Reproducibility Study of Spike-Train Level Backpropagation for Training Deep Recurrent Spiking Neural Networks

Julien Verecken, Alex Hoffman and Srikanth Amudala
Email: julien.verecken@mail.mcgill.ca, alexander.hoffman@mail.mcgill.ca, srikanth.amudala@mail.mcgill.ca

***Abstract*—In this paper, we examine the findings presented in the novel spike-train level backpropagation algorithm ST-RSBP, trained on different types of spiking neural networks (SNN). The ST-RSBP method improves upon existing spike train level propagation algorithms by improving the accuracy of the differentiation of activation layers and by adding backpropagation to recurrent neural connections. We analyze the ST-RSBP differentiation technique through an ablation study of the algorithm, where we alter or remove parts of the released code to examine their impact on the reported results. Through our analysis we conclude that the paper does make improvements to the performance of their previous algorithm.**

I. Introduction

Deep neural networks (DNN) have been performing very effectively in recent years in natural language processing, speech recognition, visual object recognition, object detection, and many other areas. An alternative to artificial neural network architectures (ANN) like the DNN is the spiking neural network, which emulates the processing of neurons in the brain. Signals travel between neurons in the form of short electrical pulses, not 32-bit floating point numbers. The pulses are also known as action potentials or spikes [3].

A spiking neural network (SNN) attempts to model the brain's processing architecture by propagating data in the form of spike trains, which are series of spikes in the time domain. These spike trains build up voltage potentials at synapses, which fire once reaching a voltage threshold. The main advantage of SNN over classical machine learning methods is their capacity to achieve trade off between computational power costs and prediction accuracy [8]. In SNN where the information is encoded in the firing rate, the inference delay of the model, in a real time application, can be trade with the uncertainty over the prediction, effectively reducing power consumption in an embedded system. Platforms such as TrueNorth [9] and ODIN [10] are developed to integrate neuromorphic networks in a low power, embedded environment.

A. SNN Architecture

SNNs depend on a time-domain model for the synapses of the network. The ST-RSBP paper uses the popular leaky integrate-and-fire model and a first order synaptic model [1]. Eq. 1 shows the model's differential equations for membrane potential $u_i(t)$ of neuron i and its input current $\alpha_i(t)$. The differential system of $\alpha_i(t)$ is has as input an weighted sum of spiking trains of neurons j.

The time constants τ_m and τ_s are parameters of the models which affect voltage and current behavior in time. We vary τ_m in this study to determine the system's sensitivity to this hyperparameter. When the membrane potential has reach a voltage threshold, the neuron i fires and is inactive for a determined refractory period.

$$\begin{aligned} \tau_m \frac{u_i}{dt} &= -u_i(t) + R\alpha_i(t) \\ \tau(s)\frac{\alpha_i}{dt} &= -\alpha_i(t) + \sum_j w_{ij} \sum_{t_j^{(f)}} D(t - t_j^{(f)}) \end{aligned} \tag{1}$$

B. SNN Backpropagation

One method for applying backpropagation through RSNNs is Backpropagation Through Time (BPTT), which unfolds the recurrent network into a completely feedforward network before applying traditional backpropagation [1], resulting in a high computational cost and exploding/vanishing gradient issues. Spike-Train level RSNN Backpropagation (ST-RSBP), does not convert the RSNN into a feedforward network. Instead, it tracks spikes over time during forward propagation and performs backpropagation at the spike train level using spike train level post synaptic potential to capture temporal interactions between neurons.

C. Spike-train Level Post-synaptic Potential (S-PSP)

S-PSP (Eq. (2)), defined by the closed form solution of the neuron differential equations, captures the spike-train level interactions between a pair of pre/post-synaptic neurons.

$$\begin{aligned} u_i(t) &= \sum_j w_{ij} \sum_{t_j^{(f)}} \epsilon(t - \hat{t}_i^{(f)}, t - t_j^{(f)}) \\ e_{ij} &= \sum_{t_i^{(f)}} \sum_{t_j^{(f)}} \epsilon(t_i^{(f)} - \hat{t}_i^{(f)}, t_i^{(f)} - t_j^{(f)}) \end{aligned} \tag{2}$$

The S-PSP e_{ij} therefore characterizes the aggregated effect of the spike train of the neuron j on the membrane potential of the neuron i and its firing activities. S-PSPs allow consideration of the temporal dynamics and recurrent connections of an RSNN across all firing events at the spike-train level without expensive unfolding through time and backpropagation time point by time point. We show the effect of pre-synaptic neuron j on post-synaptic neuron i with e_{ij} (the S-PSP)

We can perform weighted sum of all S-PSP to a post neuron i with the T-PSP (3): a_i, the total accumulated membrane potential, ν is the firing threshold, o_i is the number of firings (integrated potential over time/threshold), w_{ij} is the weight for a connection in the network.

$$\begin{aligned} a_i =& \sum_j w_{ij} e_{ij} \\ o_i =& g(a_i) \simeq \frac{a_i}{\nu} \end{aligned} \tag{3}$$

II. Reproducibility Goals

We aim to reproduce the results of the authors and verify the explanation of those results through an ablation study. The paper claims several key improvements to the spiking neural network backpropagation technique of the [2].

A. Error Gradient Approximation

One main improvement is the more precise computation of the output gradient with respect to weights. We aim to test whether this change in the algorithm is beneficial to test error. When computing $\frac{\partial a_i^k}{\partial w_{ij}^k}$, the previous algorithm ignored the dependence of e_{ij}^k on w_{ij}^k. The ST-RSBP paper illustrates their improved derivation of $\frac{\partial a_i^k}{\partial w_{ij}^k}$ in the equation 4, using the chain rule to expand the derivative into two terms rather than just the left term. The algorithm also uses a polynomial function to approximate $\frac{\partial e_{il}^k}{\partial o_i^k}$ in equation (3).

$$\begin{aligned} \frac{\partial a_i^k}{\partial w_{ij}^k} &= e_{ij}^k + \frac{1}{v^k} \frac{\partial a_i^k}{\partial w_{ij}^k} \sum w_{il}^k \frac{\partial e_{ij}^k}{\partial o_i^k} \\ \frac{\partial a_i^k}{\partial w_{ij}^k} &= \frac{e_{ij}^k}{1 - \frac{1}{v^k} \sum w_{il}^k \frac{\partial e_{ij}^k}{\partial o_i^k}} \end{aligned} \tag{4}$$

The previous paper's algorithm, HM2-BP, approximates the derivative of Eq. 4 as $\frac{\partial a_i^k}{\partial w_{ij}^k} = e_{ij}^k$

B. S-PSP Time constant Hyperparameter Search

We also performed a simple hyperparameter search of the S-PSP time constant τ_m to see how sensitive the algorithm is to its value.

C. S-PSP Computation

The S-PSP computation is detailed in the additional materials of the paper and is, in short, a component of the analytical solution of the membrane potential. However, the interactions between S-PSP and the input and output firing rates are not derived analytically and must be approximated when used in the backpropagation. While a linear dependence could be assumed, the authors insisted on simulating those interactions in MATLAB to come up with a better approximation of the dependence. Their results are shown in Fig. 1. A linear dependence is accurate for the dependence of e_{ij} with respect to o_i but this is not the case for o_j and a 4^{th} order polynomial is fitted. The interaction simulation is therefore analysed and the hypothesis of a polynomial fit is discussed.

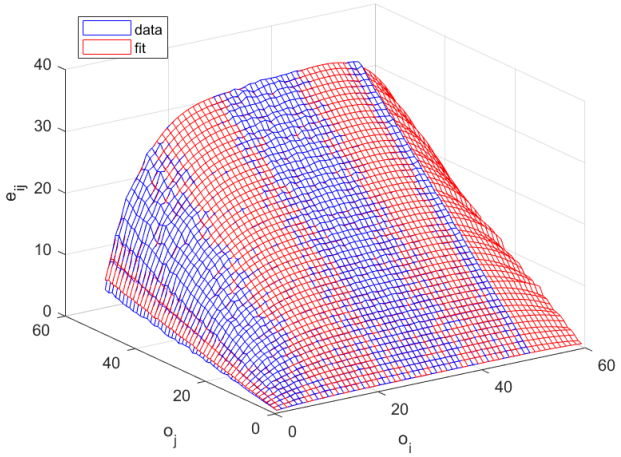


Fig. 1: e_{ij} dependence on o_i and o_j

III. Methodology

A. Error Gradient Approximation

Removing the precise gradient calculation from the ST-RSBP algorithm was straightforward. We changed the code in the spiking.cu file to set $\frac{\partial a_i^k}{\partial w_{ij}^k} = e_{ij}^k$ without considering $\frac{\partial e_{il}^k}{\partial o_i^k}$. We only made this change to the spiking layer file. We then evaluated test error on the MNIST dataset, which was the only dataset we could access and use without errors in the code.

B. S-PSP Time constant Hyperparameter Search

Performing a hyperparameter search of the S-PSP time constant τ_m involved generating new $\frac{\partial e_{il}^k}{\partial o_i^k}$ effect ratio files for each value of τ_m using the MATLAB script given in the ST-RSBP code repository. To train using the new settings, we updated the τ_m parameter in the configuration file.

C. S-PSP Computation

A new MatLab simulation is performed to compare with the original and generate the required fitting parameters for the backpropagation. The influence of those changes are compared on the MNIST model.

IV. Results

A. Error Gradient Approximation

We display the results of using the precise ST-RSBP gradient expression and the less precise approximation of HM2-BP in figure 2. The plot shows test error after each training epoch, using a training set size of 10,000 MNIST images and a test set size of 10,000 images. There is a clear improvement in test error with the more precise ST-RSBP gradient calculation. The best test error rates achieved on this small training set are 8.24% for ST-RSBP and 9.20% for the old HM2-BP method. Over 20 training epochs the story is similar, with both methods clearly converging at different test errors in figure 3.

We conclude that the new gradient calculation in ST-RSBP contributes to the algorithm's test error improvements over HM2-BP.

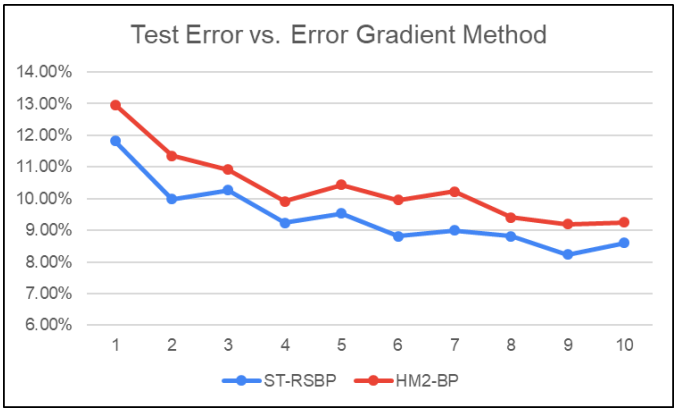


Fig. 2: Test Error over 10 training epochs using new ST-RSBP method and old approximation method

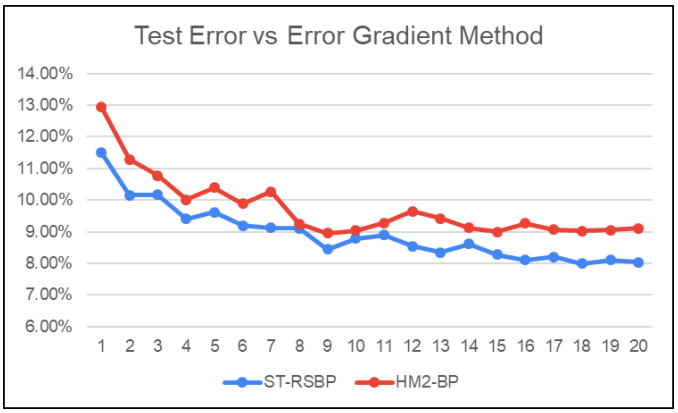


Fig. 3: Test Error over 20 training epochs using new ST-RSBP method and old approximation method

B. S-PSP Time Constant Hyperparameter Search

We evaluate the effect of the S-PSP time constant τ_m on test error by plotting test error after each training epoch, using a training set size of 10,000 MNIST images and a test set of 10,000 images. While a τ_m value of 32 caused divergence in training, the other values of 50, 64, 96, and 128 all performed adequately, as shown in figure 4. For our MNIST test, which used a smaller number of training images and epochs than the original paper, we found that a larger value for τ_m generally performed best. Due to limited computing resources and time, we did not do a more fine grain search of time constant values and could not do training on the full 60,000 images for 200 epochs. The τ_m value is a parameter which directly affects the forward and backpropagation of the SNN, and we conclude that small changes to τ_m have visible effects on the test error of the trained SNN. We are interested to learn how the authors determined their default values for these parameters.

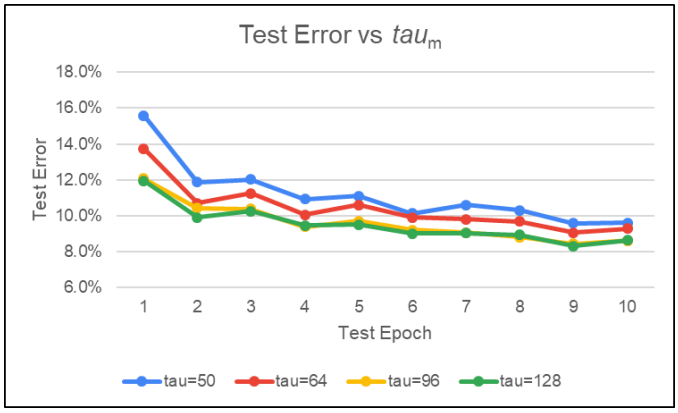


Fig. 4: Test Error over 10 training epochs using different values for τ_m

C. S-PSP Computation

1) Update Equations: A careful analysis of the MATLAB implementation revealed the parameters used were not in accordance with the mathematical equations described in the paper. The comparison is shown in Listings 1 and 2; parameters τ_m and τ_s are not correctly described in the recurrence equations and the input current is offset in the membrane potential update equation. The spike trains associated with the corrected simulation are shown in Fig. 5 for the selected parameters and show in particular an input current and membrane potential that are downscaled compared to the original results. This is further confirmed by the computation of the e_{ij} 2D interaction plot in Fig. 7a.

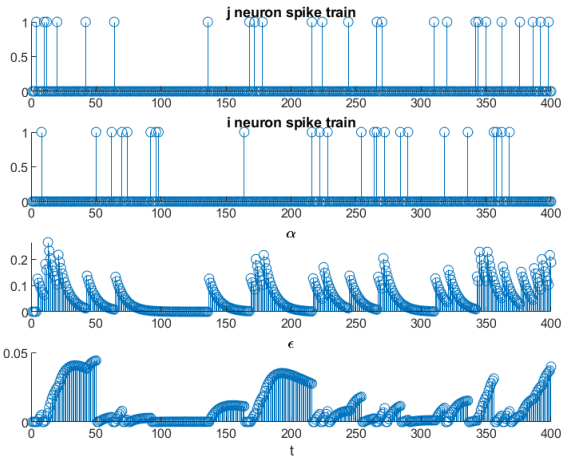


Fig. 5: Spike Level Simulation for computation e_{ij}

Since the e_{ij} values have changed, the partial derivatives in the backpropagation algorithm are scaled as well and the hyper parameter selected in the paper are no longer optimal. The backpropagation equation indicates that a way to compensate for this change is to lower the threshold voltage ν. The results on the MNIST dataset for one thousand training points are

Listing 1: Original MatLab Implementation

```
for t = 1:1:end_time
    p=p-p/TAU_S;
    if index_in<=input_num && input_time(index_in)==t-1
  p=p+1;
  index_in=index_in+1;
    end
 q = q-q/TAU_M+p/TAU_S;
 if t_ref~=0
  q=0;
  t_ref=t_ref-1;
    end
 if output_time(index_out)==t
  output = output+q;
  index_out = index_out+1;
  t_ref=T_REF;
  q=0;
    end
 if index_out>output_num
  break;
    end
end
```

Listing 2: Corrected MatLab Implementation

```
for t = 1:dt:T-1

    alpha(t+1)      = alpha(t) + dt / tau_s * (-alpha(t) + d_j(t));

    % refractory period
    if count > 0
        count           = count - dt;
        epsilon(t+1)    = 0;
        continue
    end

    epsilon(t+1)    = epsilon(t) + dt / tau_m * (-epsilon(t) + R * alpha(t));

    % if the neuron has fired
    if d_i(t+1) ~= 0
        eij(idx_j,idx_i) = eij(idx_j,idx_i) + epsilon(t+1);
        count = t_ref;
    end
end
```

shown in Fig. 6. A direct replacement leaves inconsistent results compared to the original ones, as predicted. A new threshold ν is set to 8 mV for all layers and improves accuracy but does not match the previous results. In order to further augment the values of e_{ij} to match previous values, we can reduce the spike train length. We set it to 200 ms and it achieves better performance. Due to computational resources, the comparison on the full dataset was not achievable.

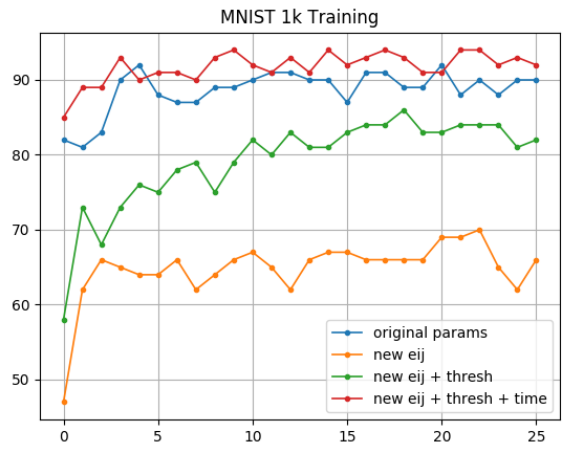
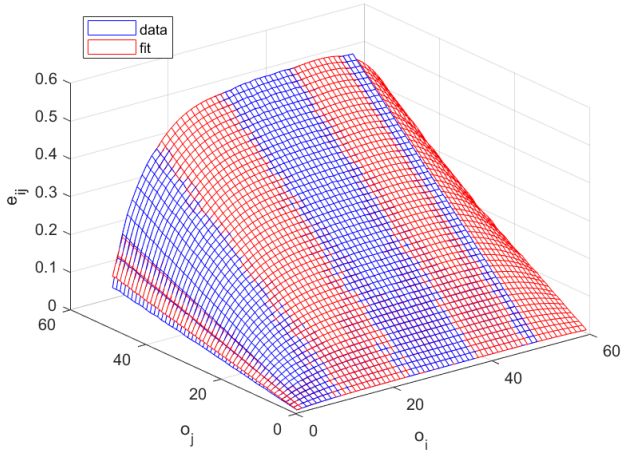
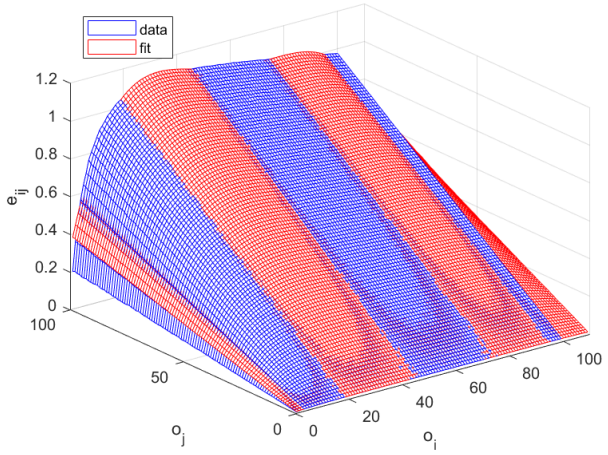


Fig. 6: Results on MNIST for the new e_{ij} approximation



(a) Polynomial fit for a 50×50 grid



(b) Polynomial fit for a 100×100 grid

2) Polynomial fitting: The investigation of the polynomial fitting revealed the approximation was giving decent results on the e_{ij} value, the polynomial degree of 4 being a good trade-off between fitting and regularization. However, while it gives decent results for a maximum firing count of 50, it worsens with a wavy behavior if the grid size is considered longer, especially for low firing rates. Since it is mainly the derivative of the approximation that is used in the backpropagation algorithm, higher deviations from the true function are observed. The CUDA implementation in fact uses saved files of polynomial fitted on a 100 by 100 grid which is less adapted, as shown in the comparison of Fig. 7a and 7b. The goal of the paper is to produce state of the art results with in general high firing rates solutions and a polynomial fit on high rates is a better choice over a polynomial extrapolation. As shown in the Figures, an approximation for points outside the simulated points are diverging because of the polynomial nature of the fit.

V. Conclusion

Our ablation study succeeded in reproducing a subset of the published results and identifying important aspects of the algorithm by removing and altering parts of the released code. We showed the sensitivity of the algorithm to the hyperparameter τ_m. By removing the precise error gradient calculation, we found support the authors' claim that their error propagation algorithm improves training performance for non-recurrent networks.

We believe a mistake has been made on the computation of the e_{ij} factor its influence can be balanced with a new set of hyper parameters because it is mainly a scaling error. The polynomial fulfills its role but is not ideal when fitting on high rate grids. A more robust approximation could be considered, by directly approximating the partial derivatives or using an other differentiable fitting like a spline or log regression, at the cost of a more complex backpropagation algorithm.

VI. Statement of Contributions

Julien did reference research, investigated the influence of the e_{ij} signal and its polynomial approximation and wrote the corresponding sections in the report. Alex modified and tested the code to examine the effects of the SNN synaptic time constant and the precise error gradient calculation. He also wrote corresponding sections of the report. Srikanth did background research and wrote part of the report.

References

[1] W. Zhang and P. Li, "Spike-train level backpropagation for training deep recurrent spiking neural networks," *ArXiv*, vol. abs/1908.06378, 2019.

[2] Y. Jin, P. Li, and W. Zhang, "Hybrid macro/micro level backpropagation for training deep spiking neural networks," *CoRR*, vol. abs/1805.07866, 2018. [Online]. Available: http://arxiv.org/abs/1805.07866

[3] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In Proceedings of the 25th international conference on Machine learning, pages 160–167. ACM, 2008.

[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998

[5] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in International Joint Conference on Neural Networks (IJCNN),. IEEE, 2015, pp. 1–8.

[6] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, big, simple neural nets for handwritten digit recognition," Neural Computation, vol. 22, no. 12, pp. 3207–3220, 2010.

[7] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, June 2014. [Online]. Available: http://goo.gl/EsQCuC

[8] Rueckauer B, Lungu I-A, Hu Y, Pfeiffer M and Liu S-C (2017), "Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification". Front. Neurosci. 11:682. doi: 10.3389/fnins.2017.00682

[9] Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., "Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. Science 345, 668–673". doi: 10.1126/science.1254642

[10] Frenkel, C., Legat, J.-d., and Bol, D. (2018). "A 0.086-mm2 12.7-pJ/SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28nm CMOS". IEEE Trans. Biomed. Circuits Syst. doi: 10.1109/TBCAS.2018.2880425.

[11] Wulfram Gerstner and Werner M Kistler. Spiking neuron models: Single neurons, populations, plasticity. Cambridge university press, 2002