Simulating Multi-Compartmental Neuron Morphologies with Gradient Support

Internship Report

Thorge Janz

June 2025

European Institute for Neuromorphic Computing, Heidelberg

Supervisors: Amani Atoui, Jakob Kaiser, Eric Müller

Abstract

As a first step towards in-the-loop gradient-based training of multi-compartmental neuron models on the BrainScales-2 (BSS-2) architecture, this internship establishes a workflow for the simulation of specified morphologies with gradient support. Different schemes for numerical integration are tested on a 3-compartment neuron consisting of an AdEx (Adaptive Exponential Leaky Integrate-and-Fire) soma as well as passive basal and apical dendrites. For this, a custom solver built in *PyTorch* is compared to a reference routine set up with *Brian2* in simulation. Alongside other testing, the custom solver is trained on an optimization task to verify the proper computation and passing of gradients.

In simulations using spike-based input, the custom solver produces spike times within $0.1\,\mathrm{ms}$ to the reference. Both solvers produce the same characteristics in the membrane traces regarding the deployed AdEx dynamics and the specified refractory period. The training procedures in the optimization task show the full differentiability of the custom solver and its support for faithful backpropagation of gradients.

Contents

Motivation **1**

Theoretical Background **2**
Setting up the ODEs 2
Numerical Integration Schemes 3

Results of Simulations **5**
Forward Euler (Brian2) 5
Implicit Euler (Hines/ Newton-Raphson) 5
Long-run comparison 5
Spike-based Input 5
Soft Spiking/ Gradient Computation 5

Specification of Methods **9**

Discussion **9**

Outlook **9**

Motivation

Spiking Neural Networks (SNNs) provide a biologically grounded alternative to conventional artificial neural networks by encoding and processing information through discrete, spike-based events. Using multi-compartmental neuron models in SNNs instead of point-like neurons further enhances biological realism by introducing spatial complexity. Furthermore, specific dynamics for compartments and their interaction can be investigated [8].

The neuromorphic platform of BSS-2 enables fast and energy-efficient emulation of SNNs on analog hardware [7]. On its circuits, Adaptive Exponential Leaky Integrate-and-Fire (AdEx) dynamics are implemented. Simulating non-linear dynamics such as AdEx poses a challenge for the numerical integration of the voltage updates and is not supported by every solver. For example, the *JAX* based tool *JAXLEY* [1] is a fully differentiable simulator for multi-compartmental neuron models. It has pre-defined ionic channels for specific, non-linear models, however not for AdEx (also, the internal *Fire* channel does not support gradients). On the other hand, there are non-differentiable platforms like *Brian2*, where user-specified, non-linear dynamics come without the support for the backpropagation of gradients.

Issues of differentiability are inherent to spike-based computation due to the use of non-differentiable step functions used for modeling spike events. Soft spiking mechanisms using surrogate gradients can overcome this barrier by the means of differentiable approximation. An interface for the gradient-based training of point-like neurons on the BSS-2 hardware was already implemented in the scope of a master thesis by Simon Jonscher [4]. The aim now is to extend this to the training of multi-compartmental neuron models.

For this, the basis of the single neuron case is essential. To replicate the behaviour of complex, biological neurons, multi-compartmental neuron models are deployed [5]. In this report, the differntiable simulation of a 3-compartment neuron with an AdEx soma and passive dendrite compartments is established. Before looking at networks consisting of multiple neurons, the first step of the bachelor thesis ensuing this report will be to move this differentiable single neuron model to BBS-2 hardware.

Employing and testing gradient-based learning on neuromorphic hardware helps to bridge the gap between biologically inspired spiking models and modern optimization techniques. By introducing gradient support to the simulation of multi-compartmental neuron models, this report helps to lay the ground work for this undertaking.

Theoretical Background

Before going into results of simulation, the necessary theoretical background for the experiment setup is presented. This covers establishing a system of coupled Ordinary Differential Equations (ODEs) passed to the solvers, as well as the methods of numerical integration used by the solvers to approximate solutions (in this case, the voltage traces of the individual compartments).

Setting up the ODEs

Each neuron compartment follows an individual equation governing the evolution of the compartmental membrane voltage in time [3]. These are usually modeled using a discretized version of the cable equation, effectively treating the different compartments as RC circuits. In general, each compartment i contributes

$$C_i \frac{dV_i}{dt} = \underbrace{g_\ell (V_\ell - V_i)}_{\text{Leak current}} + \underbrace{\sum_j g_{ij}(V_j - V_i)}_{\text{Axial currents}} + \underbrace{I_{\text{ext},i}}_{\text{External stimulus}} + \underbrace{I_{\text{dyn},i}}_{\text{Extra model dynamics}} \tag{1}$$

C_i : Membrane capacitance
V_i : Membrane voltage
g_ℓ : Leak conductance
V_ℓ : Leak current
g_{ij} : Axial conductance

to form a system of coupled ODEs. The g_{ij} define the coupling strength between compartments i and j. For disconnected compartments, g_{ij} is zero. Thus, the g_{ij} describe the topology of the neuron. $I_{ext,i}$ can take many forms. The performed simulations exclusively use synaptic input, $I_{syn,i}$. Through $I_{dyn,i}$, characteristic dynamics of different compartments are specified. In the case of AdEx dynamics for the soma discussed in this report, we regard

$$I_{dyn,s} = \underbrace{g_\ell \Delta_{exp} \exp\left(\frac{V_s - V_{\text{exp}}}{\Delta_{\text{exp}}}\right)}_{\text{Exponential depolarization current}} - \underbrace{w}_{\text{Adaptation Current}} . \tag{2}$$

Δ_{exp} : Slope factor
V_{exp} : Exponential threshold

Here, the adaptation current evolves as

$$\tau_w \frac{dw}{dt} = a(V_s - V_\ell) - w \ . \tag{3}$$

τ_w : Adaption time constant
a : Subthreshold adaption conductance

The compartment follows a spike mechanism, where $V_s \geqslant V_{\text{th}}$ triggers

$$\begin{aligned} V_s &\to V_{\text{reset}} \ , \\ w &\to w + b \ . \end{aligned} \tag{4}$$

V_{th} : Spike threshold
V_{reset} : Reset voltage
b : Spike increment

If a spike occurs, a refractory period is activated during which voltage updates are frozen. This prevents subsequent spikes after the compartment has fired.

Numerical Integration Schemes

For the numerical integration of the introduced system, a variety of different schemes can be used. A first, simple approach is the Forward Euler scheme which computes updates for equations of the form

$$\frac{dV}{dt} = f(V, t) \tag{5}$$

as

$$V_{t+\Delta t} = V_t + \Delta t \ \cdot \ f(V_t, t). \tag{6}$$

However, this method can become unstable for large Δt or complex morphologies [9]. For this reason, implicit schemes like Backward Euler are introduced where updates solve for the unknown $V_{t+\Delta t}$ instead:

$$V_{t+\Delta t} = V_t + \Delta t \ \cdot \ f(V_{t+\Delta t}, t + \Delta t). \tag{7}$$

These are better at handling stiff systems (where different variables evolve on different time scales). To leverage the sparsity of the neurons tree-like structure and make the solver more efficient, we can further bring the system into the form

$$GV^{[t+1]} = \text{RHS}^{[t+1]} \tag{8}$$

where $V^{[t+1]}$ and $\text{RHS}^{[t+1]}$ are vectors and G is a $N \times N$ matrix (N = Number of compartments). In $\text{RHS}^{[t+1]}$, all of the terms not explicitly relying on $V^{[t+1]}$ are collected. To efficient solve such a system, the Hines algorithm [2] together with Hines numbering can be deployed. Hines numbering renders each compartment only dependent on itself and its parent compartment. This is achieved by reordering the branches of the neuron tree in a way that all child branches are named before their parents. Figure 1 shows the numbering procedure for a 5-compartment neuron in branch representation[1], which will become powerful when dealing with more complex morphologies or networks consisting of multiple neurons. Figure 2 shows a possible configuration for a Hines-numbered 3-compartment neuron with the soma as the parent compartment.

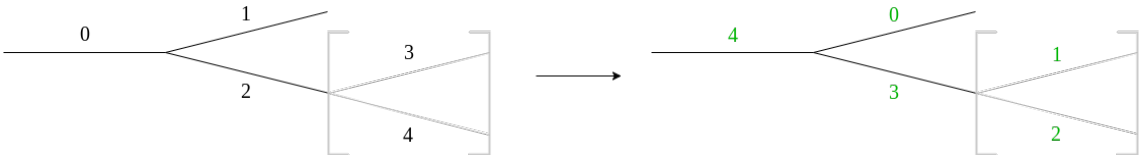

Figure 1: Hines Numbering procedure: The compartments of the neuron tree are renumbered such that all children are named before their parents. Any configuration satisfying this is valid.

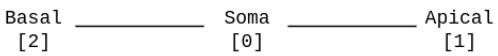

Figure 2: Linear representation of a Hines-numbered 3-compartment neuron with the soma as the parent compartment.

Hines renumbering renders the G matrix sparse and maximal tridiagonal. For the example of the 3-compartment neuron with an AdEx soma and passive dendrites, this yields

$$G = \begin{bmatrix} \frac{C_a}{\Delta t} + g_{\ell,a} + g_{as} & 0 & -g_{as} \\ 0 & \frac{C_b}{\Delta t} + g_{\ell,b} + g_{bs} & -g_{bs} \\ -g_{as} & -g_{bs} & \frac{C_s}{\Delta t} + g_{\ell,s} + g_{as} + g_{bs} - \partial_{V_s} I_{\text{exp}} \end{bmatrix} \tag{9}$$

[1] Branches are connected using nodes to form the neuron tree. Structures on branches, like compartments, are referred to as segments.

with I_{exp} denoting the exponential depolarization current from equation (2). This form enables efficient forward elimination and back substitution. This is leveraged by the Hines scheme, rendering it $\mathcal{O}(N)$ instead of $\mathcal{O}(N^3)$ as with regular lower-upper (LU) decomposition. To solve equation (8), the Hines algorithm uses two procedures: Triangulization (*TRIAG*) and Backsubstitution (*BKSUB*), both shown in Figure 3. First, *TRIANG* is applied to the branches in the original order. Then, *BKSUB* is applied following the Hines-reordering.

TRIANG

Do for i = second segment of the branch to the last segment of the branch:

$A_{i,i} = A_{i,i} - A_{i,i-1}(A_{i,i-1}/A_{i-1,i-1})$

$\texttt{RHS}_i = \texttt{RHS}_i - \texttt{RHS}_{i-1}(A_{i,i-1}/A_{i-1,i-1})$

If the last segment, j, of the branch is connected to a node, k, then:

$A_{k,k} = A_{k,k} - A_{j,k}(A_{k,j}/A_{j,j})$

$\texttt{RHS}_k = \texttt{RHS}_k - \texttt{RHS}_j(A_{k,j}/A_{j,j})$

BKSUB

If the last segment, j, of the branch is connected to a node, k:

$V_j = (\texttt{RHS}_j - V_k A_{j,k})/A_{j,j}$

else:

$V_j = \texttt{RHS}_j/A_{j,j}$

Do for i = next-to-last segment of the branch to the first segment:

$V_i = (\texttt{RHS}_i - V_{i+1}A_{i,i+1})/A_{i,i}$

Figure 3: Procedures of the Hines algorithm as presented in [2]. Here, $A = G$.

In general, there are two options for dealing with non-linearities. The first one is to use the Hines procedure with a designated scheme for linear terms only and have a different scheme deal with non-linear contributions [10]. The second option is to use a scheme that can handle non-linear entries in the G matrix like displayed in equation (9). For the latter, a robust candidate is the Newton-Raphson scheme. Given a non-linear system of equations is in the form

$$F(V) = 0, \tag{10}$$

it computes update using a Jacobian $J = \frac{\partial F}{\partial V}$ as

$$V_{k+1}^{[t+1]} = V_k^{[t+1]} - J^{-1}(V_k^{[t+1]})F(V_k^{[t+1]}). \tag{11}$$

This (with full support for the Hines algorithm using Hines numbering) is whats being implemented by the custom solver presented in this report. For this, equation (8) is brought into the form of equation (10) by subtraction. A Taylor expansion of $F(V)$ then implies

$$F(V_k^{[t+1]}) + J_k \cdot \delta V_k \approx 0. \tag{12}$$

The index k in both of these equations denotes the inner iterations eventually converging to $V^{[t+1]}$, for which the solver performs

$$\begin{aligned} &(1) : \delta V_k = \text{solve}\ (J_k, -F(V_k^{[t+1]})), \\ &(2) : V_{k+1} = V_k + \delta V_k. \end{aligned} \tag{13}$$

Results of Simulations

Two solvers are set up to simulate voltage traces under identical experiment conditions (Table 1): a custom Hines/Newton-Raphson solver and *Brian2* for reference. Alongside the investigation of the behaviour of membrane trace characteristics, the spike times of the soma compartment are compared. The results are shown in Figure 4 for a runtime of $200\,\mathrm{ms}$ (left plot) and $800\,\mathrm{ms}$ (right plot) with an integration time step of $0.01\,\mathrm{ms}$.

Forward Euler (Brian2)

First, the *Brian2* simulator employing an explicit Forward Euler scheme is regarded. To investigate soma spikes, $I_{syn,s} = 280\,\mathrm{pA} = const.$ is set. The spike threshold is set at $V_{cut} = V_{exp} + 5\Delta_{exp} = -40\,\mathrm{mV}$. If a spike occurs, a refractory period of $t_{ref} = 5\,\mathrm{ms}$ is introduced and a spike increment of $b = 100\,\mathrm{pA}$ is applied to w. A full list of initialization values of all parameters is shown in Table 1 (see additional material). The $200\,\mathrm{ms}$ simulation produces soma spikes at $26.3\,\mathrm{ms}$ and $129.4\,\mathrm{ms}$. Characteristics like the exponential depolarization before spiking or t_{ref} can be observed distinctively.

Implicit Euler (Hines/ Newton-Raphson)

The second simulation uses the custom solver which implements a Newton-Raphson scheme. For the discretization, Backward (Implicit) Euler is used. The solver is built in *PyTorch* with full support for gradient backpropagation using soft spiking and surrogate gradients. Experiment setup and parameters configuration are copied from the previous simulation with *Brian2*. Soft spiking disabled for a direct comparison with the *Brian2* reference. For a runtime of $200\,\mathrm{ms}$, soma spikes are observed at $26.3\,\mathrm{ms}$ and $129.3\,\mathrm{ms}$. Noticeably, the second spike event happens $0.1\,\mathrm{ms}$ earlier than in the reference. There is no visible difference to the reference for the aforementioned characteristics in the membrane traces, which closely follow the ones simulated with *Brian2* at all times.

Long-run comparison

To further examine differences in spike timing, the experiment setup is copied for both solvers and re-run over an extended simulation runtime of $800\,\mathrm{ms}$. The results are shown in the right plot of Figure 4. A subtle increase of the difference in spike times can be observed, exceeding $0.1\,\mathrm{ms}$ (maximum difference in the simulation over $200\,\mathrm{ms}$). The largest difference to be observed is $0.3\,\mathrm{ms}$ for the last occurring spike.

Spike-based Input

Spike-based inputs have higher biological plausibility and mimic more closely how neurons are stimulated on the BrainScales-2 hardware [7]. For the simulations, spike input of $1000\,\mathrm{pA}$ is injected to the apical dendrite compartment. Selected events are also injected into the soma at $300\,\mathrm{pA}$, with feedforward shift of $10\,\mathrm{ms}$. The parameters are now initialized with the values listed in Table 2. This simulation is also conducted over $200\,\mathrm{ms}$ using a simulation step of $dt = 0.01\,\mathrm{ms}$. Figure 5 shows the results. The first soma spike differs by $0.1\,\mathrm{ms}$, the second spike time matches perfectly. As in the previous experiment, both simulations seem to produce the same voltage traces for the individual compartments.

Soft Spiking/ Gradient Computation

For testing gradient support, the custom solver is again subjected to the same spike-based input as in previous simulation and regarded over $50\,\mathrm{ms}$ to closer examine the first soma spike. The time constant of the adaption current τ_w is slightly perturbed as shown in Figure 6, resulting in a slightly delayed spike timing. A gradient-based parameter optimization task is set up around the training of τ_w to produce the initial spike time. For this, τ_w is wrapped as a *torch.nn.Parameter* to make it trainable. Furthermore, *requires_grad=True* needs to be set inside the training loop. The loss function is chosen as the Mean Squared Error (MSE) of the difference in spike timing:

$$\mathcal{L} = \left(t_{\mathrm{out}} - t_{\mathrm{ref}}\right)^2. \tag{14}$$

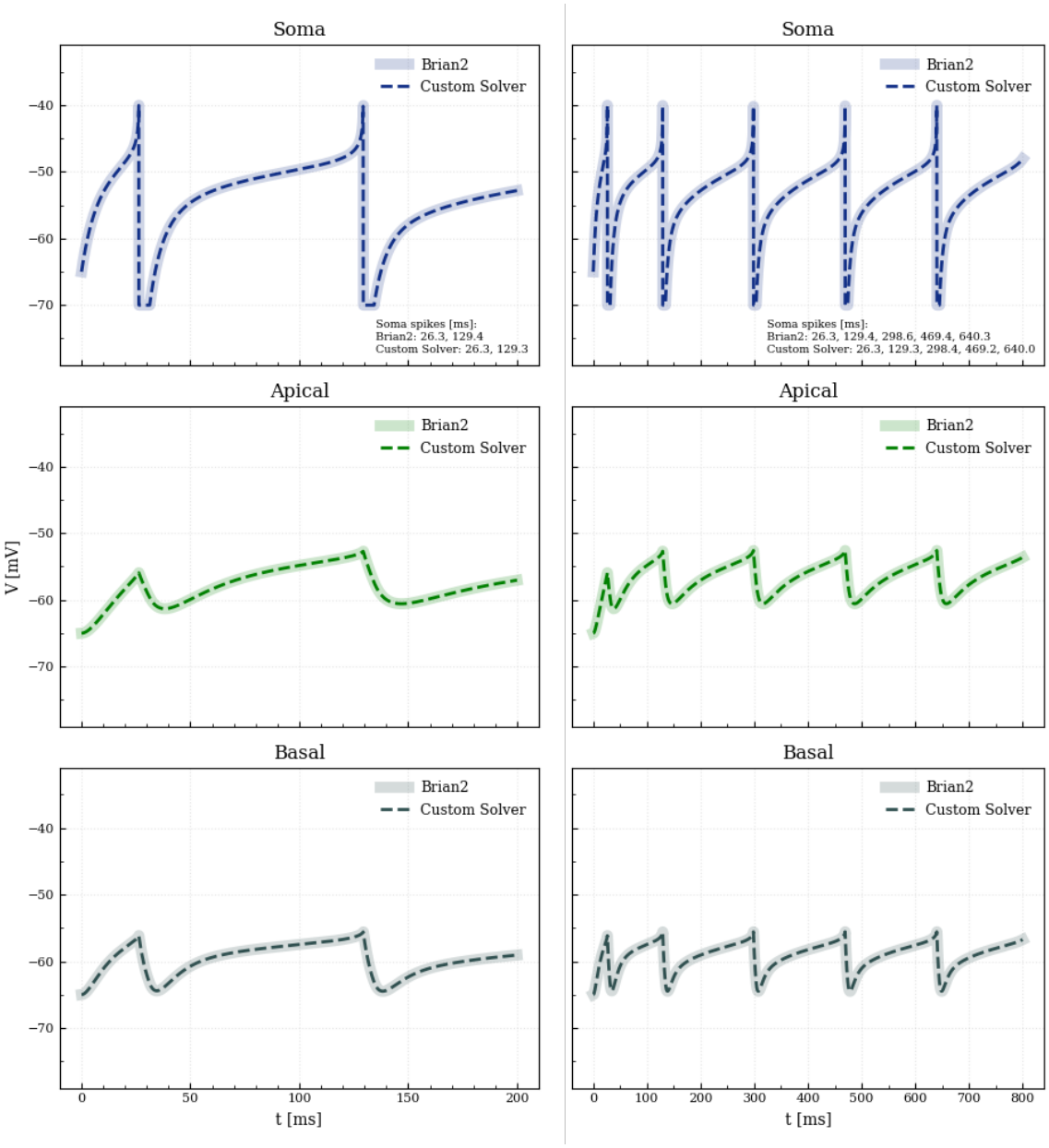

Figure 4: Left: Simulation of voltage traces with custom solver and Brian2 as a reference. Both simulations use the same neuron parameters, listed in Table 1. Also, both simulations are conducted over 200 ms runtime with a time step of 0.01 ms and deploy explicitly synaptic input current into the soma ($I_{syn,s} = 280\,\text{pA} = const.$). Right: Same simulation over 800 ms runtime.

For this, t_{out} and t_{ref} are pulled from the training model with $\tau_w = 50\,\text{ms}$ and the reference model with $\tau_w = 100\,\text{ms}$, respectively. To enable backpropagation, a Straight-Through-Estimator (STE) is defined that approximates the spike via a sigmoid function in the backward pass. The sigmoid gradient is of the form

$$\begin{aligned} \frac{d\sigma_\beta(x)}{dx} &= \beta \cdot \sigma_\beta(x) \cdot (1 - \sigma_\beta(x)), \\ \sigma_\beta(x) &= \frac{1}{1 + e^{-\beta x}} \end{aligned} \tag{15}$$

with $\beta = 50.0$. For the discrete event of the soma spike, a soft spike function is defined by computing a weighted average of the spike train entries over time. In the training loop, a standard ADAM optimizer [6] is used. After 100 epochs, a training value of $\tau_w = 90.90\,\text{ms}$ is determined and the learned spike timing is

$27.07\,\mathrm{ms}$. This already closely approximates the ground truth of $\tau_w = 100\,\mathrm{ms}$ with a corresponding spike time of $27.01\,\mathrm{ms}$. Figure 7 shows visualizations of the training procedure.

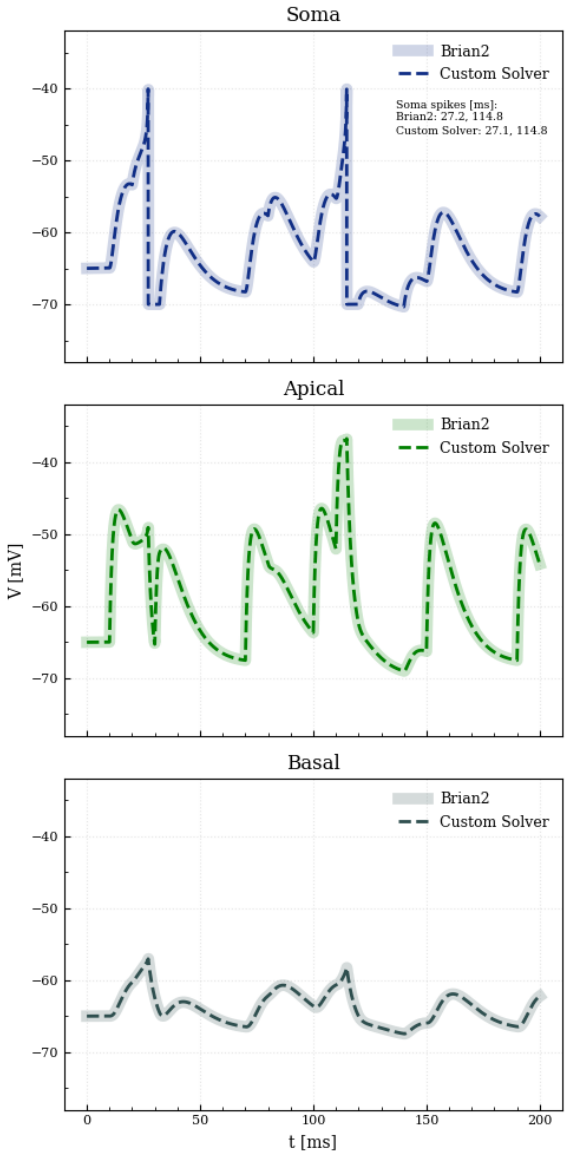

Figure 5: Simulation of the 3-compartment neuron when stimulated using spike-based input. The simulation is conducted over $200\,\mathrm{ms}$ with a time step of $0.01\,\mathrm{ms}$. Spikes are injected into the apical dendrite compartment at $t \in \{10, 30, 70, 100, 110, 150, 190\}\,\mathrm{ms}$. Selected input events are also injected into the soma compartment with a feedforward shift of $10\,\mathrm{ms}$ at $t \in \{20, 80, 140\}\,\mathrm{ms}$. Again, the custom solver is compared to *Brian2* as reference.

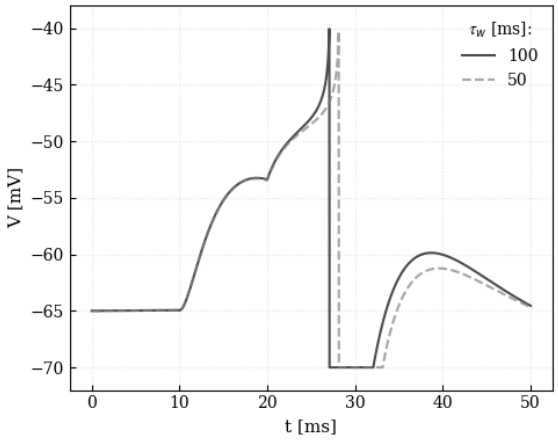

Figure 6: Times of the first soma spike from Figure 5 before and after perturbation of τ_w.

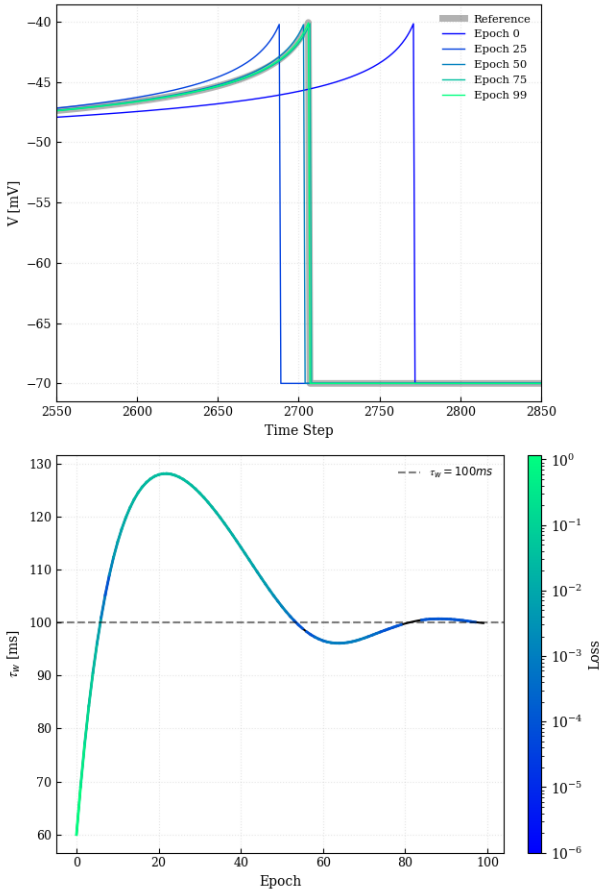

Figure 7: Top: Times of the first soma spike for selected epochs. Bottom: Training values of τ_w colored by the loss value determined using equation (16).

The results show that gradient descent is implemented faithfully. After an initial overshooting, the loss forces the training parameter to approach the ground truth. This results in a controlled learning of the correct spike timing.

Specification of Methods

For the custom Hines/Newton-Raphson solver, it is to note that the update of the adaptation current is realized semi-explicitly as

$$w_{t+1} = w_t + \Delta t \cdot \left(\frac{a(V_s^{t+1} - V_\ell) - w_t}{\tau_w} \right). \tag{16}$$

In simulations using spike-based stimulation, input currents are subjected to a decay time constant of $5\,\mathrm{ms}$ in the form of

$$\frac{dI_{\text{syn}}}{dt} = -\frac{I_{\text{syn}}}{5\,\mathrm{ms}}. \tag{17}$$

Discussion

The custom solver developed in this internship seems to faithfully reproduce the voltage traces generated with *Brian2*, an established tool for simulating multi-compartmental neuron models. In simulations over $200\,\mathrm{ms}$ runtime, differences in spike timing do not exceed $0.1\,\mathrm{ms}$ for all types of synaptic input tested. These differences are considered as negligible. Larger differences in spike timing were able to be observed for $I_{syn,s} = 280\,\mathrm{pA} = const.$ over a runtime of $800\,\mathrm{ms}$. Here, a gradual increase of the $0.1\,\mathrm{ms}$ reached $0.3\,\mathrm{ms}$ for the last occurring spike. This is likely due to the constant synaptic input, not indicative of any serious flaws in the custom solver.

The surrogate gradient in the backpass of the STE defined for the training of τ_w is rather complex. With setting $\beta = 50$, the sigmoid is modeled to closely follow the step function of the spikes. This is nice for the approximation, but expensive in computation. To render the simulation more efficient, replacing the sigmoid by a linear function should be considered.

Outlook

In order to use it for other than the specified dynamics, further generalization of the custom solver is in order. Further, computational efficiency and thus overall simulation duration is aimed to be improved. It is also planned to explore *JAX*-based implementations. One possible pathway for this would be to define a custom AdEx channel and a differentiable spike mechanism for *JAXLEY*, an established *JAX*-based simulation tool for simulating multi-compartmental neuron models. Another option would be to rewrite the custom solver in *JAX* and implement desired functionality manually. After this, the first step of the bachelor thesis following this internship will be the gradient-based training of multi-compartmental neuron to the neuromorphic hardware on BSS-2. For this, relating simulation parameters to hardware parameters will need to be examined. The overarching aim is to add gradient based, in-the-loop training of multi-compartmental neuron models to the BSS-2 software stack (depending on the final version of the solver either in *hxtorch* or *jax-snn*). This will allow to replicate the behaviour of complex biological neurons in a new setting of gradient-based learning. If this is still possible in the scope of the bachelor thesis, this would then be moved to the training of networks consisting of multiple multi-compartmental neurons.

References

[1] Michael Deistler et al. “Differentiable simulation enables large-scale training of detailed biophysical models of neural dynamics”. In: *bioRxiv* (2024). DOI: 10.1101/2024.08.21.608979.

[2] Michael L. Hines. “Efficient computation of branched nerve equations”. In: *International Journal of Bio-Medical Computing* 15.1 (1984), pp. 69–76. DOI: 10.1016/0020-7101(84)90008-4.

[3] A.L. Hodgkin and A.F. Huxley. “A quantitative description of membrane current and its application to conduction and excitation in nerve”. In: *Journal of Physiology* 117 (1952), pp. 500–544.

[4] Simon Jonscher. “Gradient-Based Learning of Neuron Parameters in Spiking Neural Networks on Neuromorphic Hardware”. Masterarbeit. Universität Heidelberg, 2025.

[5] Jakob Kaiser et al. “Emulating Dendritic Computing Paradigms on Analog Neuromorphic Hardware”. In: *Neuroscience* 489 (2022), pp. 290–300. ISSN: 0306-4522. DOI: https://doi.org/10.1016/j.neuroscience.2021.08.013.

[6] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: (2017). arXiv: 1412.6980 [cs.LG].

[7] Christian Pehle et al. “The BrainScaleS-2 accelerated neuromorphic system with hybrid plasticity”. In: *Frontiers in Neuroscience* 16 (2022), p. 795876.

[8] Panayiota Poirazi, Terrence Brannon, and Bartlett Mel. “Pyramidal Neuron as Two-Layer Neural Network”. In: *Neuron* 37 (Apr. 2003), pp. 989–99. DOI: 10.1016/S0896-6273(03)00149-1.

[9] William H Press et al. *Numerical Recipes: The Art of Scientific Computing*. 3rd. Cambridge University Press, 2007. ISBN: 9780521880688.

[10] W. M. Yamada, Christof Koch, and P. R. Adams. “Multiple Channels and Calcium Dynamics”. In: *Methods in Neuronal Modeling: From Ions to Networks*. Ed. by Christof Koch and Idan Segev. Cambridge, MA: MIT Press, 1989, pp. 97–134.

Additional Material

Symbol	Description	Value
C_s	Soma capacitance	$125\,\mathrm{pF}$
C_a	Apical dendrite capacitance	$100\,\mathrm{pF}$
C_b	Basal dendrite capacitance	$50\,\mathrm{pF}$
$g_{\ell,s}$	Soma leak conductance	$10\,\mathrm{nS}$
$g_{\ell,a}$	Apical leak conductance	$2\,\mathrm{nS}$
$g_{\ell,b}$	Basal leak conductance	$5\,\mathrm{nS}$
V_{leak}	Leak reversal potential	$-65\,\mathrm{mV}$
V_{exp}	Exponential spike threshold	$-50\,\mathrm{mV}$
Δ_T	Exponential slope factor	$2\,\mathrm{mV}$
a	Subthreshold adaptation	$4\,\mathrm{nS}$
τ_w	Adaptation time constant	$100\,\mathrm{ms}$
V_{reset}	Reset potential	$-70\,\mathrm{mV}$
V_{cut}	Spike cutoff threshold	$V_{\text{exp}} + 5 \cdot \Delta_T$
g_{as}	Soma–apical coupling	$5\,\mathrm{nS}$
g_{bs}	Soma–basal coupling	$5\,\mathrm{nS}$
$I_{\text{syn,s}}$	Somatic input current	$280\,\mathrm{pA}$
$I_{\text{syn,a}}$	Apical input current	$0\,\mathrm{pA}$
$I_{\text{syn,b}}$	Basal input current	$0\,\mathrm{pA}$

Table 1: Neuron parameters used in simulations with $I_{syn,s} = const.$

Symbol	Description	Value
C_s	Soma capacitance	$120\,\mathrm{pF}$
C_a	Apical dendrite capacitance	$60\,\mathrm{pF}$
C_b	Basal dendrite capacitance	$40\,\mathrm{pF}$
$g_{\ell,s}$	Soma leak conductance	$10\,\mathrm{nS}$
$g_{\ell,a}$	Apical leak conductance	$8\,\mathrm{nS}$
$g_{\ell,b}$	Basal leak conductance	$5\,\mathrm{nS}$
V_{leak}	Leak reversal potential	$-65\,\mathrm{mV}$
V_{exp}	Exponential spike threshold	$-55\,\mathrm{mV}$
Δ_T	Exponential slope factor	$3\,\mathrm{mV}$
a	Subthreshold adaptation	$4\,\mathrm{nS}$
τ_w	Adaptation time constant	$100\,\mathrm{ms}$
V_{reset}	Reset potential	$-70\,\mathrm{mV}$
V_{cut}	Spike cutoff threshold	$V_{\text{exp}} + 5 \cdot \Delta_T$
g_{as}	Soma–apical coupling	$30\,\mathrm{nS}$
g_{bs}	Soma–basal coupling	$5\,\mathrm{nS}$
τ_{syn}	Synaptic time constant	$5\,\mathrm{ms}$
$I_{\text{syn,a}}$	Apical spike input	$1000\,\mathrm{pA}$
$I_{\text{syn,s}}$	Somatic spike input	$300\,\mathrm{pA}$
$I_{\text{syn,b}}$	Basal spike input	$0\,\mathrm{pA}$

Table 2: Neuron parameters used in simulations using spike-based input