The role of membrane time constant in the training of spiking neural networks

Improving accuracy by per-neuron learning

Adam Pazderka[1]

Supervisors: Nergis Tömen[1]**, Aurora Micheli**[1]

[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 26, 2024

Name of the student: Adam Pazderka
Final project course: CSE3000 Research Project
Thesis committee: Nergis Tömen, Aurora Micheli, Lilika Markatou

Abstract

Spiking neural networks (SNNs) aim to utilize mechanisms from biological neurons to bridge the computational and efficiency gaps between the human brain and machine learning systems. The widely used Leaky-Integrate-and-Fire (LIF) neuron model accumulates input spikes into an exponentially decaying membrane potential and generates a spike when this potential exceeds a set threshold. A LIF neuron is characterized by learnable input weights and a manually selected membrane time constant τ, which determines the decay rate of a neuron's membrane potential. Previous work introduced the Parametric LIF (PLIF) neuron model with a learnable τ. However, the published experiments only featured a single τ per spiking layer. This leaves space for exploration of the effect of having a learnable τ for each neuron. The importance of τ is given by the trade-off it inherently introduces, prioritizing the neuron's capability to convey information about spatial or temporal features. This work examines the effect of introducing a learnable τ per neuron with a new initialization method and a new regularization term which incentivizes low variance in each PLIF layer. The experiments are done using the DVS128 Gesture dataset and compared to a baseline model from the original paper introducing the PLIF neuron model. Results are inconclusive but suggest that introducing τ per neuron does not have a significant effect on the accuracy of a spiking neural network. Moreover, the evolution of τ during training exhibits interesting behavior and leads to two new hypotheses.

1 Introduction

Power-efficient machine learning is positioned to be dominated by the emerging brain-inspired spiking neural networks (SNNs). Together with specialized neuromorphic hardware [1; 2; 3; 4], SNNs have shown to perform comparably to artificial neural networks (ANNs) while consuming two or more orders of magnitude less power [5; 6; 7]. Several applications in robotics and medicine have already benefited from the enabled low-power capabilities [8; 9; 10; 11]. However, ANNs are still better suited for a majority of tasks, and the competitiveness of SNNs in other areas than power-efficiency and latency remains unclear [12, p. 1040].

The main characteristic of SNNs is that they are temporal in nature and consist of spiking neurons. These neurons communicate using weighted binary (1-bit) spikes and perform computation using their internal state called *membrane potential*. Biological neurons can be modeled in many ways with a varying trade-off between computational cost and biological accuracy. A commonly used neuron model in the context of power-efficient deep learning is the Leaky-Integrate-and-Fire (LIF) neuron, which accumulates weighted input spikes into its exponentially decaying membrane potential [12, p. 1023]. If enough input spikes increase the membrane potential above a set threshold, its value resets, and a spike is emitted.

Input weights are the only learnable parameters in the highly simplified LIF neuron model. Experiments with learnable decay rates and adaptive firing thresholds have shown that extending the LIF neuron model with additional biologically-inspired mechanisms can lead to both faster training and better performance [13; 14]. This work focuses on the Parametric Leaky-Integrate-and-Fire (PLIF) neuron model with a learnable membrane time constant τ, which determines the decay rate of a neuron's membrane potential. More concretely, it explores the effect of introducing a learnable τ per neuron as opposed to having a single learnable τ per layer as in the PLIF neuron paper [14]. More concretely, the research question explored by this paper is: **"What is the effect of having a learnable membrane time constant per neuron on the accuracy of a spiking neural network?"**.

The membrane time constant τ plays an important role in a neuron's ability to convey information about spatial and temporal features. For example, in the case of $\tau = 0$, a neuron's membrane potential would instantly decay after receiving input, making it possible to only output a spike based on a combination of input spikes that all arrive at the same time. Conversely, in the case of $\tau = +\infty$, a neuron's membrane potential would not decay at all. Due to this dynamic, it can be seen as analogous to the function of the "forget gate" in LSTMs [15], which have recently been shown to get in language modelling "At least as far as current technologies like Transformers or State Space Models" [16, Sec. 6]. The "forget gate" analogy is explored in more detail in Section 2.3.

In the paper on PLIF neurons, having a single learnable τ per layer is believed to be biologically plausible since "the neighboring neurons have similar properties" [14, Sec. 3.3]. However, following the "forget gate" analogy, τ should be a parameter that can be trained for each neuron individually. Similar to having a trainable forget gate per unit in an LSTM. A new problem then arises, since this can introduce overfitting due to the increased number of parameters. This is addressed by a newly introduced regularization term.

The experiments utilize the DVS128 Gesture dataset [7] and the network structure from [14]. Four models are trained and analyzed: First, the baseline model with a single learnable tau in each layer is replicated, where every tau is initially set to $\tau_0 = 2$. Second, the "No Regularization" model is trained, featuring a trainable membrane time constant per neuron and a log-normal initialization of τ. Then an additional regularization term, in the form of $\lambda * Var(\tau_l)$ per spiking layer l, is introduced in the third and fourth models, with coefficients $\lambda = 0.01$ and $\lambda = 0.1$, respectively. This regularization approach follows from the assumption that setting λ to a high value should make a model perform at least as well as the baseline. Furthermore, experiments with varying λ values should enable interpolation between the two approaches, leading to a more controlled and incremental approach over the existing research.

Experimental results suggest that **having a membrane time constant per neuron instead of per layer does not have a significant effect on final accuracy**. However, this conclusion needs to be treated with low confidence, and fur-

ther experiments across more datasets, ideally with multiple initializations per model, are needed. Furthermore, two new hypotheses are formed based on the interesting evolution of τ during training;

- "*There is an inverse relation between the number of τ parameters and their learning rate*", and
- "*There is a general relationship between the learning capacity of a spiking layer and the number of learnable τ and input weight parameters*"

Similarly, four new areas for future work are identified.

This work is structured as follows; Section 2 provides the necessary background. It introduces the PLIF neuron model, motivates the case for having τ per neuron, and describes the replicated baseline model including its implementation framework. Section 3 describes new initialization and regularization methods for τ. The three newly introduced models are then presented. Section 4 describes the experimental setup and analyses the results. Replicability and usage of LLMs are discussed in Section 5. Section 6 further discusses the results and the newly created hypotheses, while Section 7 outlines potential areas for future work. The conclusion is presented in Section 8.

2 Background

This section provides a background for the methodology and experiments conducted in Section 4. It begins with Section 2.1, introducing the Leaky-Integrate-and-Fire (LIF) neuron model, followed by Section 2.2, which explains the Parametric LIF (PLIF) neuron model and introduces learnable parameters for a numerically stable way to indirectly optimize the membrane time constant. An analogy between the PLIF neuron's membrane time constant and the forget gate in Long Short-Term Memory (LSTM) units is discussed in Section 2.3. The SpikingJelly framework used for experiments is described in Section 2.4, and the baseline model used to replicate the experiment from [14] is detailed in Section 2.5.

2.1 LIF neuron

The Leaky-Integrate-and-Fire (LIF) neuron model used throughout Section 4 can, in its charging state, be expressed using Eq. 1, where τ denotes the membrane time constant, $V(t)$ denotes the membrane potential at time t, $X(t)$ denotes the input at time t, and V_{rest} the resting membrane potential.

$$\tau \frac{dV(t)}{dt} = -(V(t) - V_{rest}) + X(t) \tag{1}$$

Solving this differential equation for $V(t)$ when the input is zero gives a function that exponentially decays towards V_{rest}. The resulting decay rate is expressed as $\frac{1}{\tau}$, hence the relation between the membrane time constant and the decay rate.

The LIF neuron equation follows from empirical observations and subsequent successful modelling of a neuron as a low-pass filter circuit consisting of a resistor R and a capacitor C [12, p. 1021]. In the context of circuits, the membrane time constant τ is referred to as the RC time constant. The intuition behind τ is that it is the duration after which a capacitor charge (membrane potential in the case of a neuron) decays by $1 - e^{-1} \approx 63.2\%$ of its original value.

While many different neuron models exist, with a varying degree of biological accuracy [17], the LIF neuron is often considered to be a good choice for deep learning, having the right trade-off between complexity and computational cost [18; 19].

In addition to the charging state, a LIF neuron is defined by its behavior when the membrane potential $V(t)$ reaches the threshold V_{th}. In such case, a spike is fired, expressed as $S(t) = 1$, and the neuron's membrane potential is reset to the reset potential V_{reset}. The behavior of firing a spike and the subsequent reset can be formally described using Eq. 2 and Eq. 3 taken from [19].

$$S(t) = \Theta(V(t) - V_{th}) \tag{2}$$

$$\lim_{\Delta t \to 0^+} V(t + \Delta t) = V_{reset} \tag{3}$$

Where $\Theta(x)$ is the Heaviside step function defined by $\Theta(x) = 1$ for $x \geq 0$ and $\Theta(x) = 0$ for $x < 0$.

All neurons in the replicated baseline model from [14] as well os other models this paper, use the "hard reset" method described by Eq. 3 and values $V_{reset} = V_{rest} = 0$ and $V_{th} = 1$ [20].

Simulation using discrete-time differential equations

The continuous dynamics of a LIF neuron can be simulated on digital hardware using discrete-time differential equations. Reducing the time-discretized general equations in [19, Eq. 5-7] to a case with only the LIF neurons and substituting the constants yields three time-discretized equations.

$$\tau \left(V'[t] - V[t-1]\right) = -V[t-1] + X[t] \tag{4}$$

$$S[t] = \Theta(V'[t] - 1) \tag{5}$$

$$V[t] = V'[t] \cdot (1 - S[t]) \tag{6}$$

The equations 4, 5, and 6 describe neuron's charging, spiking, and resetting behavior respectively. $V'[t]$ denotes an intermediate membrane potential after input processing but before a potential reset. At a single time step, $V'[t]$ is first updated using Eq. 4. Then spiking is determined by Eq. 5 and if a spike occurs, $V[t]$ is reset using Eq. 6. To numerically solve these equations, the first-order Euler method is used for all models in this paper as part of the SpikingJelly framework [19, p. 9].

Trainable parameters

The LIF neuron receives weighted input spikes from all its connections. This is where optimization comes in since these input weights can be updated via backpropagation and enable learning. In the previous equations, the weighted input spikes are hidden in the $X(t)$ term. To make them explicit, we can write:

$$X(t) = \sum_i w_i \cdot x_i(t), \quad \text{where } x_i(t) \in \{0, 1\},\ w_i \in \mathbb{R}$$

The term $x_i(t)$ represents a single binary connection that is either spiking (1) or resting (0) at time t. The connection's weight is denoted as w_i. More details on backpropagation using the surrogate gradient method can be found in Section 2.4

2.2 PLIF neuron

The Parametric LIF neuron, introduced in [14], enables numerically stable optimization of the membrane time constant.

A naive approach to a learnable τ would be optimizing it directly. However, after rearranging Eq. 4 to solve for $V'[t]$, we can see that τ is in the denominator and would cause numerical instability when outside the interval $[1, \infty)$, since we are dealing with discretized time.

$$V'[t] = V[t-1] + \frac{1}{\tau}(-V[t-1] + X[t]) \tag{7}$$

This problem is solved by parameterizing τ using a newly introduced parameter a. The parameter a and τ are then related by a clamp function $k(a) \in (0, 1)$, which is substituted in place of $\frac{1}{\tau}$, i.e. $\tau = \frac{1}{k(a)} \in (1, +\infty)$. The clamp function used in the experiments is the sigmoid activation function [14]. I.e. $k(a) = \frac{1}{1+\exp(-a)}$, resulting in Eq. 8.

$$V'[t] = V[t-1] + \frac{1}{1+\exp(-a)}(-V[t-1] + X[t]) \tag{8}$$

The parameter a can now be used to optimize τ in a numerically stable way.

For the derivation of the gradient equations, see [14, Sec. 3.7].

2.3 The "Forget gate" analogy

The PLIF neuron paper features a single learnable τ per layer, arguing for its biological plausibility since "neighboring neurons have similar properties" [14]. However, an analogy can be found between τ and the forget gate in the highly successful LSTMs [16]. Conversely, in an LSTM, each unit (analogous to a spiking neuron) has a learnable forget gate that determines how much of the previous state is retained, similar to the role of τ, suggesting that introducing a learnable τ for each neuron is worth exploring.

To illustrate this similarity, let's look at Eq. 9 and Eq. 11 from [16], describing a case of an LSTM memory cell with scalar variables. A memory cell stores its information in a cell state c_t and a hidden state h_t. The hidden state is a function of the cell state and is omitted for simplicity. The cell state c_t is updated by adding a previous cell state c_{t-1} multiplied by a forget gate f_t to a processed input z_t multiplied by an input gate i_t.

$$c_t = f_t c_{t-1} + i_t z_t \tag{9}$$

Rearranging Eq. 4 shows a clear similarity. When computing a new $V[t]$, the term $\frac{1}{\tau}$ linearly interpolates between the previous state and the current input. This analogy is also briefly touched upon in [14].

$$V'[t] = \left(1 - \frac{1}{\tau}\right) V[t-1] + \frac{1}{\tau} X[t] \tag{10}$$

The case for exploring learnable τ per individual neuron is made even stronger by the fact that even in such a case, the complexity of a LIF neuron still remains much lower than that of an LSTM unit. We can see from Eq. 11, where w_x, w_h, b are learnable parameters and σ is the sigmoid activation function, that the forget gate is not a trained parameter, but a trained function, dependent on the current input x_t and the previous hidden state h_{t-1}.

$$f_t = \sigma(w_x^\top x_t + w_h h_{t-1} + b) \tag{11}$$

2.4 SpikingJelly framework

All experiments in this paper, including the replicated baseline mode, use the PyTorch-based SpikingJelly framework, an open-source deep learning framework bridging deep learning and SNNs. The SpikingJelly framework was created with three characteristics in mind [19]:

- exploits and accelerates spike-based operations
- supports both simulations on CPUs/GPUs and deployment on neuromorphic chips
- and provides a full-stack toolkit for building, training, and analyzing deep SNNs

SpikingJelly shares a lot of similarities with the snnTorch [12] and Norse [21] frameworks, which are also PyTorch-based and represent viable alternatives. However, this work builds on top of an already existing codebase implemented using SpikingJelly.

Surrogate gradients

Backpropagation through the Heaviside function $\Theta(x)$ used in Eq. 5 is not possible since its derivative is 0 at $x \neq 0$ and ∞ at $x = 0$. The SpikingJelly framework offers the surrogate gradient method to solve this, using a smooth and differentiable surrogate function to define the derivative of $\Theta(x)$ during backpropagation. Specifically, the sigmoid $\sigma(x) = \frac{1}{1+\exp(-\alpha x)}$ with a hyperparameter α is used [19, p. 10].

2.5 Baseline model for the DVS128 Gesture dataset

The network structure for all models is the same as the replicated baseline, except for the changes made to the PLIF neuron layer to support having a learnable membrane time constant per each neuron instead of each layer.

> "*c128k3s1* represents the convolutional layer with *output channels* 128, *kernel size* = 3 and *stride* = 1. BN is the batch normalization. *MPk2s2* is the max-pooling layer with *kernel size* = 2 and *stride* = 2. *PLIF* is the PLIF spiking neurons layer. *DP* represents the dropout layer. *FC2048* represents the fully connected layer with *output features* = 2048. The symbol {}* indicates the repeated structure." [20, Sec. 2]

The structure used for all models (see Appendix B):

> {c128k3s1-BN-PLIF-MPk2s2}*5-DP-FC512-PLIF-DP-FC110-PLIF-APk10s10

All parameters, except the ones used to optimize the membrane time constants, are initialized implicitly, using the default PyTorch initialization.

The training and validation datasets are obtained by dedicating 85% of samples of each class in the original training set as the new training set and using the rest 15% as the validation set [20].

Input preprocessing and encoding

The events from the event-based DVS128 Gesture dataset are split into $T = 20$ slices with nearly the same number of events in each slice and integrated into frames. The exact method can be found in [20, Sec. 5].

Instead of using a standard input encoding algorithm, " the input is directly fed to the network without being first converted to spikes and the image-spike encoding is done by the first Conv2d-Spiking Neurons module (BN is omitted), which can be seen as a learnable encoder" [20, Sec. 10].

Hyperparameters

Except for the batch size, as discussed in Section 4.2, all hyperparameters are taken from [20].

> "We use the Adam optimizer with a learning rate of 0.001 and the cosine annealing learning rate schedule with $T_{\text{schedule}} = 64$. The batch size is set to 16 to reduce memory consumption. The drop probability p for dropout layers is 0.5. The clamp function for PLIF neurons is $k(a) = \frac{1}{1+e^{-a}}$ and the surrogate gradient function is $\sigma(x) = \frac{1}{\pi}\arctan(\pi x)+\frac{1}{2}$, thus $\sigma'(x) = \frac{1}{1+(\pi x)^2}$. We set $V_{\text{reset}} = 0$ and $V_{\text{th}} = 1$ for all neurons." [20, Sec. 6]

Loss function

The loss function used for all models is taken from [14].

> "Denote the simulating time-steps as T and the number of classes as C. The output $O = [o_{t,i}]$ is a $C \times T$ tensor. For a given input with label l, we encourage the neuron that represents class l to have the highest excitatory level while other neurons should remain silent. So the target output is defined by $Y = [y_{t,i}]$ with $y_{t,i} = 1$ for $i = l$ and $y_{t,i} = 0$ for $i \neq l$. The loss function is defined by the mean squared error (MSE) $L = \text{MSE}(O, Y) = \frac{1}{T}\sum_{t=0}^{T-1} L_t = \frac{1}{T}\sum_{t=0}^{T-1}\frac{1}{C}\sum_{i=0}^{C-1}(o_{t,i} - y_{t,i})^2$. And the predicted label l_p is regarded as the index of the neuron with the maximum firing rate $l_p = \arg\max_i \left(\frac{1}{T}\sum_{t=0}^{T-1} o_{t,i}\right)$" [14, Sec. 3.7].

3 Methodology

In this section, we outline the methodology used to replicate and extend the baseline model from Section 2.5. First, we replicated the baseline model to verify our implementation, achieving results close to those reported in the original paper. The baseline model is then used as a foundation for new models with modified PLIF neuron layers, each featuring individual membrane time constants τ per neuron. Section 3.2 details the initialization of the τ parameters using a log-normal distribution, while Section 3.3 explains the variance-based regularization term, enabling an incremental approach over the baseline. These subsections set the stage for Section 4.

3.1 Replication and extension of the baseline model

To ensure a sound implementation of the new models, we first replicated the baseline model described in Section 2.5.

The replicated model with a 15% validation set split was expected to achieve a peak test set accuracy of 96.53% [20, Table S2]. However, our model achieved a peak test set accuracy of 95.5%. This may have been caused by changing the batch size during training due to interruptions and availability of different GPUs (16, 24, and 32) as opposed to keeping a batch size of 16 according to Section 2.5. Another possibility is that the dataset loading method used in the original experiment differed from the one provided in the published implementation as noted in Section 4.1. Considering these possibilities and the very slight deviation from the expected accuracy, the replication was considered successful.

The three new models were built by extending the baseline with PLIF neuron layers that instead of having a single τ per layer have a τ per each neuron. This introduced a need for new initialization and regularization methods as discussed in Section 3.2 and 3.3.

3.2 τ initialization

Parameter initialization can have a significant impact on the learning process of a neural network [22; 23; 24]. However, no research has been found about the collective initialization of membrane time constants of a population of neurons at the time of writing of this paper. Furthermore, rigorously devising an initialization method is outside of the scope of this work. Nevertheless, previous research suggests that initial parameters in a layer should be drawn from a distribution instead of being all set to a single value [24].

This section follows the reasoning from Section 2.2, where τ is parameterized using a according to Eq. 12.

$$\tau = \frac{1}{k(a)} \in (1, +\infty), \quad \text{where } k(a) = \frac{1}{1 + \exp(-a)} \tag{12}$$

The optimized parameters a are initialized via τ using a log-normal distribution shifted by $+1$, resulting in samples correctly lying in the interval $(1, +\infty)$ as discussed in Section 2.2. The reasoning behind initializing a indirectly is that future research may build upon this work and parameterize τ differently. The indirect initialization is more robust to changes in methodology and potentially prevents unnecessary work.

Sampling from a log-normal distribution

The initial τ parameters are sampled from a log-normal distribution. Given a mean μ_τ and standard deviation σ_τ for the desired distribution of τ, we can compute parameters of the underlying normal distribution using $\mu'_\tau = \mu_\tau - 1$ and Eq. 13.

$$\mu_{\log} = \ln\left(\frac{{\mu'_\tau}^2}{\sqrt{\sigma_\tau^2 + {\mu'_\tau}^2}}\right), \quad \sigma_{\log} = \sqrt{\ln\left(1 + \frac{\sigma_\tau^2}{{\mu'_\tau}^2}\right)} \tag{13}$$

Using these, a sample τ from the log-normal distribution is generated where $\tau = 1 + e^Y$ and $Y \sim N(\mu_{\log}, \sigma_{\log}^2)$. This sample τ, which lies in the interval $(1, +\infty)$, can then be transformed to the directly trained parameters A using Eq. 14 derived from Eq. 12.

$$A = -\ln(\tau - 1) \tag{14}$$

This maps $\tau \in (1, +\infty)$ to $A \in (-\infty, +\infty)$.

3.3 τ regularization

Recurrent neural networks (RNNs) have been shown to benefit from different regularization methods than feedforward neural networks [25]. Similarly, properly regularizing spiking neural networks with learnable membrane time constants may require novel research.

Thorough research on the regularization of SNNs with a learnable τ per neuron is outside of the scope of this work. However, introducing a large number of new parameters may lead to overfitting and needs to be addressed. Therefore, this work adopts an incremental approach over the existing research discussed in Section 2 and adds a new regularization term designed for this purpose.

The regularization term follows from the assumption that the baseline model rule, which constrains all membrane time constants in a layer to a single shared parameter, can be adapted to a model where each neuron has its own learnable τ without degrading the model's learning capacity. For this purpose, a regularization term, $\lambda * \text{Var}(\tau_l)$, is added to the loss function for each spiking layer l. It is hypothesized that a high value of λ will keep the membrane time constants together while not harming performance, effectively simulating the baseline model. Lowering λ can then be seen as a sort of interpolation between the former and the newly introduced approach to the optimization of τ.

Using this approach, a "control" model with a high value of $\lambda = 0.1$ is introduced. This model is expected to behave similarly to the baseline, in which case it will be easier to analyze the effect of lowering λ on the learning process and the model's accuracy. In the end, three different models are analyzed, which is discussed in Section 4.2.

Later experiments show that the "$\lambda = 0.1$" model indeed achieves a competitive performance. However, the learning process and values of its membrane time constants differ from the baseline model. More details can be found in Section 4.3.

The value $\lambda = 0.1$ was found empirically by searching for models with a high value of λ, such that the standard deviation of their membrane time constants for each layer stays below 0.05 within the first 20 epochs, while their validation set accuracy improves.

4 Experimental Setup and Results

This section presents the experimental setup and results, detailing the use of the DVS128 Gesture dataset in Section 4.1, introducing the baseline and extended models in Section 4.2, and analyzing their comparative performance in Section 4.3.

4.1 DVS128 Gesture dataset

The experiments are performed on the DVS128 Gesture dataset, which was collected using a 128x128 event-based camera, specifically to be used with an SNN [7].

> "The DvsGesture dataset comprises 1,342 instances of a set of 11 hand and arm gestures [...], grouped in 122 trials collected from 29 subjects under 3 different lighting conditions. During each trial, one subject stood against a stationary background and performed all 11 gestures sequentially under the same lighting condition. The gestures include hand waving (both arms), large straight arm rotations (both arms, clockwise and counterclockwise), forearm rolling (forward and backward), air guitar, air drums, and an "Other" gesture invented by the subject. The 3 lighting conditions are combinations of natural light, fluorescent light, and LED light, which were selected to control the effect of shadows and fluorescent light flicker on the DVS128. Each gesture lasts about 6 seconds." [7, Sec. 5.1]

The dataset is split as follows; 288 instances are dedicated for testing. The remaining 1056 instances are split into 156 instances (15%) for validation and 900 instances (85%) for training. The original paper states there are 1,342 instances. However, after downloading the dataset [7], processing it using the provided implementation, and loading it, a total of 1,344 instances were found. It is possible that the test accuracy discrepancy, mentioned in Section 3.1, was caused by some minor difference between the published preprocessing implementation and the one actually used.

The DVS128 Gesture dataset was chosen for the following reasons: it was used in one of the original experiments with the baseline model, enabling us to verify our implementation through replication; the stated peak test set accuracy of 96.53% left room for improvement; and the presence of seven PLIF neuron layers provided a greater chance of improved results than many other models if any of the layers benefited from having a τ per neuron.

4.2 Models

Baseline The baseline model from Section 2.5 uses a single learnable τ per layer, which is initialized as $\tau_0 = 2$. The purpose of this model is to verify the implementation before conducting experiments with the new models. This replication was considered successful, as discussed in Section 3.1.

No Regularization A single τ per layer is replaced with a unique τ per neuron. This model does not apply any regularization to the individual τ parameters. It is expected to suffer from overfitting, potentially achieving a worse accuracy than all the other models.

$\lambda = 0.01$ and $\lambda = 0.1$ On top of having a unique τ per neuron, a regularization term introduced into the loss function, in the form of $\lambda * \text{Var}(\tau_l)$ per spiking layer l. The "$\lambda = 0.1$" model serves as a "control" model. It is expected to behave similarly to the baseline model. The "$\lambda = 0.01$" is expected to show whether letting individual neurons optimize their τ increases final accuracy while avoiding potential overfitting.

Hyperparameters

Except for a batch size of 32, all hyperparameters from the baseline are also used in the new models. Details on the baseline model's hyperparameters are discussed in Section 2.5.

Initialization of the membrane time constant parameters in the new models uses $\mu = 2$ and $\sigma = 0.1$. This translates into an initialization of the optimized parameters around zero, where the sigmoid activation function's gradient is the largest. Further details on initialization are provided in Section 3.2.

4.3 Results

As seen in Table 1 and Figure 2, the results suggest that **having a membrane time constant per neuron instead of per layer does not have a significant effect on final accuracy**. Specifically, the models "$\lambda = 0.01$" and "$\lambda = 0.1$" outperform the baseline in terms of their peak test set accuracies by only 0.3% and 0.7% respectively. This minor improvement could easily be due to chance, considering the 1.03% difference between the baseline's expected and replicated accuracy discussed in Section 3.1. Therefore, this observation should be taken with caution and requires further validation across a more diverse set of datasets, such as CIFAR10-DVS [26], and through initialization with multiple different seeds.

Model	Test Accuracy	Val Accuracy
Baseline	95.5%	100%
No Reg.	95.1%	99.2%
$\lambda = .01$	95.8%	99.2%
$\lambda = .1$	96.2%	100%

Table 1: Comparison of the maximum achieved test and validation set accuracies. Each model has been trained once for 1200-1500 epochs using the seed 2020. All the best-performing instances on the test set were encountered in the first 600 epochs.

Evolution of τ during training

As seen in Fig. 1, the learning trajectory of membrane time constant parameters is highly similar for all the new models. However, their trajectories significantly differ from those of the baseline model. This could be explained by the inherent limitation of visualizing the evolution of parameters in the new models using their mean value. Only a small subset of parameters may be needed to significantly decrease the total loss. However, this is contradicted by the final distributions and extrema of parameters. For example, in the fifth layer, none of the new model parameters are close to the τ of the baseline model, indicating a fundamental difference in how the new models find a good set of parameters.

A new hypothesis may explain this: *"Having a single τ per layer makes learning faster."* This is supported by the evolution of the baseline model's sixth layer τ during the 200 initial epochs. The baseline model overshoots and then corrects, while other membrane time constants remain stable. This points to the possibility that τ is optimized too quickly before the input weights can adapt to it. On the other hand, the evolution of the new models is slower and more stable. This is likely because when a neuron's τ updates due to a spike gradient, the membrane time constants of other neurons are updated in a delayed manner through the variance term in the loss function in subsequent time steps.

The final accuracy of all models is very similar despite a widely different evolution of τ between the baseline and the new models. It is possible that the accuracy is not highly sensitive to different values of τ and that the input weights can successfully adapt to a wide range of τ values. The opposite may also be true and can be reformulated into another hypothesis: Given that the input is temporally encoded, *there is a general relationship between the learning capacity of a spiking layer and the number of learnable membrane time constant and input weight parameters*. This implies that a spiking layer with a single τ could be replaced by a smaller spiking layer with a τ per neuron without negatively impacting learning capability.

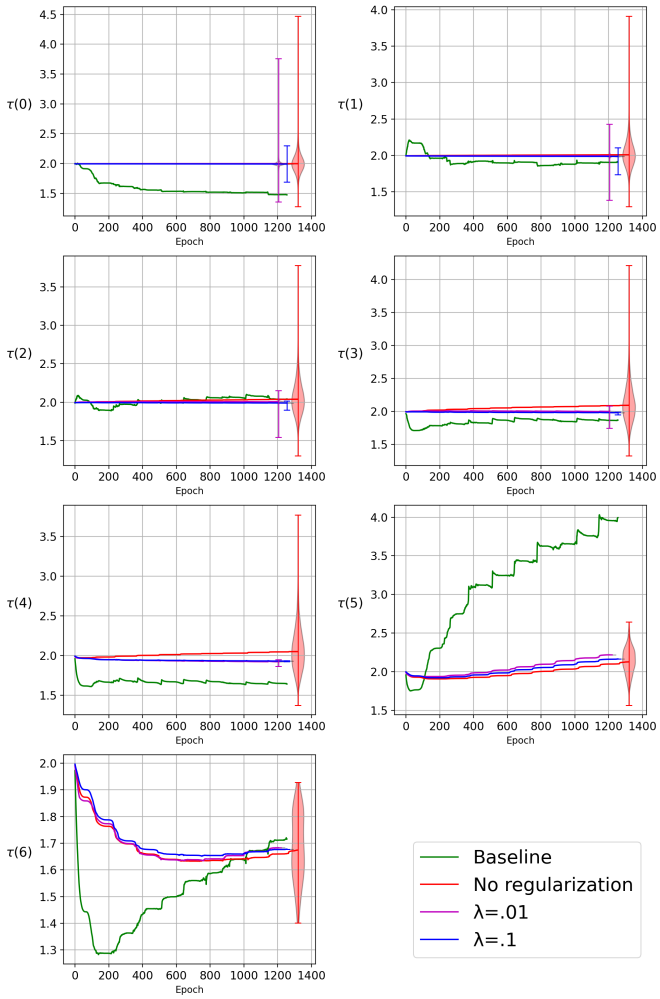

Figure 1: **Evolution of the membrane time constant τ in individual layers of PLIF neurons.** While the baseline model from [14] uses a single τ per layer, the other models use τ per each neuron. Multiple parameters per layer are visualized using mean values. Moreover, the final distributions of parameters and their extrema are visualized using violin plots. The y axis denotes values of the τ parameters in layer l as $\tau(l)$. The new models exhibit a vastly different learning trajectory compared to the baseline model. However, despite these differences, all models achieve similar final accuracy. This suggests that the input weights can adapt similarly well to a wide range of τ values.

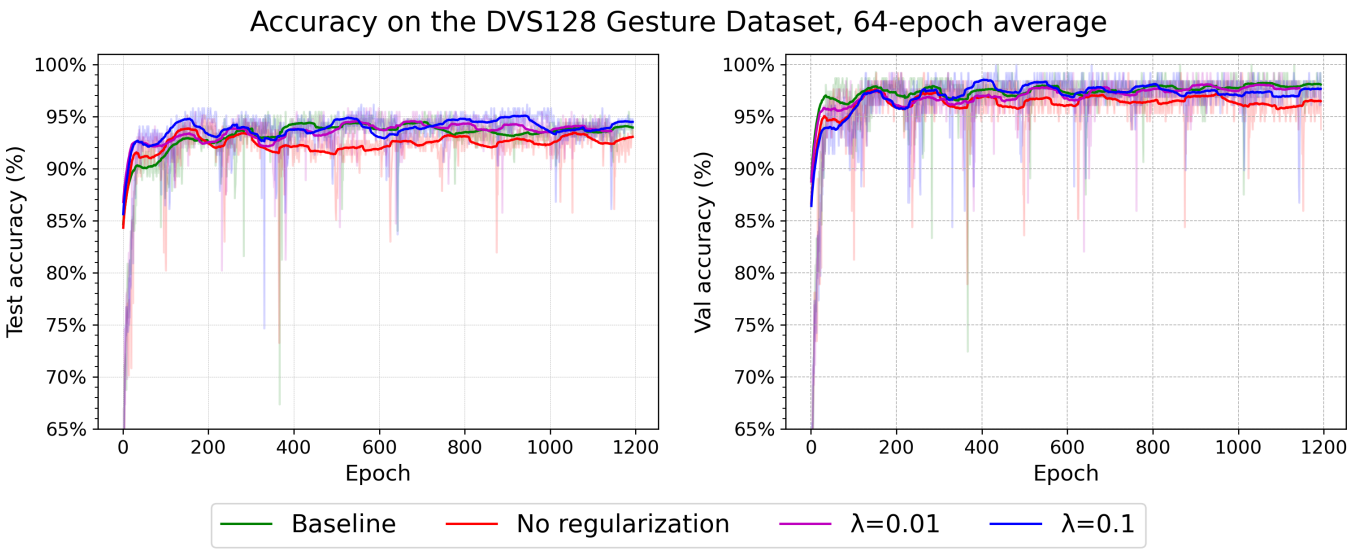

Figure 2: **Test and validation accuracy during training on the DVS128 Gesture dataset visualized as a 64-epoch moving average**. The baseline model from [14] uses a single learnable τ per spiking layer. The "No regularization" model replaces the single τ per layer with a learnable τ per neuron. The "$\lambda = 0.01$" and "$\lambda = 0.1$" models also add a regularization term to the loss function, calculated as $\lambda * \text{Var}(\tau_l)$ per spiking layer l. Despite their differences, all models achieve similar final accuracies, suggesting that having a membrane time constant per neuron instead of per layer does not significantly affect final accuracy.

"No regularization" model

The "No regularization" model performed better than expected with an accuracy of 95.1%. Nevertheless, its performance was marginally worse compared to the other models.

"$\lambda = 0.1$" model

The "$\lambda = 0.1$" model performed as expected, achieving an accuracy of 96.2%, comparable to the expected accuracy of the baseline model at 96.5%.

"$\lambda = 0.01$" model

The "$\lambda = 0.01$" model performed worse than expected with an accuracy of 95.8%. It was hypothesized that letting neurons optimize τ more freely would lead to increased accuracy relative to the "$\lambda = 0.1$" model. This hypothesis turned out false.

5 Responsible Research

Replicability is ensured by making all code available on GitHub. The repository includes a README file with a detailed description of the project, as well as instructions on how to install dependencies and run the experiments. All experiments were conducted using the seed 2020, ensuring replicability.

Large Language Models were extensively utilized for various purposes, which are listed, along with prompt examples, in Appendix A.

6 Discussion

Due to the resource constraints of this work, the results were obtained by training each of the four models only once on the DVS128 Gesture dataset. This represents a significant limitation to how confidently the results can be interpreted. Experiments across more datasets, ideally with multiple initializations per model, are required to arrive at a definite conclusion. However, two new hypotheses were identified.

There is an inverse relation between the number of τ parameters and their learning rate. This hypothesis is strongly supported by Fig. 1 and discussed in detail in Section 4.3. In all layers, the baseline model's τ changes significantly faster than the mean of the other models' parameters. However, to verify this hypothesis, it is necessary to define the learning rate in the context of both all τ parameters and input weights, since they learn together.

There is a general relationship between the learning capacity of a spiking layer and the number of learnable τ and input weight parameters. This hypothesis is supported by the fact that despite different values of the τ parameters, all models performed comparably well. This points to a relation between input weights and τ parameters, in which they can be optimized interchangeably. Therefore, there may be some equivalence, in terms of performance, between small models with a large number of τ parameters and large models with a small number of τ parameters.

7 Future Work

Layers as heterogeneous networks Both in this and the previous work, it was assumed that a single layer of neurons forms a population whose τ parameters should be regularized to stay together. However, there seems to be no reason why a single layer should correspond to only one population. It is possible that performance improvements can be achieved

by treating spiking layers as heterogeneous networks [17, Ch. 12.2.2]. This would require an improved regularization term that incentivizes clustering but does not limit the number of clusters to one.

Initialization of τ It still remains a mystery why the initializations of $\tau_0 = 2$ and $\tau_0 = 16$ in the previous work consistently produced significantly different results [14, Fig. 6]. Further investigation into the effect of τ initialization is needed.

Adaptive τ learning Despite differences in the evolution of the τ parameters in Fig. 1, all models achieved similar performance. The baseline model's τ in the sixth layer overshoots and corrects during the first 200 epochs, which suggests a complex training dynamic between input weights and τ parameters. This interplay requires further analysis and could enable more efficient learning by dynamically adapting the respective learning rates.

Local regularization It is possible that different layers would benefit from different regularization terms.

8 Conclusion

The purpose of this work was to investigate the effect of having a learnable membrane time constant τ per neuron on the accuracy of a spiking neural network (SNN). This was achieved by introducing a modified version of the PLIF neuron layer, featuring a learnable membrane time constant τ per neuron as opposed to per layer. This created the need for new initialization and regularization methods for τ, which were also introduced.

Experiments on the DVS128 Gesture dataset produced inconclusive results, suggesting with low confidence that **having a membrane time constant per neuron instead of per layer does not have a significant effect on final accuracy**. However, experiments on more datasets are needed to validate this conclusion, ideally with multiple initializations per model.

The evolution of τ during training exhibited interesting behavior. This was used to further analyze the role of the membrane time constant in the training of spiking neural networks, setting the groundwork for future research on this topic. This includes the formulation of two hypotheses supported by experimental results and four concrete areas for future research.

Appendix

A Prompts for LLMs

This work has extensively utilized Large Language Models (LLMs) for the following purposes:

- Providing feedback and improvements on the text's structure, readability, and grammar.
- Offering high-level interactive introductions to areas previously unfamiliar to the author, including relevant terminology and context within existing scientific disciplines.
- Ensuring consistent notation and terminology throughout the text.
- Converting natural language or pseudo-code into LaTeX equations.
- Evaluating the completeness and clarity of the presented information from a reader's perspective.
- Generating a section overview paragraph, which is then used as a skeleton for a manually written overview paragraph.
- Generation of python code for generation of figures.

Example prompts:

Q: "I need to convert the following into a latex equation: "X(t) = sum over all w_i * x_i(t), where x_i is discrete either 0 or 1 and w_i is a real number""

A: "Here's a LaTeX equation that represents your description: $X(t) = \sum_i w_i \cdot x_i(t)$, where $x_i(t) \in \{0, 1\}$, $w_i \in \mathbb{R}$"

Q: "I have the following code: ``` py [current code for figure generation]``` I want to also color other "parts" of the violin plot than just the "bodies". Also, I'd like to remove the left part of the violin plot."

A: "To customize the violin plots further according to your requirements, including coloring additional parts and removing the left part of the violin plot (which typically represents the mirrored density), you can modify the plotting parameters of the violin plots. Here's how you can do it: ``` py [modified code]``` "

Q: "Reformulate this: However, it is necessary to first adopt an incremental approach based on existing work and establish a new baseline to which new regularization methods can be compared."

A: "However, it is essential to first take an incremental approach based on existing work and establish a new baseline for comparing new regularization methods."

B Network structure diagram

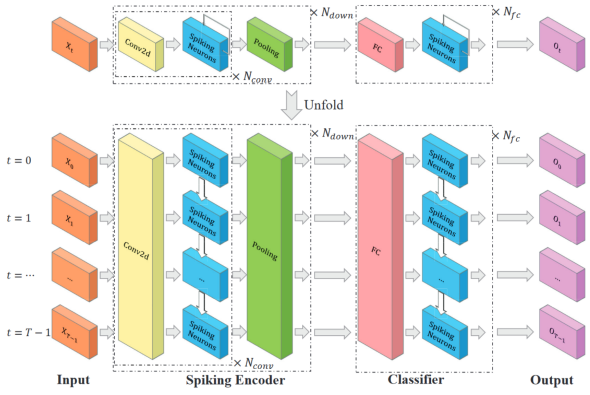

Figure 3: Network structure taken from the original work. "The general formulation of our networks and its unfolded formulation. $\times N_{conv}$ indicates there are N_{conv} {Conv2d-Spiking Neurons} connected sequentially. $\times N_{down}$ and $\times N_{fc}$ have the same meaning. Note that the network's parameters are shared at all time-steps." Section 3.3 in [14]. For the DVS128 Gesture dataset, the specific structure is $N_{conv} = 1$, $N_{down} = 5$, and $N_{fc} = 2$.

References

[1] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[2] C. Pehle, S. Billaudelle, B. Cramer, J. Kaiser, K. Schreiber, Y. Stradmann, J. Weis, A. Leibfried, E. Müller, and J. Schemmel, "The brainscales-2 accelerated neuromorphic system with hybrid plasticity," 2022.

[3] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the spinnaker system architecture," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, 2013.

[4] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses," *Frontiers in Neuroscience*, vol. 9, 2015.

[5] A. Henkes, J. K. Eshraghian, and H. Wessels, "Spiking neural networks for nonlinear regression," *CoRR*, vol. abs/2210.03515, 2022.

[6] Y. Sandamirskaya, M. Kaboli, J. Conradt, and T. Celikel, "Neuromorphic computing hardware and neural architectures for robotics," *Sci. Robotics*, vol. 7, no. 67, 2022.

[7] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, M. Debole, S. Esser, T. Delbruck, M. Flickner, and D. Modha, "A low power, fully event-based gesture recognition system," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7388–7397, 2017.

[8] F. C. Bauer, D. R. Muir, and G. Indiveri, "Real-time ultra-low power ECG anomaly detection using an event-driven neuromorphic processor," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 6, pp. 1575–1582, 2019.

[9] Z. Yan, J. Zhou, and W. Wong, "Energy efficient ECG classification with spiking neural network," *Biomed. Signal Process. Control.*, vol. 63, p. 102170, 2021.

[10] S. Afshar, A. P. Nicholson, A. van Schaik, and G. Cohen, "Event-based object detection and tracking for space situational awareness," *CoRR*, vol. abs/1911.08730, 2019.

[11] J. Timcheck, S. B. Shrestha, D. B. D. Rubin, A. Kupryjanow, G. Orchard, L. Pindor, T. M. Shea, and M. Davies, "The intel neuromorphic DNS challenge," *Neuromorph. Comput. Eng.*, vol. 3, no. 3, p. 34005, 2023.

[12] J. K. Eshraghian, M. Ward, E. O. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu, "Training spiking neural networks using lessons from deep learning," *Proceedings of the IEEE*, vol. 111, no. 9, pp. 1016–1054, 2023.

[13] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Long short-term memory and learning-to-learn in networks of spiking neurons," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada* (S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), pp. 795–805, 2018.

[14] W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang, and Y. Tian, "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," in *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pp. 2641–2651, IEEE, 2021.

[15] F. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: continual prediction with lstm," in *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, vol. 2, pp. 850–855 vol.2, 1999.

[16] M. Beck, K. Pöppel, M. Spanring, A. Auer, O. Prudnikova, M. Kopp, G. Klambauer, J. Brandstetter, and S. Hochreiter, "xlstm: Extended long short-term memory," 2024.

[17] W. Gerstner, W. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. 08 2014.

[18] S. S. Chowdhury, C. Lee, and K. Roy, "Towards understanding the effect of leak in spiking neural networks," *Neurocomputing*, vol. 464, pp. 83–94, 2021.

[19] W. Fang, Y. Chen, J. Ding, Z. Yu, T. Masquelier, D. Chen, L. Huang, H. Zhou, G. Li, and Y. Tian, "Spikingjelly: An open-source machine learning infrastructure platform for spike-based intelligence," 2023.

[20] W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang, and Y. Tian, "Supplementary materials for: Incorporating learnable membrane time constant to enhance learning of spiking neural networks,"

[21] C. Pehle and J. E. Pedersen, "Norse - A deep learning library for spiking neural networks," Jan. 2021. Documentation: https://norse.ai/docs/.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015.

[23] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Y. W. Teh and M. Titterington, eds.), vol. 9 of *Proceedings of Machine Learning Research*, (Chia Laguna Resort, Sardinia, Italy), pp. 249–256, PMLR, 13–15 May 2010.

[24] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Training pruned neural networks," *CoRR*, vol. abs/1803.03635, 2018.

[25] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," 2014.

[26] H. Li, "CIFAR10-DVS," 5 2017.