Backpropagating in time-discretized multi-spike spiking neural networks

How are the training accuracy and training speed (in epochs and time) of a spiking neural network affected when numerically integrating with the forward-Euler and Parker-Sochacki methods?

Max Guichard

Supervisor(s): Nergis Tömen, Aurora Micheli, Olaf Booij

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: Max Guichard
Final project course: CSE3000 Research Project
Thesis committee: Nergis Tömen, Aurora Micheli, Evangelia Anna Markatou

Abstract

Spiking neural networks have gained traction as both a tool for neuroscience research and a new frontier in machine learning. A plethora of neuroscience literature exists exploring the realistic simulation of neurons, with complex models requiring the formulation and integration of ordinary differential equations. Overcoming this challenge has led to the exploration of various numerical integration techniques with the goal of highly stable and accurate simulations. In contrast, training spiking neural networks is often done with simple leaky integrate-and-fire models and rudimentary integration methods such as the forward-Euler method. In this research we explore how more complex numerical integration methods, borrowed from neuroscience research, affect the training of networks based on current-based leaky integrate-and-fire neurons. We derive equations required for the integration process and suggest the use of spike time interpolation. Furthermore, we provide insights into applying backpropagation on numerically integrated networks and highlight possible pitfalls of the process. We conclude that numerically integrated networks can achieve training accuracies close to their theoretical limits, with good convergence and training time characteristics. Specifically, high order integrations achieve robust and computationally viable training. Additionally, we explore the effects of spike time interpolation on network accuracy and use our findings to provide insights into the role of different integration parameters on the effective training of spiking neural networks.

1 Introduction

Spiking neural networks have gained traction as a tool for neuroscience research and a new frontier in machine learning. Neuroscience research using spiking neural networks focuses primarily on exploring biologically observed phenomena such as criticality in the visual cortex [1]. Machine learning research, on the other hand, focuses on training procedures for spiking neural networks and how to transfer these networks into hardware [2] [3]. Hardware implementations in specific, such as [4], offer high energy efficiency for inference tasks with spiking neural networks but are limited to simulating neurons with discrete time steps.

The neuroscience literature offers a variety of neuron models for simulating networks, such as the leaky integrate-and-fire, Izhikevich, and Hodgkin-Huxley models. Each model, in turn, provides a higher degree of biophysical accuracy at greater computational costs, leading to more complex dynamics such as bursting, mixed mode firing patters and frequency adapted spiking [5]. These models are described in terms of ordinary differential equations, requiring numerical integration schemes to simulate the neuron dynamics. To this end, a plethora of techniques have been borrowed from physics and applied to these neural models with adaptations for discontinuities in their dynamics . On the other hand, the machine learning literature explores spiking neural networks with a limited selection of neuron models, primarily leaky integrate-and-fire derivatives, with the primary integration method being adaptations of forward-Euler [6].

We aim to close the gap between the two sides of the literature by exploring: *How are the training accuracy and training speed (in epochs and time) of a spiking neural network affected when numerically integrating with the forward-Euler and Parker-Sochacki methods?* To that end, we adapt the neuron model introduced in [2], a current-based leaky integrate-and-fire neuron, and apply spike time based backpropagation. This model is chosen since it is similar to models in the training literature, as mentioned above, and can easily be integrated analytically and using a variety of techniques from the simulation literature. We stick to spike time based backpropagation due to the analytical solution available in [2] for our neuron model. It is hypothesized that higher-order integrations will converge quicker, to higher accuracy solutions, due to their inherently lower numerical errors. Furthermore, due to the found importance of spike timing in biological networks, we hypothesize that spike time interpolation will assists in the learning process, as further discussed in section 3.5. To explore this last point, the following question is also addressed: *How accurate is a numerically simulated spiking neural network with spike time interpolation compared to its analytical counterpart with the same weights when tested on MNIST?*

We contribute an analysis of the current-based leaky integrate-and-fire neuron model under the forward-Euler and Parker-Sochacki integration methods, as well as how to apply spike time interpolation to improve simulation fidelity. We then derive effective backpropagation equations to train the numerically integrated networks. We conclude that with proper parameter choices, numerically simulated networks perform on par with their analytical counterpart, with higher order integrations generally performing better, resulting in similar accuracies and training convergence. Spike time interpolation makes training more lenient as it allows accurate integrations over a wide range of parameters through its time step adapations.

Further discussion of related works is discussed in section 2, giving an overview of the relevant literature. Background for this research is introduced in section 3, which outlines the neuron and network model used in the rest of the paper, as well as the numerical integration techniques explored. Section 4 expands on the methodology of the exploration, including the equations for backpropagation. Section 5 discusses the results obtained from training using the techniques introduced in section 3, on the MNIST dataset. Section 6 will discuss and compare the results in section 5, and highlights limitations of our research. Section 7 will summarize the findings of this paper and suggest avenues for further research into training spiking neural networks. Finally section 8 reflects on the ethics of the results and production of this paper.

2 Related Works

The study of numerical methods for solving ordinary differential equations offers a variety of algorithms, broadly categorized into explicit and implicit methods. Explicit techniques, which include the forward-Euler method, Runge-Kutta methods and the Parker-Sochacki methods, are common place for neuron simulations [7; 8] - [7] compares the errors introduced in a Izhikevich neuron model in fixed-step simulation with the forward-Euler, Runge-Kutta 2 and Runge-Kutta 4 methods, finding that the errors were similar in form, but became lower as the order of integration increased. [9] looks at trainig leaky integrate-and-fire, Izhikevich, FitzHugh-Nagumo and Hodgkin-Huxley neurons for simple function regression, with one network layer, using the forward-Euler and Runge-Kutta 4 methods. Training was done through automatic differentiation. Similarly to [7], they find increases in overall simulation accuracy with higher order explicit integrators and observe that more biologically plausible neuron models produce better predictions, with lower spike counts, at high computational costs. Given the prominence of the leaky integrate-and-fire model in the training literature, and its more manageable computational costs, it will be our focus.

Implicit integration algorithms such as the Backward Differential Formula family of methods find use in simulating complex neuron systems, where they provide better stability guarantees than explicit methods [10]. This follows from the stiffness of neurons' dynamics, where explicit methods may require prohibitively small time steps in order to accurately capture transiently large gradients induced by events (such as the input or output of a spike). Implicit solvers solve implicit functions at every simulation step, which don't often afford closed-form solutions. Backpropagating on their outputs generally requires more advanced analysis, discussed in section 7.1, or generating large computational graphs which demand higher compute resources. Is is primarily for this reason that we explore explicit methods.

Specialized integration techniques adapted to neuron behaviour have also been proposed, principally quantized state system methods which handle transient behaviours well at low computational costs [11; 12].

3 Background

In this section, we introduce the network models and integration methods used in this paper. The integration methods are then applied to the selected model to derive the neural dynamic equations.

3.1 Current-Based Leaky Integrate-and-Fire neuron

This paper adapts the neuron model introduced in [2]. The Current-Based Leaky Integrate-and-Fire (CuBa LIF) neuron has a closed-form solution for the membrane potential, which can be analytically solved for the spike times depending on the choice of time constants.

The governing equations of the CuBa LIF given in [2] as follows:

$$\frac{du}{dt} = -\frac{1}{\tau_m}u(t) + g(t) - \vartheta\delta(u(t) - \vartheta) \tag{1}$$

$$\frac{dg}{dt} = -\frac{1}{\tau_s}g(t) + \sum_i w_i \sum_{t_i} \delta(t - t_i) \tag{2}$$

Where $u(t)$ is the membrane potential, τ_m is the membrane time constant, $g(t)$ is the post-synaptic current, ϑ is the spiking threshold, τ_s is the synaptic time constant and w_i is the synaptic efficacy from the pre-synaptic neuron i to the current neuron. $\delta(\cdot)$ denotes the dirac-delta function.

By integrating (1) and (2), a closed-form solution for $u(t)$ can be derived with the Spike-Response-Model formulation [13]. By setting $\tau_m = 2\tau_s$, this form can be factorized to produce an analytical solution for the neuron's spike times. A full derivation is presented in [2], along with a simulation framework.

It's worth noting that this neuron model uses a "soft" reset mechanism, whereby the membrane potential decreases by the spiking threshold, rather than being set to a predetermined reset value. In [14], this was demonstrated to partially reduce integration errors.

3.2 Network model and Training

The network trained in this paper consist of densely connected feed-forward architectures with implicit recurrence in the reset dynamics of the neurons.

Spike-time based backpropagation, introduced by [15], is used to train the networks, with errors are decomposed into both inter-neuron and intra-neuron components in order to deal with the reset dynamics [2; 16]. For some loss function $\mathcal{L}$ the following equations are used for weight updates:

$$\Delta w_{i,j}^{(l)} = \sum_{t_k^{(l,j)}} \frac{\partial \mathcal{L}}{\partial t_k^{(l,j)}} \frac{\partial t_k^{(l,j)}}{\partial w_{i,j}^{(l)}} = \sum_{t_k^{(l,j)}} \delta_k^{(l,j)} \frac{\partial t_k^{(l,j)}}{\partial w_{i,j}^{(l)}} \tag{3}$$

Where i is the pre-synaptic neuron, j is the post-synaptic neuron and l is the layer j belongs to. $\delta_k^{(l,j)}$ is further rewritten as the inter-neuron error $\phi_k^{(l,j)}$ and intra-neuron error $\mu_k^{(l,j)}$ for a spike t_k, such that $\delta_k^{(l,j)} = \phi_k^{(l,j)} + \mu_k^{(l,j)}$. These equations are mirrored from [2].

$$\phi_k^{(l,j)} = \sum_i \sum_{t_z^{(l-1,i)}} \frac{\partial \mathcal{L}}{\partial t_z^{(l+1,i)}} \frac{\partial t_z^{(l+1,i)}}{\partial t_k^{(l,j)}} \tag{4}$$

$$\mu_k^{(l,j)} = \sum_{t_z^{(l,j)} \in \{t_z^{(l,j)} > t_k^{(l,j)}\}} \frac{\partial \mathcal{L}}{t_z^{(l,j)}} \frac{\partial t_z^{(l,j)}}{\partial t_k^{(l,j)}} \tag{5}$$

Terms which require the derivative of spike time $t_k^{(l,j)}$ by some variable affecting the membrane potential u can be rewritten through the chain rule and the linear approximation from [15]. For example, $\frac{t_k^{(l,j)}}{\partial w_{i,j}^{(l)}}$ can be rewritten as:

$$\frac{\partial t_k^{(l,j)}}{\partial w_{i,j}^{(l)}} = \frac{\partial t_k^{(l,j)}}{\partial u\left(t_k^{(l,j)}\right)} \frac{\partial u\left(t_k^{(l,j)}\right)}{\partial w_{i,j}^{(l)}} = \frac{-1}{\frac{\partial u\left(t_k^{(l,j)}\right)}{\partial t_k^{(l,j)}}} \frac{\partial u\left(t_k^{(l,j)}\right)}{\partial w_{i,j}^{(l)}} \tag{6}$$

As shown by [17], this is mathematically sound for multiple spiking neural networks and yields the same results as [2].

3.3 Forward-Euler method

The forward Euler method is an explicit first-order integrator commonly used in the simulation of spiking neural networks. For training it is often adapted by applying non-physiological assumptions to simplify its equations [6]; a naive treatment is applied for this paper. For an ordinary differential equation in the form:

$$\frac{dy}{dt} = f(t, y(t))$$

We can estimate the dynamics with time step Δt as follows:

$$y[t + \Delta t] - y[t] = \int_t^{t+\Delta t} f(t, y(t))\, dt$$

Which can be approximated to be:

$$y[t + \Delta t] = y[t] + \Delta t f(t, y[t])$$

But for $t < t_i < t + \Delta t$,

$$\int_t^{t+\Delta t} \delta(t - t_i) = 1$$

This is is the basis of the forward-Euler method, which we apply in turn to our chosen neuron model. Using (1) and (2), we derive:

$$u[t + \Delta t] = u[t] + \Delta t\left(g[t] - \frac{u[t]}{\tau_m}\right) - \vartheta S[t + \Delta t]$$

$$g[t + \Delta t] = g[t] - \frac{\Delta t}{\tau_s} g[t] + \sum_i w_i S_i[t + \Delta t]$$

Where $S[t]$ is either 0 (no spike) or 1 (spike). And $S[t] = \Theta(U[t] - \vartheta)$, $\Theta(\cdot)$ being the Heaviside step function.

3.4 Parker-Sochacki method

The Parker-Sochacki method is a variable-order iterative explicit integration method introduced by [18]. It can provide high integration accuracies with reasonable simulation times and computational costs, but has seen little application to neural networks [8]. In [8], the method yields accurate simulations of Izhikevich and Hodgkin-Huxley models with an adaptive ordering and spike time interpolation schemes. This section will follow a similar methodology, but applied to the CuBa LIF neuron introduced in subsection 3.1. The Parker-Sochacki method can be seen as a generalization of the forward-Euler method, where they are equal when the integration order is one.

In the Parker-Sochacki method, a Maclaurin series is first defined for each model variable. For variable y:

$$y(t) = \sum_{p=0}^{\infty} y_p t^p \tag{7}$$

Where $y_p = \frac{\partial^p y(t)}{\partial t^p} \frac{1}{p!}$. $y_{p+1} = \frac{\partial y_p}{\partial t} \frac{1}{p+1}$ [19], a result which will not be derived here.

For $\frac{\partial y}{\partial t} = f(t, y(t), g(t), \ldots)$, $\frac{\partial y_p}{\partial t} = f(t, y_p, g_p, \ldots)$.

Finally when simulating, a truncated version of (7) is used up to order n:

$$y(t + \Delta t) = y(t) + \sum_{p=1}^{n} y(t)_p (\Delta t)^p \tag{8}$$

Applying (8) to (1) and (2), and assuming we start at $t = 0$, we derive for our model:

$$u[t + \Delta t] = u[t] + \sum_{p=1}^{n} u[t]_p (\Delta t)^p - \vartheta S[t + \Delta t] \tag{9}$$

$$u[t]_{p+1} = \frac{-u[t]_p}{\tau_m (p+1)} + \frac{g[t]_p}{p+1} \tag{10}$$

$$g[t + \Delta t] = g[t] + \sum_{p=1}^{n} g[t]_p (\Delta t)^p + \sum_i w_i S_i[t + \Delta t] \tag{11}$$

$$g[t]_{p+1} = -\frac{g[t]_p}{\tau_s (p+1)} \tag{12}$$

Unlike in [8], all simulations in this paper will be done with a fixed integration order n.

3.5 Spike time interpolation

Spike time interpolation is the use of sub-steps during integration to accurately capture event timings in a neuron, such as output spikes. Due to the importance of spike timings in backpropagation and overall information transfer, spike time interpolation is applied to the forward Euler and Parker-Sochacki methods used in this paper. [20] shows that applying a linear interpolation to the forward Euler method "greatly improves integration performance", due to the elimination of the Δt global order errors in the spike times. Spike time interpolation facilitates the comparison of integration methods by eliminating the method agnostic effects of global errors.

Given that the forward-Euler method is an order one Parker-Sochacki integration, interpolation will described using a procedure provided by [8], with modifications. Whereas they allow one intermediate step per global step, we allow arbitarily many to capture event bursts. The events we consider are input spikes, output spikes and a global time step when the prior do not occur. This global time step is not necessarily aligned with respect to the starting time of the simulation.

The simulation procedure is described in the psuedo code of algorithm 1.

Algorithm 1 Spike time interpolation applied to the Parker-Sochacki method for a CuBa LIF neuron.

Input: $f(u[t], g[t], \Delta t)$ the Parker-Sochacki solver returning $u[t + \Delta t]$ and $g[t + \Delta t]$.
Input: $u[0]$ and $g[0]$ the initial states.
Input: $\overline{t_{in}}$ the sorted input spikes array.
Input: Δt the global time step.
Input: ϑ the spiking threshold.
Input: max_simulation the maximum simulation time.
Output: $\overline{t_{out}}$ the interpolated output spikes.

```
u[t] ← u[0]
g[t] ← g[0]
simulation_time ← 0
while simulation_time < max_simulation do
    Δt_0 ← min(Δt, next(t_in))
    u[t + Δt_0], g[t + Δt_0] ← f(u[t], g[t], Δt_0)
    if (t + Δt_0) ∈ t_in then
        g[t + Δt] = g[t + Δt_0]+ weight(t + Δt_0)
    end if
    if u[t + Δt_0] ≥ ϑ then
        Δt_0 ← newton_raphson(u[t], g[t])
        t_out ← put (t + Δt_0, t_out)
        u[t + Δt_0], g[t + Δt_0] ← f(u[t], g[t], Δt_0)
    end if
    simulation_time ← simulation_time +Δt_0
    u[t], g[t] ← u[t + Δt_0], g[t + Δt_0]
end while
```

The Newton Raphson method is applied to find the exact point that the threshold was crossed during simulation. More specfically, we find the root of the function:

$$f(\Delta t_{pre}) = u[t] + \sum_{p=1}^{n} u[t]_p (\Delta t_{pre})^p - \vartheta$$

Where Δt_{pre} is the time from t to the spike. The derivative of $f(\Delta t_{pre})$ with respect to Δt_{pre} is as follows:

$$\frac{df(\Delta t_{pre})}{d\Delta t_{pre}} = \frac{du[t]}{d\Delta t_{pre}} + \sum_{p=1}^{n} u[t]_p \frac{d(\Delta t_{pre})^p}{d\Delta t_{pre}} - \frac{d\vartheta}{d\Delta t_{pre}} \tag{13}$$

$$= \sum_{p=0}^{n} u[t]_{(p+1)} (p+1) (\Delta t_{pre})^p$$

Starting with an initial guess of 0, the Newton Raphson will converge to the first Δt_{pre} such that $f(\Delta t_{pre}) = 0$. There are caveats to this however, which will be discussed in section 5.

4 Methodology

This section will present the steps carried out to explore the effects of spike time interpolation on network dynamics and will introduce and explain an event-based approach to backpropagation for discrete simulations based on the general formulas from subsection 3.2. The general form is similar to backpropagation through time, but only events contribute to the error. We also note that all experiments were carried out with Numpy seed 42801237 and Cupy seed 526457712.

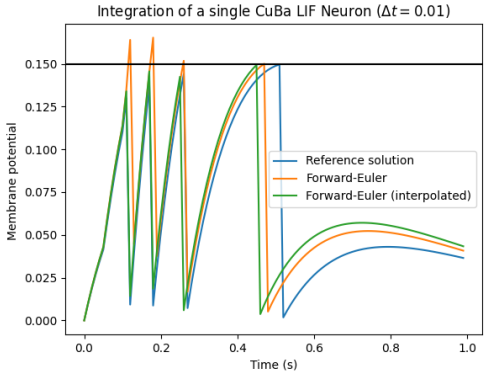


Figure 1: Example integration comparing the forward-Euler method with and without interpolation. The black line represents the threshold ϑ. Interestingly, the interpolated version does not always perform better, which can be seen with how early it fires for the last spike with respect to the the blue reference solution.

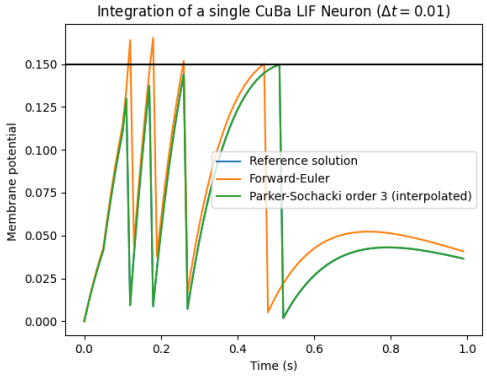


Figure 2: Example integration where a third order Parker-Sochacki integration with interpolation now consistently outperforms the forward-Euler method, tracing the reference solution perfectly.

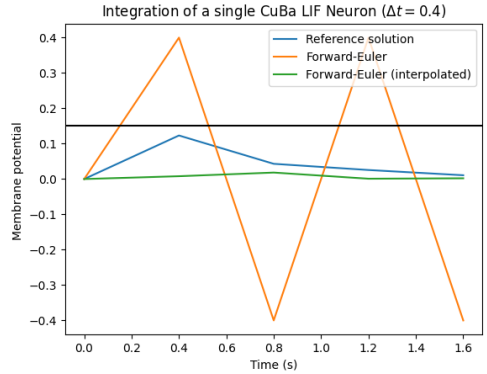


Figure 3: Example integration of stiff neuron dynamics, with a large time step. The forward-Euler method no longer exhibits A-stability, resulting in infinite oscillation. The interpolated version still manages to converge to the referenec solution, indicating that interpolation may help alleviate instability problems during simulation by providing adaptive time stepping around critical regions.

4.1 Weight transfer

To aid the analysis of training data, we first look at the effects of different numerical integrations on network accuracy during inference. An analytical network with a 784-800-10 feedforward architecture is first pre-trained for 20 epochs on MNIST, and its weights are transferred to networks of the same morphology simulated using the Parker-Sochacki method in the forward pass. Similar to [2], we test on a subset of 10,000 images not used during training.

4.2 Parker-Sochacki backpropagation

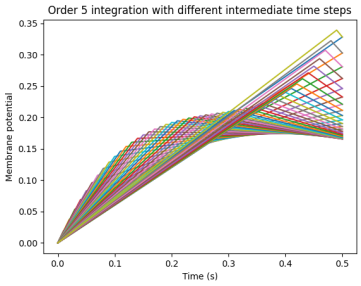
Backpropagation for the forward-Euler and Parker-Sochacki methods with spike time interpolation are simpler to analyze than fixed step integrations due to the continous spike times, but require careful treatment. We derive equations for: $\frac{\partial u[t_k^{(l,j)}]}{\partial w_{i,j}^{(l)}}$, $\frac{\partial u[t_k^{(l,j)}]}{\partial t_z^{(l-1,i)}}$, $\frac{\partial u[t_k^{(l,j)}]}{\partial t_z^{(l,j)}}$ and $\frac{\partial u[t_k^{(l,j)}]}{\partial t_k^{(l,j)}}$. In order to simplify notation, we introduce $f_u(u[t], g[t], \Delta t)$ to be an integration step of the Parker-Sochacki method for u, and similarly $f_g(g[t], \Delta t)$ for g. The methods derived generalize to other explicit integration methods however.

Gradients are accumulated forward, meaning we first differentiate at the origin of the gradients and then derive an second equation that propagates this forward through time.

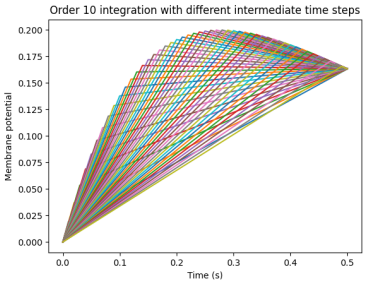
$$\frac{\partial u[t_{post}]}{\partial w_{i,j}^{(l)}} = \tag{14}$$
$$\frac{\partial f_u\left(u[t_k^{(l,j)}], g[t_k^{(l,j)}] + w_{i,j}^{(l)}, t_{post} - t_k^{(l,j)}\right)}{\partial t_{post} - t_k^{(l,j)}} \frac{\partial t_{post} - t_k^{(l,j)}}{\partial w_{i,j}^{(l)}} +$$
$$\frac{\partial f_u}{\partial u[t_k^{(l,j)}]} \frac{\partial u[t_k^{(l,j)}]}{\partial w_{i,j}^{(l)}} + \frac{\partial f_u}{\partial g[t_k^{(l,j)}] + w_{i,j}^{(l)}} \frac{\partial g[t_k^{(l,j)}] + w_{i,j}^{(l)}}{\partial w_{i,j}^{(l)}}$$
$$= \frac{\partial f_u}{\partial u[t_k^{(l,j)}]} \frac{\partial u[t_k^{(l,j)}]}{\partial w_{i,j}^{(l)}} + \frac{\partial f_u}{\partial g[t_k^{(l,j)}] + w_{i,j}^{(l)}} \frac{\partial g[t_k^{(l,j)}] + w_{i,j}^{(l)}}{\partial w_{i,j}^{(l)}}$$

Where t_{post} is the integration point following an input spike. Since the ODEs used by the Parker-Socachki method are linear, the last equation above is the same as doing a forward step with $\frac{\partial u[t_k^{(l,j)}]}{\partial w_{i,j}^{(l)}}$ and $\frac{\partial g[t_k^{(l,j)}] + w_{i,j}^{(l)}}{\partial w_{i,j}^{(l)}}$ as the membrane potential and current, with time step $t_{post} - t_k^{(l,j)}$. When propagating from a non-input spike forward, the weight term is removed in the derivative i.e. $\frac{\partial g[t]}{\partial w_{i,j}^{(l)}}$ is used instead. For an input spike we get:

$$\frac{\partial u[t_{post}]}{\partial t_k^{(l,j)}} = \tag{15}$$
$$\frac{\partial f_u\left(u[t_k^{(l,j)}], g[t_k^{(l,j)}] + w_{i,j}^{(l)}, t_{post} - t_k^{(l,j)}\right)}{\partial t_{post} - t_k^{(l,j)}} \frac{\partial t_{post} - t_k^{(l,j)}}{\partial t_k^{(l,j)}} +$$
$$\frac{\partial f_u}{\partial u[t_k^{(l,j)}]} \frac{\partial u[t_k^{(l,j)}]}{\partial t_k^{(l,j)}} + \frac{\partial f_u}{\partial g[t_k^{(l,j)}] + w_{i,j}^{(l)}} \frac{\partial g[t_k^{(l,j)}] + w_{i,j}^{(l)}}{\partial t_k^{(l,j)}}$$



(a)



(b)

Figure 4: Sample integration of membrane potential from 0s to 0.5s with 2 steps. The different lines show the integration path with varying times for the intermediate integration point. Ideally the results are identical, but in 4a there are large discrepencies. This subsides in the higher order integration of 4b. Practically, this means the derivative of the result with respect to the intermediate integration point is not zero, introducing noise into the training process.

This first derivation is naive as is does not account for noise introduced by the integration process as illustrated in Fig. 4. This noise is characterized by $\frac{\partial u[t_{post}]}{\partial t_k^{(l,j)}}$ without taking into account the weight term. When subtracted from (15), the result is:

$$\frac{\partial u[t_{post}]}{\partial t_k^{(l,j)}} = -\frac{\partial f_u\left(0, w_{i,j}^{(l)}, t_{post} - t_k^{(l,j)}\right)}{\partial t_{post} - t_k^{(l,j)}} \tag{16}$$

Both the naive and adjusted versions of the backpropagation equations were tested. For brevity, the $\frac{\partial u[t_k^{(l,j)}]}{\partial t_z^{(l,j)}}$ term is not derived, but follows the same form as (15). Finally, $\frac{\partial u[t_k^{(l,j)}]}{\partial t_k^{(l,j)}}$ is identical to (13).

5 Experimental Setup and Results

5.1 Weight transfer

Two experiments were run with the weight transfer method for MNIST. The first one is run on the original MNIST dataset where pixel values range from 0 to 255 and a linear latency encoding is used. The second experiment thresholds all values to either 0 or 255, with the threshold at 128. The baseline performances are 98.79 and 98.55 respectively.

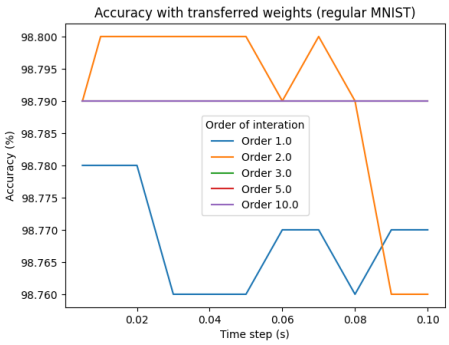


Figure 5: Accuracies at different integration orders on the original MNIST dataset with weights transferred from an analytical network. All orders perform similarly across different time steps, with the largest differences being 0.04%.

The thresholded test eliminates the effects of spike time interpolation for input spikes in the first layer since they all occur at $t = 0$.

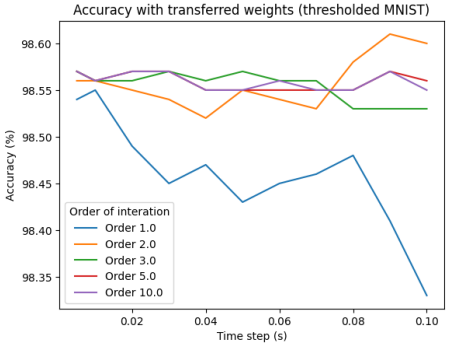


Figure 6: Accuracies at different integration orders on a thresholded MNIST dataset with weights transferred from an analytical network. Higher orders perform similarly with increasing time steps, but the forward-Euler method sees gradual performance degredation.

Figures 5 and 6 both demonstrate how numerically simulated networks with spike time interpolation can achieve on-par accuracies with their analytical counterpart. Furthermore, minimizing spike time interpolation at the first layer leads to greater relative accuracy variations across time steps.

5.2 Training

Training experiments were done under the same condiitons as the weight transfer experiments, but the numerically simulated networks were trained directly through backpropagation. Three time steps (0.1, 0.05, and 0.01) were explored across three orders of integration (1, 3, 5). Data was collected related to the accuracy of the networks during training, their spike counts, and real training time. The experiments were run on an NVIDIA GeForce RTX 4070 mobile graphics card, which represented the main bottleneck during training. The first three graphs are for the naive backpropagation, and the last for the adjusted backpropagation.

Figures 7 and 10 both confirm that higher order integrations converge quicker and to higher accuracies as predicted.

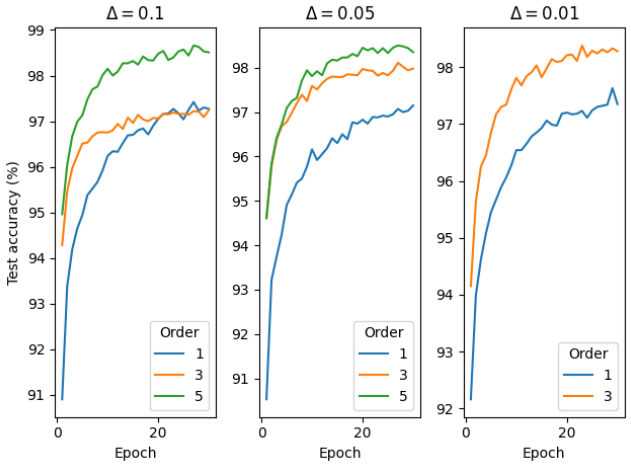


Figure 7: Test accuracies across different Δt and integration orders with naive backpropagation. No results were recorded for $\Delta t = 0.01$ and order 5. As Δt decreases, the discreprencies between higher orders begins to close.

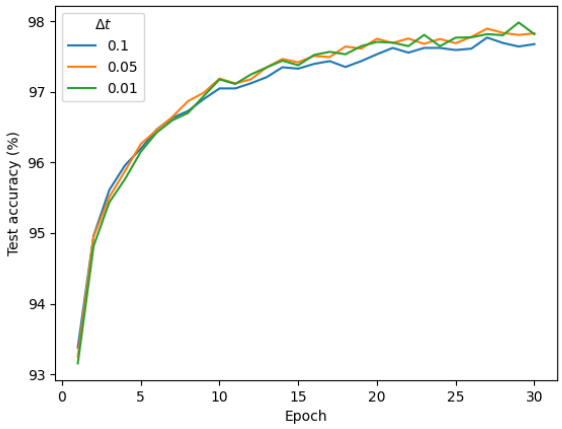


Figure 8: Average test accuracies across different Δt with naive backpropagation. Lower Δts train marginally better, but the influence of the parameter is low.

The difference is most notable between orders 1 and 5. In Fig.8 and 11, we see how Δt has little effect on training when employing spike time interpolation, which is most clear in the similarities of the subfigures in Fig. 10. This pattern does not hold for order 3 in Fig. 7. The overall final accuracies after training are on par with the analytical network, which achieved 98.63% accuracy. In comparison, the order 5 network with $\Delta t = 0.1$ achieved 98.54% accuracy with the adjusted backpropagation. Training time is the largest difference between the two experiments. In Fig. 9, the training time increases as high as 500 seconds per epoch, with a hidden layer spike count of 4000. This is expected due backpropagation being quadratic with respect to output spike counts. With the modified backpropagation rule, Fig.12, we see at most a training time of 210 seconds, with fluctuations peak-

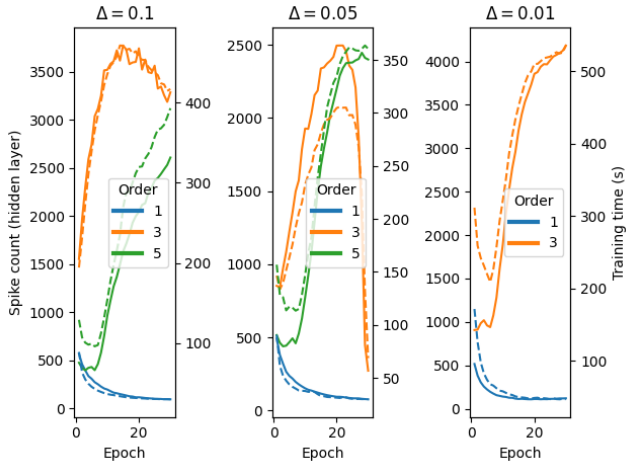


Figure 9: Training times across different Δt and integration orders with naive backpropagation. The dashed and solid lines represent the time and hidden layer spike count respectively. They are highly positively correlated. First order integrations tend toward lower hidden layer spike counts, whilst higher ones tend toward higher ones.

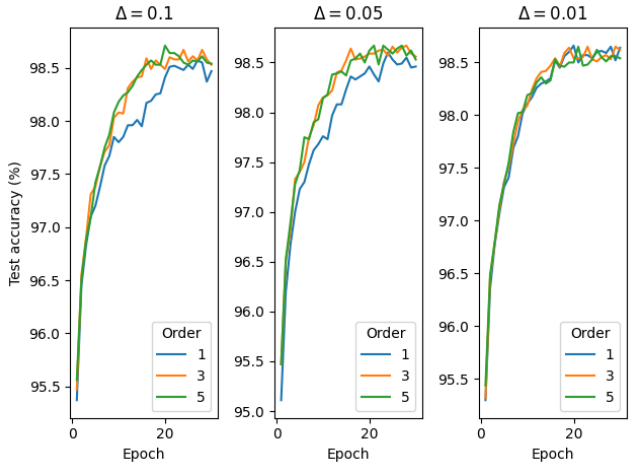


Figure 10: Test accuracies across different Δt and integration orders with adjusted backpropagation. Higher orders onverge quicker to better accuracies, but there is little difference to be noted between order 3 and 5.

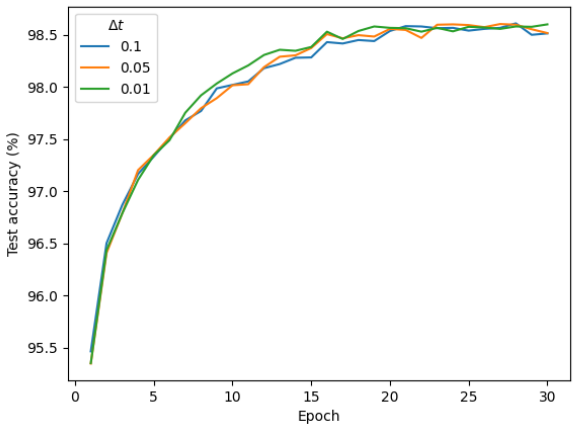


Figure 11: Average test accuracies across different Δt with adjusted backpropagation. Lower Δts converge faster, but they all achieve a similar accuracy after 30 epochs.

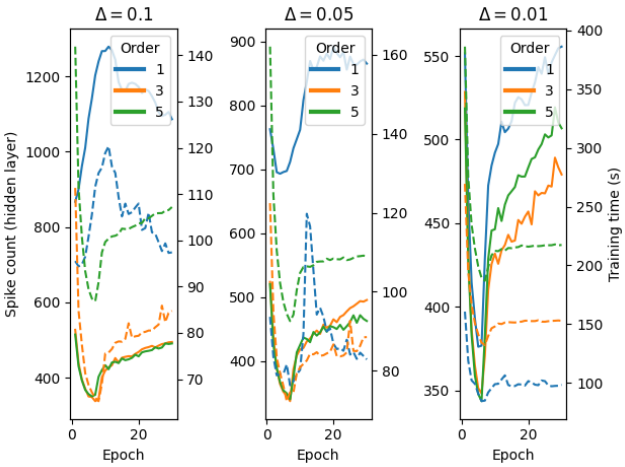


Figure 12: Training times across different Δt and integration orders with adjusted backpropagation. The dashed and solid lines represent the time and hidden layer spike count respectively. Training times stay relatively consistent throughout, despite fluctuations in the hidden spike count.

ing at 40 seconds. The trend is flatter in general and more closely resembles the consistent training times of the analytical network.

6 Discussion

In this paper we tested how numerically simulating spiking neural networks affects their training accuracy and speed (both in epochs and time), and how spike time interpolation techniques may have contributed to this observed behaviour.

In section 5.1, the performance of numerically simulated networks with spike time interpolation were close to those of an analytical network across different integration orders and time steps. A general downward trend was expected with step size, but was only observed in Fig. 6 for a first order integration. The corresponding experiment was designed to minimize the impact of spike time interpolation by increasing the input sparsity, suggesting two conclusions; spike time interpolation helps stabilize network activity and accuracy when activity is regular, and; higher order integrations can provide accurate results over wide ranges of time deltas, meaning they are more stable with low event sparsity than the forward-Euler method. The main challenges faced by the implementation of spike time interpolation are more sophisticated hardware requirements, potentially infeasible in architectures such as [4], due to use of the Newton-Raphson method, and floating point spike times and integration calculations. We also note that despite higher order integrations

showing stability with a Δt as high as 0.1 for a simulation time of 0.2, excessive Δts will cause large overestimations of the membrane potential, leading to ghost spikes or backward time travelling spike times due to the initial conditions of the Newton-Raphson method. We guard agains this in our implementation through generous use of assert statements.

In section 3.2, we proceed to train numerically simulated networks and find they approach the accuracy of their analytical counterpart. Training time becomes a key issue. Convergence with respect to epochs remains consistent for an integration order over different Δts, but an interesting exception occurs for $\Delta t = 0.01$ and order 1 in Fig. 10. As the delta time is very small, the forward-Euler method can still perform similarly to high-order ones, as demonstrated by looking at Fig. 5 and 6, due to having more integration points. The lower computational cost also results in lower training times, despite having more spikes in the hidden layer, which is seen in Fig. 12. The latter result conflicts with Fig. 9 due to the relative magnitude of the spike count changes; changes of a couple hundred spikes will not make a noticeable difference to performance, presumably since the GPU is not yet bottlenecked. This suggests that with low time deltas, the integration order does not have much effect on the training accuracy and convergence of the model, but does negatively affect training times. If we instead consider network sparsity in Fig. 12, the higher order integrators find sparser solutions with equivalent accuracy, a result which agrees with the analysis in [9]. This latter point is important for neuromorphic hardware where spike counts are directly correlated to energy consumption.

In 3.2, a distinction has been made between a naive and modified backpropagation procedure, the latter of which accounts for noise in the derivatives introduced by the integration process. We hypothesize that the network tries to minimize the influence of this noise in the naive solution by increasing network activity. In tandem with spike time interpolation, this decreases the effective time delta of the network, thereby making the output of the integration less reliant on any specific event. This effect needs further exploration to be substantiated, however. Regardless, the effect is antagonistic to the goal of achieving reasonable training times. In automatic differentiation engines, a naive backpropagation can therefore lead to such issues without the awareness of the programmer. A possible fix is discussed in the future works section.

Despite compelling results, we note that this research is limited in its analysis, in several regards. Due to bugs encountered whilst developing the simulation software, and time constraints, few experiments were run. Only 10 experiments are shown in section 5.1 and 17 experiments in section 5.2. The limited set of data points may be flawed in their generality but do show consistent trends. We also note that the dataset used, MNIST, is trivially solved by most neural network architectures to high accuracies and may not provide insight into the more complex dynamics of numerically integrating spiking neural networks. More datasets could not be explored due to time constraints. Finally, we note that we combine an analysis of numerical integration methods and spike time interpolation, but do not do an ablation study. We provide numerous suggestions for their interplay, but have little empirical evidence for them.

7 Conclusions and Future Work

We aimed to address two questions:

How accurate is a numerically simulated spiking neural network with spike time interpolation compared to its analytical counterpart with the same weights when tested on MNIST?

How are the training accuracy and training speed (in epochs and time) of a spiking neural network affected when numerically integrating with the forward-Euler and Parker-Sochacki methods?

By exploring a CuBa LIF based spiking neural network, we provided methods for the application and analysis of spike time interpolation and backpropagation when numerically integrating neuron dyanmics. We conclude that networks with spike time interpolation perform similarly to their analytical counterparts during inference on MNIST with shared weights. Furthermore, spike time interpolation provides stability to networks with regular activity. With this result in mind, we conclude that numerically integrated networks have similar characteristics, overall network accuracy, and convergence rate to their analytical counterparts, as long as the backpropagation is properly treated. We provide insight into how to do the latter by demonstrating that integrations can introduce noise into backpropagation equations. Our research shows that higher order integrations will generally perform better, but this depeends on your time delta and network activity.

7.1 Future Work

Our analysis focused on the CuBa LIF neuron model, but as discussed in the introduction, there are many more available. Future work can investigate training models with a wider range of neuron models, on deeper tasks to expand upon our work, and the work of [9]. Furthermore, spike time interpolation is applied in this paper, but no ablation study is performed without it. Further insight into its effects are useful due to the added hardware complexity and computational costs of its implementation.

This paper solely explores explicit integration methods with transparent implementations where backpropagation can be directly applied. In the case of implicit methods or black-box integrators in general, this analysis is not as useful. Attention has recently been given to models such as Nueral ODEs, where solving the ODE and backpropagation are made independent through adjoint sensitivity analysis techniques [21]. A similar concept has been applied in conjunction with hybrid system analysis in the EventProp algorithm, which allows exact backpropagation on simple spiking neural networks with arbitrary integrators [22]. The paper only does analytical integration, so further research of its method applied to numerically integrated networks is suggested. This also opens the door to more efficient backpropagation approaches.

8 Responsible Research

8.1 Reproducibility

All the code use for experiments can be found on GitLab. Hyperparameters are the same as found in the Parker-Sochacki spike count training files in the MNIST experiment. By fixing and providing the seed in the experimental setup section, the experiments are deterministically reproducible. The code for data processing is also provided, along with the experiments used in this paper. Some experiments may require changing high level variables, such as integration orders, which should be clear from the context of the graphs they produce. It could however be that some changes have not been systematically recorded and would require more effort to reproduce.

8.2 Experimental data

All experimental data is included in the analysis provided by the paper. Any data removed represents incomplete training runs, results from buggy code or data which didn't affect the final analysis.

8.3 Programming

It would be prudent that researchers using our code check for bugs. One particular issue related to backpropagating errors to input spikes triggered assert statements under unknown conditions. As it was not easily reproducible this particular bug has not been addressed yet. When not crashing, these assertions also guarantee the correctness of the experimental results. If any inconsistencies have been found between our descriptions and the code, please contact our primary author.

8.4 Environmental impact

Training and simulating neural networks can be computational expensive and energy intensive. This research focuses on shallower networks with low computational costs, but still considers the effects of training methods on network efficiency. The study of spiking neural networks may also be beneficial in the long term by providing low power alternatives to traditional deep neural networks. Further research should keep in mind the balance of their contributions' effect on the environment.

References

[1] Nergis Tomen and Udo Ernst. The role of criticality in flexible visual information processing. *The functional role of critical dynamics in neural systems*, pages 233–264, 2019.

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

[3] Julian Göltz, Laura Kriener, Andreas Baumbach, Sebastian Billaudelle, Oliver Breitwieser, Benjamin Cramer, Dominik Dold, Akos Ferenc Kungl, Walter Senn, Johannes Schemmel, et al. Fast and energy-efficient neuromorphic deep learning with first-spike times. *Nature machine intelligence*, 3(9):823–835, 2021.

[4] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Prasad Joshi, Andrew Lines, Andreas Wild, Hong Wang, and Deepak Mathaikutty. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, PP:1–1, 01 2018.

[5] Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.

[6] Jason K Eshraghian, Max Ward, Emre O Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D Lu. Training spiking neural networks using lessons from deep learning. *Proceedings of the IEEE*, 2023.

[7] Giuseppe de Alteriis and Calogero Maria Oddo. Trade-off between accuracy and computational cost of euler and runge kutta ode solvers for the izhikevich spiking neuron model. In *2021 10th International IEEE/EMBS Conference on Neural Engineering (NER)*, pages 730–733, 2021.

[8] Robert D Stewart and Wyeth Bair. Spiking neural network simulation: numerical integration with the parker-sochacki method. *Journal of Computational Neuroscience*, 27:115–133, 2009.

[9] Mario De Florio, Adar Kahana, and George Em Karniadakis. Analysis of biologically plausible neuron models for regression with spiking neural networks. *arXiv preprint arXiv:2401.00369*, 2023.

[10] Bruno Magalhães, Michael Hines, Thomas Sterling, and Felix Schürmann. Fully-asynchronous fully-implicit variable-order variable-timestep simulation of neural networks. In *Computational Science–ICCS 2020: 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part V 20*, pages 94–108. Springer, 2020.

[11] Gang Zheng, Arnaud Tonnelier, and Dominique Martinez. Voltage-stepping schemes for the simulation of spiking neural networks. *Journal of computational neuroscience*, 26:409–423, 2009.

[12] Guillermo L Grinblat, Hernán Ahumada, and Ernesto Kofman. Quantized state simulation of spiking neural networks. *Simulation*, 88(3):299–313, 2012.

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

[14] Yufei Guo, Yuanpei Chen, Liwen Zhang, YingLei Wang, Xiaode Liu, Xinyi Tong, Yuanyuan Ou, Xuhui Huang, and Zhe Ma. Reducing information loss for spiking neural networks. In *European Conference on Computer Vision*, pages 36–52. Springer, 2022.

[15] 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.

[16] Wenrui Zhang and Peng Li. Temporal spike sequence learning via backpropagation for deep spiking neural

networks. *Advances in neural information processing systems*, 33:12022–12033, 2020.

[17] Wenyu Yang, Dakun Yang, and Yetian Fan. A proof of a key formula in the error-backpropagation learning algorithm for multiple spiking neural networks. In *International Symposium on Neural Networks*, pages 19–26. Springer, 2014.

[18] G Edgar Parker and James S Sochacki. Implementing the picard iteration. *Neural, Parallel & Scientific Computations*, 4(1):97–112, 1996.

[19] Joseph W Rudmin. Numerical solution of differential equations by the parker-sochacki method. *arXiv preprint arXiv:1007.1677*, 2010.

[20] David Hansel, Germán Mato, Claude Meunier, and L Neltner. On numerical simulations of integrate-and-fire neural networks. *Neural Computation*, 10(2):467–483, 1998.

[21] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[22] Timo C Wunderlich and Christian Pehle. Event-based backpropagation can compute exact gradients for spiking neural networks. *Scientific Reports*, 11(1):12829, 2021.