Adapting unconstrained spiking neural networks to explore the effects of time discretization on network properties

The effects of time-discretization on spike-based backpropagation as opposed to membrane-potential backpropagation

Lubov Chalakova[1]

Supervisor(s): Nergis Tomen, Aurora Micheli

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Lubov Chalakova
Final project course: CSE3000 Research Project
Thesis committee: Nergis Tomen, Aurora Micheli, Lilika Markatou

Abstract

The promise of Artificial Neural Networks has lead to their immense usage intertwined with concerns over energy consumption. This has led to development of alternatives, such as Spiking Neural Networks (SNNs), which allows their implementation on neuromorphic hardware. In effect, the network must be time discretized. SNNs learn through backpropagation, similarly to ANNs, where two main methods exist: spike-based backpropagation and backpropagation through time. This study investigates and compares the effect of varying timesteps on the convergence rate of BATS, an example of spike-based backpropagation, and of SLAYER - an example of backpropagation through time on the N-MNIST and MNIST dataset. The results conclude that BATS withstands growing timestep sizes. Although SLAYER maintains a good performance for small timestep sizes, it seems to self-destruct as they grow.

1 Introduction

Artificial Neural Networks have gained immense popularity due to their state of the art performance and multitude of application domains. Concerns over high energy consumption [8], however, drive the development of efficient alternatives, one of which are Spiking Neural Networks (SNNs). SNNs are a biologically plausible neural network models which enhance energy efficiency by transmitting information through binary spikes as opposed to analogue activation, thus enabling efficient deployment of neural networks on neuromorphic hardware [2], such as SpiNNaker and Loihi [4].

The state of spiking neurons is represented by membrane potential and is modeled by differential equations dependent on time and incoming spikes. These differential equations are difficult to solve analytically, hence a numerical method through a sequence of time steps is needed. This approach, however, affects the accuracy of the model [13].

Recent studies analyze architectural choices and models of SNNs and their effect on training. An important aspect is backpropagation, which enables the network to learn from errors. Dampfhoffer et al. has successfully compiled a survey comparing backpropagation techniques [3], where three main categories of backpropagation mechanisms can be identified. ANN-to-SNN mechanisms involve first training an ANN through a respective backpropagation mechanisms and then transferring the weights to an SNN model. In spike-based backpropagation mechanisms, learning occurs only when there is a spike in the output layer. Lastly, in BackPropagation Through Time (BPTT) mechanisms, errors are backpropagated on every timestep. The survey, however, alike a large majority of other studies, investigates constrained SNNs, where neurons may fire at most once. Unconstrained SNNs, which allow neurons to fire multiple times, are rarely explored, despite evidence suggesting a higher convergence rate [11].

Therefore, a research gap can be identified in the study of backpropagation of unconstrained SNNs. Thus, this paper compares the effect of time-discretization on spike-based mechanisms, as introduced in the BATS model [1], and a Backpropagation Through Time mechanism, as implemented in SLAYER [12].

Thus this paper answers the research question **What is the effect of time discretization on spike-based backpropagation as opposed to backpropagation through time?** The following hypothesis was formulated in response: *The convergence rate will increase in both models as the timestep size grows, but the increase will be more noticeable in spike-based models as they depend on exact spiking times.*

The hypothesis is tested by comparing the effects of varying time-step sizes on BATS and SLAYER. While both enable the training of SNNs with continuous time dynamics, they differ in their specific methodologies. BATS emphasizes the temporal precision of spike timing, whereas SLAYER introduces error reassignment to enable backpropagation through spikes. Both models are modified to allow the simulation of different timestep sizes and the propagation mechanisms are adjusted accordingly as described in section 3 and 2. The difference in training convergence rate is derived for different time-steps on the N-MNIST dataset. Further details on the experimental set-up and results can be found in section 4. Subsequently, a comparative analysis is conducted of the effects of different time-steps on the convergence rate of both models as discussed in section 5.

2 Preliminaries

2.1 Introducing the CuBa-LIF neuron

Defining The CuBa-LIF Neuron

Spiking neurons are the building units of a Spiking neural network. This paper develops its analysis on the Current-Based Leaky Integrate and Fire Neuron (CuBa-LIF) [6]. It is chosen as it is a common building block of both SLAYER and BATS and can be easily adapted to various network architectures and learning algorithms.

The CUBA LIF neuron model is characterized by two values: the neuron current and the neuron membrane potential, both of which decrease exponentially over time until they reach a steady state. Unlike other neuron models where the membrane potential experiences a discontinuous jump when a spike is received, the CUBA LIF model raises the membrane potential in a continuous manner. This smooth transition allows for more accurate representation and analysis at varying temporal resolutions, ensuring that the behavior of the neurons remains consistent across different timesteps.

The membrane potential u and post-synaptic current I for neuron j on layer l are described in Equations 1 and 2 [1]:

$$\frac{\mathrm{d}I^{(l,j)}}{\mathrm{d}t} = -\frac{1}{\tau_s} I^{(l,j)}(t) + \sum_{i=1}^{N^{(l-1)}} w_{i,j}^{(l)} \sum_{z=1}^{n^{(l-1,i)}} \delta(t - t_z^{(l-1,i)}) \tag{1}$$

$$\frac{\mathrm{d}u^{(l,j)}}{\mathrm{d}t} = -\frac{1}{\tau}u^{(l,j)}(t) + I^{(l,j)}(t) - \theta\delta(u^{(l,j)}(t) - \theta) \quad (2)$$

where

$$\delta(x) = \begin{cases} +\infty & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases}$$

is the Dirac delta function.

Equation 1 models how the post-synaptic current I decays exponentially over time and gets instantaneously increased by incoming spikes from connected neurons. τ_s, which is the synaptic time constant, denoting the factor of decay for the post-synaptic current. An increment is caused by a received spike at synapse i of a neuron j by an amount $w^{(l)}_{(i,j)}$, which denotes the strength of the synaptic connection. The equation updates I based on each synaptic connection of the current neuron j and based on the input spikes receives from each synapse. $N^{(l)}$ denotes the number of neurons on layer l and $n^{(l,j)}$ denotes the number of spikes fired by neuron j of layer l. The k^{th} spike fired by a neuron is denoted by $t^{(l,j)}_k$.

Equation 2 describes how the membrane potential u evolves over time, decays, and responds to input currents, incorporating the neuron's firing mechanism. u decays over time with a factor based on τ, the membrane time constant. The potential increases every time an input spike is received as determined bu the post-synaptic current. If the membrane potential reaches a threshold value θ, the neuron fires and the membrane potential undergoes a soft reset.

SRM Form of the CuBa-LIF Neuron

The Spike Response Model (SRM) is a generalization of the Leaky Integrate and Fire model where parameters depend on the time passed from the last output spike produced by the neuron. This form is used for analysis and derivation of gradients due to its explicit dependence on time and the separation of the effect of input spikes (through kernel ϵ) and of the neuron's own firing (through kernel η)[5]. It is introduced here for the sake of better understanding the derivation of both backward mechanisms, as well as the steps taken for their discretization.

The SRM form of a neuron's membrane potential is defined in Equation 3:

$$u^{(l,j)}(t) = \sum_{i=1}^{N^{(l-1)}} w^{(l)}_{i,j} \sum_{z=1}^{n^{(l-1,i)}} \epsilon(t - t^{l-1,i}_z) - \sum_{z=1}^{n^{(l,j)}} \eta(t - t^{(l,j)}_z) \quad (3)$$

As outlined by [1], the CuBa-LIF neuron can be expressed in SRM form by defining ϵ and η as Equations 4 and 5 respectively:

$$\epsilon(t) = \Theta(t)\frac{\tau\tau_s}{\tau - \tau_s}\left[\exp\left(\frac{-t}{\tau}\right) - \exp\left(\frac{-t}{\tau_s}\right)\right] \quad (4)$$

$$\eta(t) = \Theta(t)\theta\exp\left(\frac{-t}{\tau}\right) \quad (5)$$

where $\Theta(t)$ is the Heaviside step-function. These definitions are derived by equating Equations 2 and 3.

2.2 The SLAYER Model: BackPropagation Through Time

Spike LAYer Error Reassignment (SLAYER) [12] is a general method for backpropagation employing temporal error credit assignment, where credit is distributed back through layers and through time. The weights are updates on every timestep based on the spike events observed up to that point.

SLAYER is implemented using the Time-discretized CuBa-LIF neuron introduced in section 3.2. The backpropagation mechanisms is derived from the SRM Model of the CuBa-LIF neuron as expressed in equation 3. The gradient term in layer l for weight between neuron i in layer l and neuron j in layer $l+1$ is defined with a sampling time T as follows:

$$\frac{\partial E}{\partial w^{(l)}_{i,j}} = T\sum_{m=0}^{N_s} a^{(l,i)}[m] \sum_{n=m}^{N_s} \frac{\partial L[n]}{\partial u^{(l+1,j)}[m]} \quad (6)$$

where

$$a^{(l,i)}[t] = \epsilon\Big(\sum_{z=1}^{n^{(l-1,i)}} \delta(t - t^{(l-1,i)}_z)\Big) \quad (7)$$

and E is the loss function calculated over the interval $[0, T]$, whereas $L[t]$ is the loss at time instance t. N_s denotes the total number of samples in the period $[0, T]$. ϵ is defined in Equation 4.

Although not explicitly derived in the paper, the SLAYER change of weight is performed using the ADAM gradient descent [7].

2.3 The BATS Model: Spike-driven Backpropagation

The forward propagation of the BATS model is defined based on the continuous CuBa-LIF This paper discretized the model as described in section 3.2.

The BATS model generalizes the approach taken by Fast&Deep by allowing neurons to fire multiple times and constructing a backpropagation mechanism accordingly [1]. The equations are derived from the SRM form of a CuBa-LIF neuron as defined in Equation 3. By setting $\tau = 2\tau_s$, a closed form solution can be derived of the form

$$0 = -a^{(l,j)}_k \exp\left(-\frac{t^{(l,j)}_k}{\tau}\right)^2 + b^{(l,j)}_k \exp\left(-\frac{t^{(l,j)}_k}{\tau}\right) - c^{(l,j)}_k \quad (8)$$

where

$$a^{(l,j)}_k := \sum_{i=1}^{N(l-1)} w^{(l)}_{i,j} \sum_{z=1}^{n(l-1,i)} \left(t^{(l,j)}_k - t^{(l-1,i)}_z\right) \exp\left(\frac{t^{(l-1,i)}_z}{\tau_s}\right)$$

$$b^{(l,j)}_k := \sum_{i=1}^{N(l-1)} w^{(l)}_{i,j} \sum_{z=1}^{n(l-1,i)} \left(t^{(l,j)}_k - t^{(l-1,i)}_z\right) \exp\left(\frac{t^{(l-1,i)}_z}{\tau}\right) - \frac{\vartheta}{\tau} \sum_{z=1}^{n^{(l,j)}} \left(t^{(l,j)}_k - t^{(l,j)}_z\right) \exp\left(\frac{t^{(l,j)}_z}{\tau}\right)$$

$$c := c_k^{(l,j)} = \frac{\vartheta}{\tau}.$$

Equation 8 yields the following solution:

$$t_k^{(l,j)} = \tau \ln \left(\frac{2a_k^{(l,j)}}{b_k^{(l,j)}} + x_k^{(l,j)} \right) \tag{9}$$

where $x_k^{(l,j)} = \frac{-b_k^{(l,j)} \pm \sqrt{b_{k2}^{(l,j)} - 4a_k^{(l,j)} c_k^{(l,j)}}}{2a_k^{(l,j)}}$. Thus, it is possible to infer the neuron spike trains in an event based manner.

Finally, BATS defines the change of weight $\Delta w_{i,j}^{(l)}$ as the sum of all errors $\delta_k^{(l,j)}$ applied to their corresponding spike k derivatives:

$$\Delta w_{i,j}^{(l)} = \sum_{k=1}^{n^{(l,j)}} \delta_k^{(l,j)} \frac{\partial t_k^{l,j}}{\partial w_{i,j}^{(l)}}$$

where

$$\frac{\partial t_k^{l,j}}{\partial w_{ij}^{(l)}} = \sum_{z=1}^{n_i^{(l-1)}} \Theta(t_k^{(l,j)} - t_z^{(l-1,i)}) \left[f_k^{(l,j)} \exp\left(\frac{t_z^{(l-1,i)}}{\tau_s} \right) - h_k^{(l,j)} \exp\left(\frac{t_z^{(l-1,i)}}{\tau} \right) \right]$$

and

$$f_k^{(l,j)} := \frac{\partial t_{(l,j)k}}{\partial a_{(l,j)k}} = \frac{\tau}{a_k^{(l,j)}} \left[1 + \frac{c}{x_k^{(l,j)}} \exp\left(\frac{t_k^{(l,j)}}{\tau} \right) \right],$$

$$h_k^{(l,j)} := \frac{\partial t_k^{(l,j)}}{\partial b_k^{(l,j)}} = \frac{\tau}{x_k^{(l,j)}}$$

3 Methodology

This section introduces the model used to perform the experiment and outlines the steps taken to apply time discretization on the CuBa-LIF neuron and adapt the SLAYER and BATS backpropagation mechanisms accordingly.

3.1 Defining The Timestep

This paper analyzes the effect of timestep size on network properties. However, in different models, the implementation of time can vary.

SLAYER is a time-driven model, which means the model works at clock intervals. The time is counted in terms of algorithmic timesteps, where one timestep refers to one iteration where the neurons are updated and the spikes are checked. Therefore, the timestep size represents the number of iterations passed between network updates.

BATS, on the other hand, is event-driven. The network is updated whenever a spike event is registered in the output layer at a certain time measured in seconds. Therefore, the timestep size represents the number of seconds passed between network updates.

Such discrepancy in definition and implementation must be accounted for in order to draw meaningful comparisons between the two models. One approach is to use timebins. Time bins discretize time into consistent intervals, such as milliseconds, regardless of whether the models operate in algorithmic timesteps or real-time seconds.

This paper derives equations in terms of timestep size. In terms of implementation, however, the timestep size is determined based on the desired number of timebins for a given batch. The timestep size for BATS is found as so:

$$\text{timestep size} = \frac{\text{simulation time}}{\text{\#time bins}}$$

where simulation time denotes the total total duration over which the neural network is simulated.

The timestep size for SLAYER can be calculated as so:

$$\text{timestep size} = \frac{\text{sample length}}{\text{\#time bins}}$$

where sample length is the duration for which data is sampled and binned into. This directly affects the number of iterations as each iteration evaluates one timebin. Thus, it has a notion similar to the simulation time.

3.2 The Time-Discretized CuBa-LIF Neuron

To simulate the functions of continuous models on hardware operating on discrete time intervals, approximation in discrete timesteps is necessary. The Lava software framework[1], which contains the implementation of the SLAYER architecture, defines a time-discretized version of the CuBa-LIF neuron based on Loihi [4] according to the following equations:

$$I^{(l,j)}[t] = (1-\alpha) I^{(l,j)}[t-1] + \sum_{i}^{N^{(l-1)}} w_{i,j}^{(l)} s[t] \tag{10}$$

$$u^{(l,j)}[t] = (1-\beta) u^{(l,j)}[t-1] + I^{(l,j)}[t] + bias \tag{11}$$

where $s[t]$ is a vector representing the binary spike inputs from all pre-synaptic neurons at time t. α and β are decay constants which can be substituted by the membrane and synaptic time constants $-\frac{1}{\tau}$ and $-\frac{1}{\tau_s}$ respectively to better model equations 2 and 1. Additionally, the model considered uses no bias, so the term can be omitted.

The membrane potential is reset whenever the neuron emits a spike according to the following equation:

$$u^{(l,j)}[t] = u^{(l,j)}[t](1 - s_o^{(l,j)}[t]) \tag{12}$$

where $s_o[t]$ is a binary function defining whether the neuron has fired at time t or not. The equation for membrane potential used in this experiment embeds soft reset by redefining $u[t]$ as so:

$$u^{(l,j)}[t] = (1 - \frac{1}{\tau}) u^{(l,j)}[t-1] + I^{(l,j)}[t] - \theta S[t] \tag{13}$$

where

$$S[t] = \Theta(u^{(l,j)}[t] - \theta) \tag{14}$$

[1]https://github.com/lava-nc/lava-dl

and Θ is the Heaviside side-step function. This omits the need for equation 12.

Additionally, the experiment strives to determine the effect of timestep size on the network. Therefore, timestep size Δt is embedded as so:

$$I^{(l,j)}[t] = (1 - \frac{\Delta t}{\tau_s})I^{(l,j)}[t - \Delta t] + \sum_{i}^{N^{(l-1)}} w_{i,j}^{(l)} s[t] \quad (15)$$

$$u^{(l,j)}[t] = (1 - \frac{\Delta t}{\tau})u^{(l,j)}[t - \Delta t] + I^{(l,j)}[t] - \theta S[t] \quad (16)$$

Lastly, the neuron emits a spike when the following condition is satisfied:

$$u^{(l,j)}[t] > \theta \quad (17)$$

This implementation is introduced in both the BATS and SLAYER models.

3.3 The Time-Discretized SLAYER Model

SLAYER is a clock-driven method, meaning the model is updated at every algorithmic timestep. In the Lava-dl implementation of SLAYER, the default size of each algorithmic timestep is 1. This means that one timestep corresponds to the time required to update every neuron in the system consecutively. In SLAYER and other time-driven systems, the timestep size refers to the number of algorithmic timestepes defined withing an interval.

The code has been modified to allow for tuning the timestep size Δt as illustrated by Algorithm 1. Whereas the original algorithm calculates a gradient for every timestep per neuron, the modified version calculates the gradient only at the correct timestep intervals. To maintain code stability, on timesteps indivisible by the desired timestep size, the gradient value of the previous timestep is passed along.

It can be noted that this backpropagation mechanism functions on the basis of timesteps. A change in timestep of size Δt will affect the total number of samples N_s, thus no further derivation is required.

The backpropagation method and the implementation of the CuBa-LIF neuron has been implemented according to the equations in section3.2 [2].

3.4 The Time-Discretized BATS Model

In an event-driven model, such as BATS, the output layer will be checked for spikes in intervals of the stepsize Δt. Thus, a spike will be detected within a time $d_k^{l,j}$ after it has fired. Therefore, if a spike occurs at time $t_k^{l,j}$, then the discretized spike-time $t_{\sim k}^{(l,j)}$ can be expressed as so:

$$t_{\sim k}^{(l,j)} = t_k^{(l,j)} + d_k^{(l,j)} \quad (18)$$

where

$$d_k^{(l,j)} = \Delta t - \mod(t_k^{(l,j)}, \Delta t) \quad (19)$$

The new spike-time calculation must be taken into account when calculating the gradient used during backpropagation. This paper utilizes a straight-through estimator (STE) [9], which propagates the same gradient from the output to the input of the neuron. Thus, although the values used during forward propagation are the discretized spikes $t_{\sim k}^{(l,j)}$, the gradient considered during backpropagation remains $\frac{\partial t_k^{l,j}}{\partial w_{ij}^{(l)}}$.

[2]http://repository.tudelft.nl/

Algorithm 1 The modified SLAYER model.

```
1: procedure TRAIN-ORIGINAL($data, dt$)
2:     data ← data in batches
3:     for batch in data do
4:         Forward-Propagation(batch,dt)
5:         Calculate loss for batch
6:         Backward-Propagation(batch,dt)
7:     end for
8: end procedure
9:
10: procedure FORWARD-PROPAGATION($batch, dt$)
11:     for neuron in network do    ▷ $network$ is an attribute
12:         for timestep in timesteps passed do
13:             if timestep $\mod dt \neq 0$ then
14:                 record old neuron state
15:                 Do not check for spiking
16:             end if
17:             Update neuron state
18:             Check if neuron is spiking
19:         end for
20:     end for
21: end procedure
22:
23: procedure BACKWARD-PROPAGATION($batch, dt$)
24:     for each time step $n$ from last to first do
25:         if timestep $\mod dt \neq 0$ then
26:             Calculate gradient for timestep
27:         else
28:             Carry over previous gradient
29:         end if
30:     end for
31: end procedure
```

3.5 Dataset

MNIST

The MNIST dataset is a frequently seen benchmark in neural network evaluations. It contains images of handwritten digits split into a training set of 60,000 examples, and a test set of 10,000 examples. The images have shape 28x28.

This training set is used to evaluate the time-discretized BATS algorithm. This decision was made as BATS has been implemented to work well on static image data.

N-MNIST

N-MNIST [10] is a widely used dataset in SNN evaluation and study methods. It is a spiking version of the original MNIST dataset, also consisting of 60 000 training and 10 000 testing samples captured at a scale 28x28 pixels. The original static MNIST images were presented to a neuromorphic vision sensor that generated a stream of spikes in response to the visual stimuli.

The N-MNIST dataset is used to train and evaluate the discretized SLAYER model. Using N-MNIST allows SLAYER to leverage its strengths in processing and learning from spike-based temporal data. Additionally, N-MNIST provides a direct exploration of temporal dynamics due to its inherent temporal structure.

An attempt was made to train BATS on this dataset. However, changes beyond the scope of the project were required to make the necessary changes.

3.6 Convergence Rate Metric

Convergence Rate (CR) is a metric which illustrates the speed with which the model increases its accuracy and learns. It is derived by calculating the area under the curve of an accuracy vs. epochs graph. A higher convergence rate denotes that the network is learning more efficiently.

This paper looks at the convergence rate achieved during training. The area under the curve is calculated using the Python function `trapz` from the `scipy` library.

In this paper, the convergence rate is calculated for the non-discretized versions of the models. This is be considered as the benchmark convergence rate. Next, the convergence rate data is derived for each timestep size - these are the data points. In order to measure the change in convergence rate caused by the timestep size, the benchmark is subtracted from each data point and then normalized as so:

$$\text{change in CR} = \frac{(\text{data point CR}) - (\text{benchmark CR})}{(\text{benchmark CR})}$$

4 Experimental Setup and Results

4.1 Experimental Set-up

The experiment is carried out on a Lenovo Yoga Creator 7 laptop with 16GB Ram, 2.60GHz Intel(R) Core(TM) i7 processor, NVIDIA GeForce GTX 1650 GPU and Windows 11. The Lava-dl SLAYER Python library[3] and the bats repository[4] were used as a basis for implementation.

Model Set-up

The two networks consist of an input layer, one hidden layer and an output layer. The input layer of SLAYER has 34*34*2 neurons, which is the input size of the N-MNIST dataset on which it is trained. The input layer of BATS consists of 28x28 neurons, which is the input size of the MNIST dataset. Both networks have 512 neurons in the hidden layer and 10 output neurons. Weight normalization is applied on all layers and synaptic delays are included.

SLAYER uses the Spike Rate Loss function, whereas BATS uses the Spike Count Class Loss function. Out of the loss function options in both models, these two loss functions were most suitable for the task and had the greatest overlap. Both models employ ADAM gradient descent.

The two models were instantiated with equal parameters where possible. However, changes were made in values such as thresholds and time constants for the sake of enabling the network to train. The parameter values differentiate in cases when setting them as equal prevented the non-discretized benchmark model from training. For further information about the parameters refer to Table 1.

Model	Parameter Name	Parameter Value
Both	Batch size	32
	Output threshold θ	1.25
	membrane time constantτ	0.26
	ADAM learning rate	0.001
	Epochs	15
SLAYER	Hidden threshold θ	1.25
	Synaptic time constant τ_s	0.25
	tau_grad	0.03
	Scale_grad	3
	Loss - true rate	0.2
	Loss - false rate	0.03
	Sample time	300
BATS	Spike Buffer Size	30
	Hidden threshold θ	0.2
	Synaptic time constant τ_s	0.13
	Learning rate decay epoch	10
	Learning rate decay factor	1.0
	Loss function False Target	3
	Loss function False Target	15
	Dataset time window	0.1
	Dataset max value	255

Table 1: The hyper-parameters of the models.

Experiment Variables

The convergence rate of each model was derived for 5 different timebin sizes. The model was ran three times per timebin size. The tested timebin sizes and their corresponding timestep sizes can be found in table **??**.

Timebin size	BATS timestep Δt	SLAYER timestep Δt
150	0.002	2.0
75	0.004	4.0
50	0.006	6.0
37	0.008	8.0
30	0.010	10.0

Table 2: The independent variables of the experiment. The timebin sizes selected for the experiments and the corresponding timestep size for each model.

4.2 Results

The experiment has been ran on SLAYER and BATS.

Time bin	SLAYER Avg. CRC	BATS Avg. CRC
150	-0.0067	0.4386
75	-0.0860	0.1477
50	-0.1924	- 0.0027
37	-0.7071	-0.0041
30	-0.8970	-0.0605

Table 3: The change in convergence rate (CRC) for each of the models for each timebin.

[3]https://github.com/lava-nc/lava-dl

[4]https://github.com/Florian-BACHO/bats

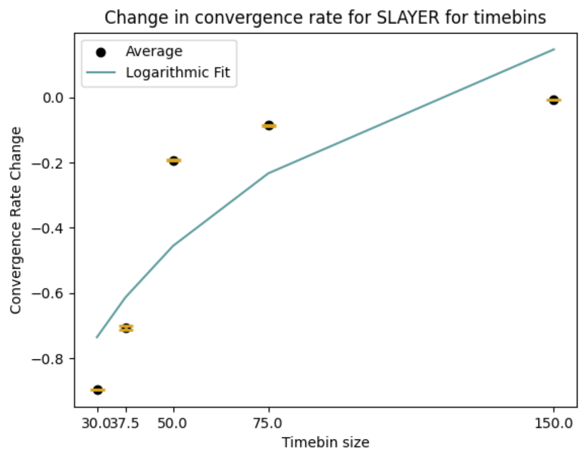

Figure 1: A plot of change in convergence rate vs time bin sizes for the SLAYER algorithm.

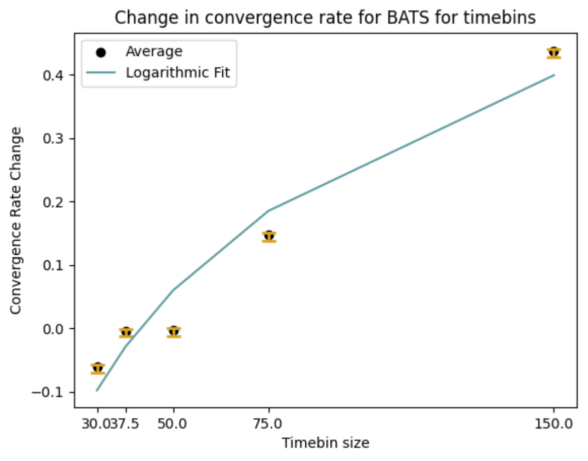

Figure 2: A plot of change in convergence rate vs time bin sizes for the BATS algorithm.

5 Discussion

This section discusses the results drawn from the experiment.

5.1 SLAYER

The result for SLAYER shown in 1 demonstrate that change in convergence rate decreases logarithmically as the number of time bins grows. This indicates that the more timebins there are, i.e. the more refined the time is, the closer the performance to the baseline. This makes sense since a larger number of timebins means that the timestep is smaller. A smaller timestep size, in effect, is a more closer approximation to continuous time.

Furthermore, it must be noted that 50 or more timebins (meaning timestep sizes less 6) are very close to the benchmark performance. Afterwards there is a sudden drop to a low value which indicates that the network achieves minimal performance and no longer learns. This could be explained by the fact that a lot of information is not taken into consideration when the timestep size is so large.

Lastly, it can be noticed that the convergence rate change is negative for all data points. Hence, the time discretized model is always slower to converge than the benchmark continuous model.

5.2 BATS

The result for BATS shown n 2 demonstrate an interesting dynamic. When there are more than 75 timebins, the model appears to converge faster than the baseline model with a significant difference. This contradicts the initial hypothesis of this paper. A possible explanation is that time discretization in for the MNIST dataset filters some noise present in the data. Another possibility is that for low timestep values, the spike time approximation is accurate.

For less than 75 timebins, however, the change in convergence rate become negative and the model learns slower than the baseline. It can be explained by the less accurate approximation of continuous time where the approximated gradient no longer accurately represents the true gradient. Regardless, the change in convergence rate remains minimal.

5.3 Comparing BATS and SLAYER

The results lead to a conclusion opposing the original hypothesis. Whereas it was initially assumed that SLAYER would demonstrate a less drastic change in convergence rate due to its implementation based on timesteps, the results show that the opposite holds. Moreover, BATS remains close to the benchmark performance even for very large timestep sizes, whereas SLAYER seems to stop learning. Overall, BATS seems to be more resilient to time-discretization than SLAYER even for a small amount of timebins. BATS further indicates a possible improvement in performance.

6 Responsible Research

To ensure reproducibility of the methods, the selected model and its configuration have been explained in detail with additional details regarding the system it was run on. The modified code is publicly available and the dataset is open-source.

To ensure ethical research, all sources have been properly cited and for a given piece of information, the original source has been examined and referenced.

7 Conclusions and Future Work

This paper investigated and compared the effect of varying timestep sizes on the change in convergence rate of Spiking Neural Networks employing event-driven backpropagation as opposed to backpropagation through time. The models SLAYER and BATS where used to represent a spike-based model and a time-driven model respectively. These models were time-discretized and trained on N-MNIST and MNIST respectively where hyperparameters were aligned as much as possible. The following conclusions can be drawn from the results:

- The SLAYER model maintains a convergence rate close to the benchmark for small timestep sizes. However, once the timestep size grows larger than 6, the model appears to self-destruct.
- The BATS model maintains a close of better learning rate in comparison to the benchmark performance. This disproves the initial hypothesis which assumed the opposite.

7.1 Further research

This investigation can be extend in the following ways:

- **More backpropagation mechanisms.** This study tested one example of each back-propagation technique - SLAYER for backpropagation through time and BATS for spike-based backpropagation. Other techniques can be implemented and compared to ascertain a stronger relation between the effect of timestep size and the backpropagation model.
- **More datasets.** More datasets can be used to train and evaluate these models. Efforts can be made to input the same dataset to both models.
- **More similar architecture.** Although attempts were made to keep the architecture of the models as close together as possible, it must be noted that there are many differences, which could have caused the drstic difference in performance of the two models.

References

[1] Florian Bacho and Dominique Chu. Exploring Trade-Offs in Spiking Neural Networks. *Neural Computation*, 35(10):1627–1656, 09 2023.

[2] Manon Dampfhoffer, Thomas Mesquida, Alexandre Valentian, and Lorena Anghel. Are snns really more energy-efficient than anns? an in-depth hardware-aware study. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 7(3):731–741, 2023.

[3] Manon Dampfhoffer, Thomas Mesquida, Alexandre Valentian, and Lorena Anghel. Backpropagation-based learning techniques for deep spiking neural networks: a survey. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–16, 2023. hal-04064177.

[4] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[5] Wulfram Gerstner and Werner M. Kistler. *Formal spiking neuron models*, page 93–146. Cambridge University Press, 2002.

[6] Kriener L. Baumbach A. et al. Goltz, J. Fast and energy-efficient neuromorphic deep learning with first-spike times. *Nat Mach Intell 3*, page 823–835, 2021.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[8] Da Li, Xinbo Chen, Michela Becchi, and Ziliang Zong. Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus. In *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, pages 477–484, 2016.

[9] Z. Liu, K-T. Cheng, D. Huang, E. P. Xing, and Z. Shen. Nonuniform-to-uniform quantization: Towards accurate quantization via generalized straight-through estimation. pages 4942–4952, 2022.

[10] Garrick Orchard, Gregory Cohen, Ajinkya Jayawant, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, 9(437), October 2015. Open Access.

[11] Michael Pfeiffer and Thomas Pfeil. Deep learning with spiking neurons: Opportunities and challenges. *Frontiers in neuroscience*, 12:774, 2018.

[12] Sumit Bam Shrestha and Garrick Orchard. Slayer: Spike layer error reassignment in time. 2018.

[13] Sergio Valadez-Godínez, Humberto Sossa, and Raúl Santiago-Montero. The step size impact on the computational cost of spiking neuron simulation. In *2017 Computing Conference*, pages 722–728, 2017.