_{ji}^{t-1} + \frac{\partial \mathbf{h}_j^t}{\partial W_{ji}}\right) \tag{3}$$

Bellec Architecture





Traub

Traub, M, Butz, M. V., Baayen, R. H., & Otte, S. (2020). Learning Precise Spike Timings with Eligibility Traces.

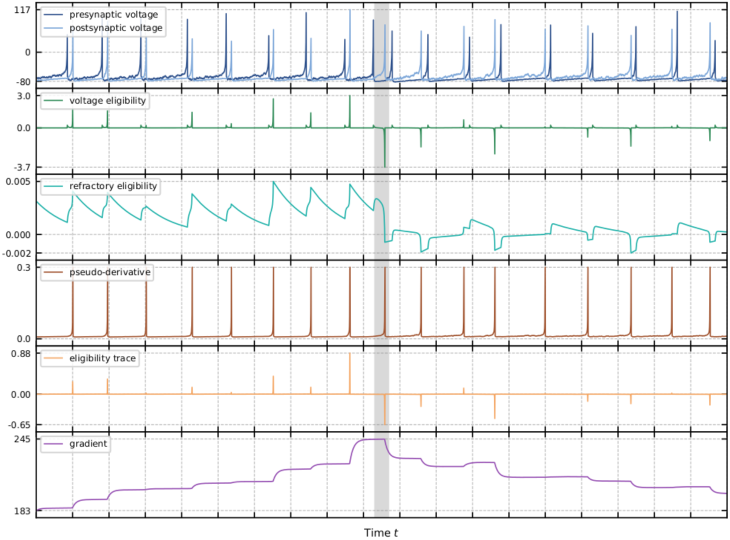
Corrects STDP behavior for negative learning.

For (A)LIF, this is realized when hard resetting v to zero after refractory period, because this automatically resets e.

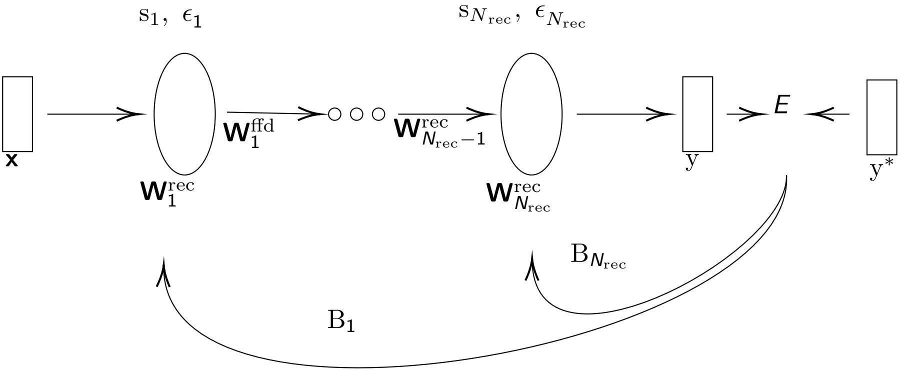
Izhikevich neurons naturally incorporate refractory periods and don't necessitate explicit refractory resets.

$$
\begin{aligned}
\tilde{u}_j^t &= v_j^t - \left(v_j^t + 65\right) z_j^t\right) \\
\tilde{v}_j^t &= u_j^t + 2 z_j^t \\
u_j^{t+1} &= \tilde{u}_j^t + \delta t \left(0.004 \tilde{v}_j^t - 0.02 \tilde{u}_j^t\right) \\
v_j^{t+1} &= \tilde{v}_j^t + \delta t \left(0.04 \left(\tilde{v}_j^t\right)^2 + 5 \tilde{v}_j^t + 140 - \tilde{u}_j^t + I_j^t\right)
\end{aligned}
$$

Izhikevich e-prop simulation



My own idea: multilayer Izhikevich



This is already partly functional!

Planning

Work done so far:

1. Verified & reproduced Traub's Izhikevich and STDP-(A)LIF simulations in a dynamic framework;

2. Implemented a multi-layer RSNN with basic visualization and input/output Poisson streams (via Bernoulli distribution);

3. Implemented all three neuron models in the RSNN and unit tests;

4. Implemented learning signal L. Currently broadcast alignment only;

5. Implemented a hyperparameter sweep using mixed-integer linear programming: grid search over integers, Nelder-Mead optimization for real numbers.

Concatenated matrices

Challenge: can't apply recursive update on a layer, and then feedforward, because that means that a time step would pass twice.

My current solution: concatenate the recurrent weights of layer A and feedforward weights between A and B prior to the dot product. Also concatenate two layers:

$$\text{Layer A: } \begin{pmatrix} a_0 \\ a_1 \end{pmatrix}, \text{ Layer B: } \begin{pmatrix} b_0 \\ b_1 \end{pmatrix}$$

$$\text{Weights A (recurrent): } \begin{pmatrix} 0 & w^{\mathrm{a}}_{1,0} \\ w^{\mathrm{a}}_{0,1} & 0 \end{pmatrix}$$

$$\text{Weights A to B: } \begin{pmatrix} w^{\mathrm{ab}}_{0,0} & w^{\mathrm{ab}}_{1,0} \\ w^{\mathrm{ab}}_{0,1} & w^{\mathrm{ab}}_{1,1} \end{pmatrix}$$

Concatenated matrices

Instead of separate recursive and feedforward updates, i.e.

$$\begin{pmatrix} a_0 \\ a_1 \end{pmatrix} \leftarrow A + \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} \cdot \begin{pmatrix} 0 & w^{a}_{1,0} \\ w^{a}_{0,1} & 0 \end{pmatrix} = \begin{pmatrix} a_1 w^{a}_{1,0} \\ a_0 w^{a}_{0,1} \end{pmatrix}$$

$$\begin{pmatrix} b_0 \\ b_1 \end{pmatrix} \leftarrow B + \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} \cdot \begin{pmatrix} w^{ab}_{0,0} & w^{ab}_{1,0} \\ w^{ab}_{0,1} & w^{ab}_{1,1} \end{pmatrix} = \begin{pmatrix} a_0 w^{ab}_{0,0} + a_1 w^{ab}_{1,0} \\ a_0 w^{ab}_{0,1} + a_1 w^{ab}_{1,1} \end{pmatrix}$$

we do it in a single step:

$$\left(\begin{array}{c} a_0 \\ a_1 \\ \hline b_0 \\ b_1 \end{array}\right) \leftarrow C + \left(\begin{array}{c} a_0 \\ a_1 \\ \hline b_0 \\ b_1 \end{array}\right) \cdot \left(\begin{array}{cc|cc} 0 & w^{a}_{1,0} & 0 & 0 \\ w^{a}_{0,1} & 0 & 0 & 0 \\ \hline w^{ab}_{0,0} & w^{ab}_{1,0} & 0 & 0 \\ w^{ab}_{0,1} & w^{ab}_{1,1} & 0 & 0 \end{array}\right)$$

This way, neurons in layer A aren't computed twice. Maybe we can take this a step further and compute the whole network in one go? Drawback would be large weight matrix with *lots* of zeros.

Planning

Next steps:

1. Implement Bellec TIMIT (this requires hooking up the phoneme dataset and reading/visualization functions, expecting quite some work).

2. Try to reproduce Bellec's results. This may naturally lead to architectural improvements;

3. Improve the visualization, both for analytical purposes as well as detecting errors;

4. Refactor the code, improve 'time sinks', etc;

Questions

1. Input activity fades out in deeper layers – neurons in second layer rarely cross threshold. Any good ways to overcome this, besides obvious solutions such as increasing the weights?

2. How to convert from output spikes back to real-valued output? Currently using exponential moving average over spikes.

3. Ideas for very simple tasks to verify that learning can occur?