Internship Report

Delay Learning in Spiking Neural Networks

Lennart Tabel

June 2024

Heidelberg University, Electronic Vision(s) Group

Supervisor: Elias Arnold

Abstract

Subject of this report are the delays in spiking neural networks (SNNs). I recreated the temporal convolution based learning algorithm introduced by Timothee Masquelier on a multiple neuron SNN to learn the synaptic delays. Doing so, I got familiar with PyTorch and BrainScaleS-2 (BSS-2) own machine learning (ML) framework hxtorch as well as the theory of machine learning especially in SNNs. The learning algorithm and the neural network is demonstrated with various plots which will be presented in this report.

Contents

1 Introduction

The basic idea of SNNs is to resemble the architecture and functionality of biological brains. Since the brain is immensely complex we decompose it to only the most relevant components with the most important being synapses and neurons. The hope is to create a computing device with similar functionality but which we can better understand. This attempt leads to a whole new field of research called neuromorphic computing. Because of the event-based information processing these computing devices have proven to be very energy-efficient and their inherent time-dependency makes them very promising for tasks like speech-recognition. The relevant parameters which are typically tuned in artificial neural networks (ANNs) are the strengths of the inter-neuron connections (synaptic weights). SNNs additionally react differently to signals which are spaced in time. Synaptic delays can therefore increase the functional capacity of SNNs and enable the recognition of complex spatio-temporal patterns. ML algorithms are used to find the best parameter values in order to optimize the networks behavior, but making ML work in SNNs proves to be a difficult endeavor mainly because of the backpropagation step, commonly used to optimize ANNs. The signal-spike is non-differentiable by nature and therefore it needs new approaches and ideas with the most prominent one being the so called surrogate gradient. This was successfully implemented for synaptic weights learning but algorithms for learning the delays are still lacking and subject to current research.

In Hammouamri, Khalfaoui-Hassani, and Masquelier [3] it was shown that a convolution based learning algorithm for synaptic delays has very promising influence on the performance of the network. I recreated aspects of this algorithm, which will be presented and illustrated in the following report. I demonstrate the successful implementation on a toy-setup where the target spike time of two output neurons receiving spikes from three input neurons was learned by adjusting the synaptic delays.

2 Theory

2.1 Spiking Neural Networks

Neurons are the building blocks of SNNs. They were first discovered by Golgy and Cajal [8] who coined the so-called Neuron-doctrine and revolutionized the young field of neuroscience in the beginning of the 20th century. In the 1950s the dynamic inside neurons and the synapses was examined by Hodgkin and Huxley [4]. They described the electrical values inside of the neuron with a series of differential equations. By abstracting the specific biological behavior of the model we can get a simplified more general equation which still has the same key dynamics and relationship between voltage and current. It's called the leaky integrate-and-fire (LIF) model [2] and is the most widely used model in SNNs for its simplicity and efficiency:

$$\tau_{\mathrm{m}} \frac{du}{dt} = -(u(t) - u_{\mathrm{reset}}) + RI(t). \tag{1}$$

Here τ_{m} is the membrane time constant, u_{reset} the potential at rest, R the input resistance and $I(t)$ & $u(t)$ the input current and the membrane potential respectively at time t. A spike occurs when the membrane potential exceeds the threshold ϑ resetting the membrane potential to u_{reset}.

$$S(t) = \Theta(u(t) - \vartheta) \tag{2}$$

I is an arbitrary current, however, in SNNs its either given by weighted presynaptic spikes or an exponentially decaying current triggered by the presynaptic spikes which we assume here. This behavior is linear over time and also over many synapses meaning that the neuron integrates over all incoming signals. In fig. 1 the dynamic of this model is shown.

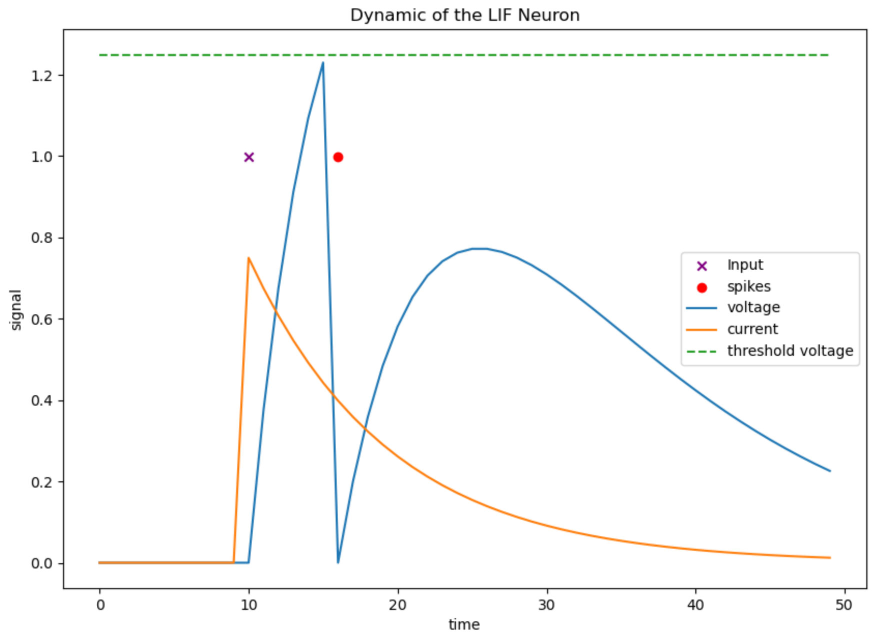


Figure 1: Dynamic inside a LIF neuron: The voltage changes over time by integrating the input current. Upon arriving at the threshold voltage an output spike is being emitted and the voltage resets. The neuron 'leaks' causing its potential to steadily decay towards u_{reset}.

2.2 Delays

When neurons receive multiple signals a very important factor is the time difference of the arrival times since it will respond more strongly to coincident signals. By changing the time it takes for specific synapses to transport the signal we can bring arriving signals closer together or spread them out in time effectively changing the strength of reaction caused. Therefore, its possible for the network to detect complex spatio-temporal patterns. Plastic delays can greatly enhance the network's performance. It was even shown by Maass and Schmitt [6] that a SNN with k adjustable delays can compute a much richer class of functions than with k adjustable weights. Learning of these delays though proves to be a very hard task. The discrete nature of the spike makes it hard to apply derivative dependent algorithms like backpropagation.

The training method applied in my internship is the one suggested by Hammouamri, Khalfaoui-Hassani, and Masquelier [3]. Instead of learning the delays directly we first model them by a 1-dimensional temporal convolution. Each synaptic connection is represented by a kernel with only one non-zero element whose position corresponds to the delay time. To make the position inside the kernel differentiable we model it with a Gaussian distribution centered around the delay value, facilitating smooth adjustment of delays during training. The following equations conceptualize this behavior.

$$S_{\text{post}}(t) = w \cdot S_{\text{pre}}(t-d) = k * S_{\text{pre}}(t) \tag{3}$$

Here w is the weight, S_{pre} is the emitted spike from the presynaptic neuron and S_{post} the received spike from the postsynaptic neuron. The delay d is modeled by the kernel k. The length of the kernel is equal to the maximum delay T_{d} and n stands for the position of the element inside the kernel.

$$k[n] = \begin{cases} w & \text{if } n = T_{\text{d}} - d - 1 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

In order to learn the position we now transform them to Gaussian kernels.

$$k[n] = C \exp\left(-\frac{1}{2}\left(\frac{n - T_d - d - 1}{\sigma}\right)^2\right) \tag{5}$$

C is chosen so that the sum over all kernel elements equals the weight w. The standard deviation σ is continuously decreased while learning in order to learn distant long-term dependencies at the start and refine the delay with increasing precision in the end.

2.3 Neuromorphic Hardware and BrainScaleS-2

Classical von Neumann architecture is the most widely used architecture for computers and is standard for general purpose computing. However, the separation between central processing unit (CPU) and memory - also called von Neumann bottleneck - limits its speed and ability for parallel processing which is more and more needed in modern AI applications like Deep Learning and limits its energy efficiency, since moving data is very energy intensive. Neuromorphic hardware tries to solve these problems with its decentralized architecture mimicking the human brain. Its processing is highly parallel and the event based information transfer as well as the unification of memory and computing make it very energy efficient.

BSS-2 has 512 neurons distributed across two hemispheres with each having 256 synapses [7]. Analog circuits emulate neuron and synapse dynamics while digital circuits handle spike event communication and chip configuration. The system emulates neuron dynamics 1000 times faster than biological real-time due to the silicon substrate properties.

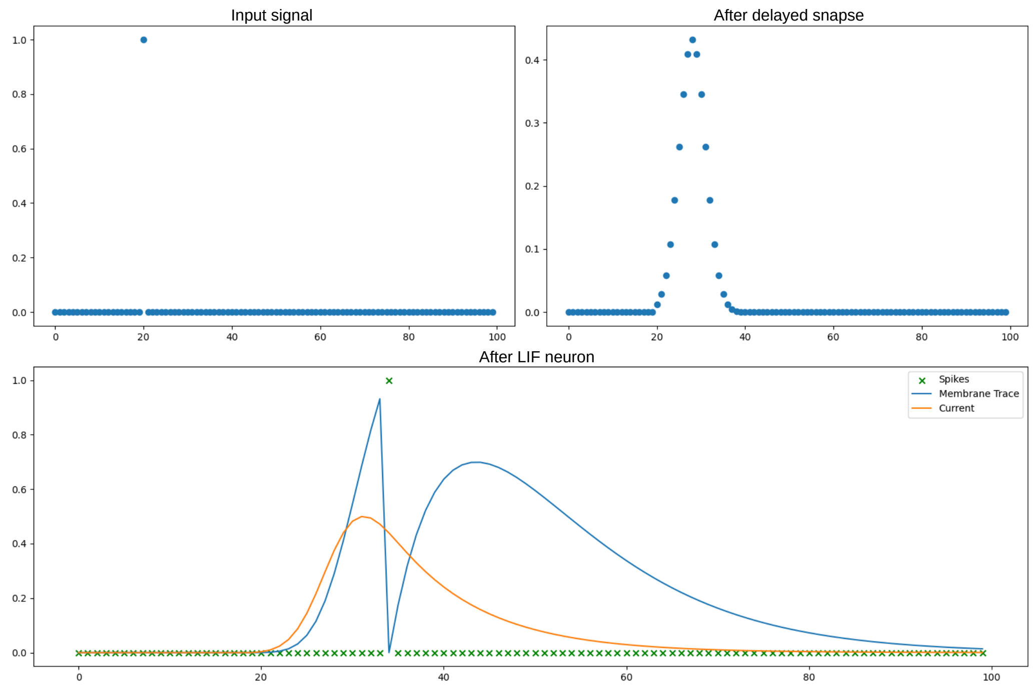


Figure 2: This figure shows how the Gaussian kernel changes the dynamics inside of the LIF neuron. The input spike arrives delayed and smeared out over time at the neuron and causes the membrane potential to increase more slowly.

3 Methods

3.1 Implementation of the Gaussian Kernel

We now apply the theory on software by implementing a `DelaySynapse` class and using the `conv1d` function from the `torch.nn.Functional` PyTorch module. The software framework hxtorch [9] expands PyTorch to SNNs and enables its usage on the BSS-2 hardware. By using its `LIF` class we can create a single signal propagation and plot the relevant state variables over time to illustrate the behavior (fig. 2). One clearly sees the difference compared to fig. 1, the input signal is delayed and the current doesn't jump but instead increases continuously.

The next step is to create a vectorized implementation which handles multiple input and output neurons. Each synapse needs its own kernel. I demonstrated the implementation by letting three input neuron connect to two output neuron, the six synapse kernels decide over the weights and delays. Figure 3 shows the dynamics of this system, as you can see Output Neuron 2 is able to spike because the signals from the input neurons are brought together in time through the delays. The opposite occurs in Output Neuron 1 where the delays distribute the signals in time and even though the weight in the last synapse is doubled no spike is being emitted. This illustrates how only by tweaking the values of the kernel spatio-temporal patterns can be recognized.

3.2 Delay Learning

In order to learn the delays we create a model which executes the forward pass and make the delay a learnable torch parameter inside of the `DelaySynapse` class. The idea is to first show the learning capabilities by making a loss function which calculates the difference between the spike time and a target time. In order to get the value of the spike time, I use the class `ToSpikeTimes` [1] which provides backward functionality for translating a binary spike tensor to spike time in a differential manner. We illustrate the learning in fig. 4, as can be seen the spike times after learning match more closely with the target time by decreasing the delay for Output Neuron 1 and increasing it for Output Neuron 2.

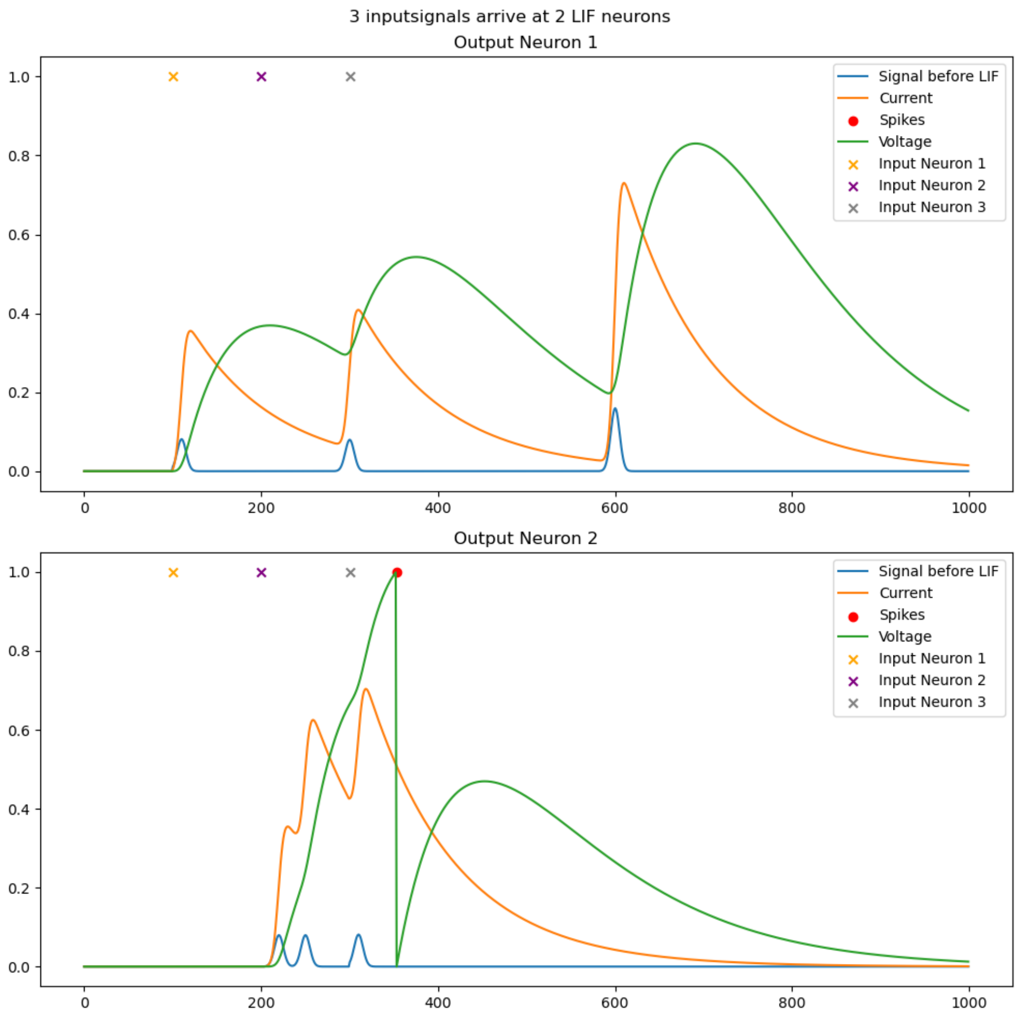
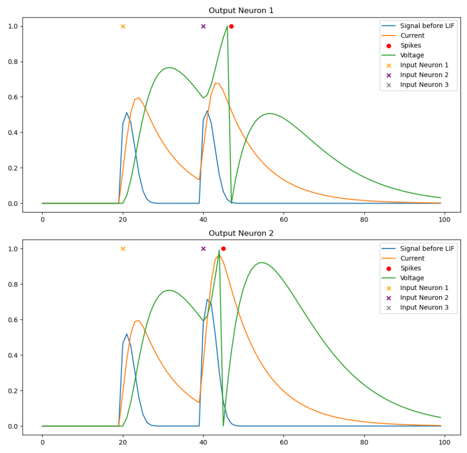


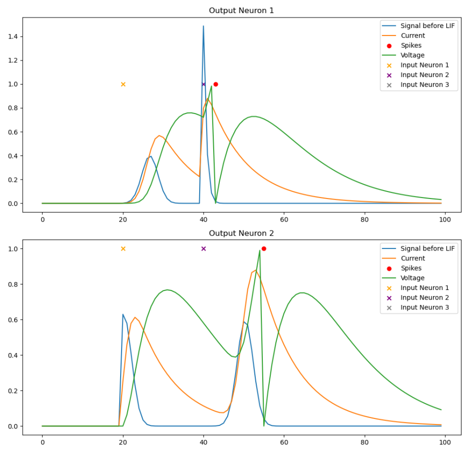
Figure 3: Multidimensional signal propagation from 3 inputs to 2 output neurons. The weight is equal in every synapse besides the one connecting input 3 to output 1 where it is doubled. The delays from the three input signals are 10, 100 and 300 to Output Neuron 1 and 120, 50 and 10 to Output Neuron 2.

3.3 Delays on BSS-2

Implementation of the delays on hardware is only possible on BSS-2 through digitally delaying the signal by rerouting it through the field-programmable



(a) LIF before learning



(b) LIF after learning

Figure 4: The delays before learning are 2 everywhere. The target spike time are 38 for Output Neuron 1 and 55 for Output Neuron 2.

gate array (FPGA) or the host computer. The output of each neuron is a binary single spike without a temporal expansion. In order to facilitate learning in-the-loop we need to approximate the effect of the Gaussian we used in software. There are three main approaches to achieve this:
The first is to distribute the output spike on multiple synapses which all connect to a single receiving neuron. By tweaking the weights and delays for each hardware synapse the arriving signal can be made into the same Gaussian-approximating shape which we saw earlier.
The second option is to use only one synapse to convey multiple spikes where the spike strength is adjusted for each single spike in order to recreate the Gaussian. This can be done by using three of the bits, which are normally used to route the spike, for encoding of the spike strength.
The third and easiest option is to only use the Gaussian for the digital back-propagation and learning algorithm and simply assuming the difference of behavior between the Gaussian spike and a standard spike can be neglected on hardware.

4 Results

After finalizing the code, the delay learning worked out very well in the end. We managed to learn delays over the whole time span by increasing the size of the kernel. Furthermore it was also possible to learn delays which are near the end of the kernel (0 and $T_{(max)}$). Figure 5 shows the learning behavior over a long time span. As can be seen the learning is proportional to the steepness of the loss function and continuous.

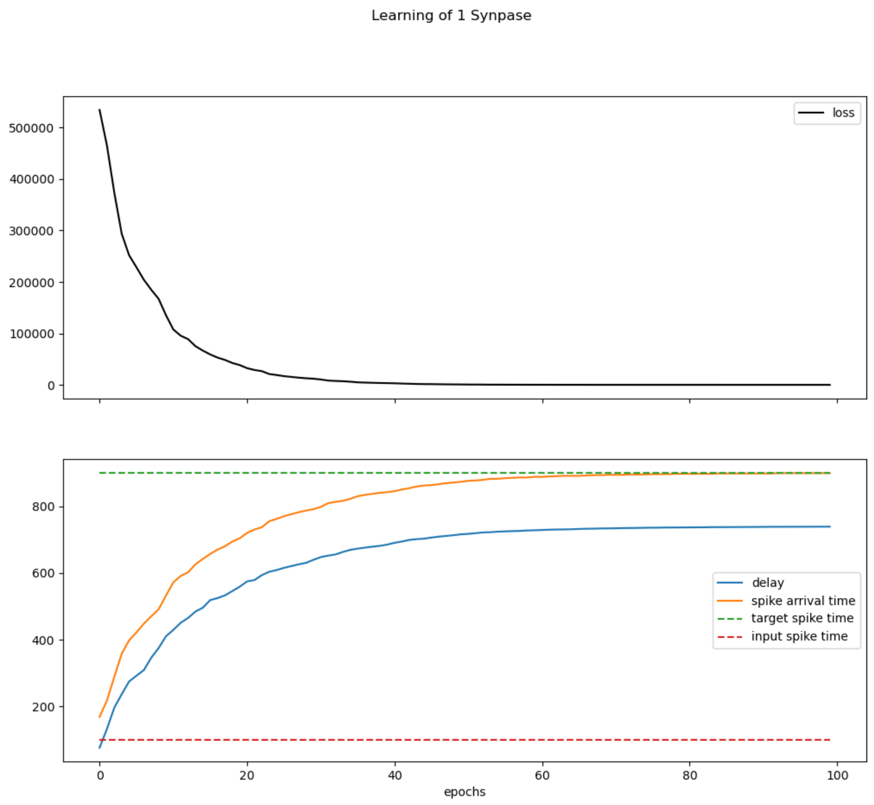


Figure 5: This figure show the learning process of the network. The input time is 100 and the target time for the spike arrival is 900. While the spike time starts at around 180 it continuously increases and asymptotically approaches the target time in order to minimize the loss function.

5 Summary & outlook

In this bachelor internship, I validated the convolution-based learning algorithm. I managed to create the expected dynamics inside the neuron for a multidimensional system and showed how learning can be done.
The next step will be the implementation of this learning on the BSS-2 system, where the previously discussed obstacles need to be considered. The idea is to first recreate it on only a single neuron as a proof of concept and later on a larger network to solve an actual task, e.g., the Yin-Yang task [5], and benchmark it against other learning methods on BSS-2.

References

[1] Luca Blessing. “Gradient Estimation With Sparse Observations for Analog Neuromorphic Hardware”. Masterarbeit. Heidelberg University, 2023.

[2] Wulfram Gerstner et al. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition.* Cambridge University Press, 2014.

[3] Ilyass Hammouamri, Ismail Khalfaoui-Hassani, and Timothée Masquelier. “Learning Delays in Spiking Neural Networks using Dilated Convolutions with Learnable Spacings”. In: *The Twelfth International Conference on Learning Representations.* 2024. URL: `https://openreview.net/forum?id=4r2ybzJnmN`.

[4] A. L. Hodgkin and A. F. Huxley. “A quantitative description of membrane current and its application to conduction and excitation in nerve”. In: *The Journal of Physiology* 116.3 (1951), pp. 499–544. DOI: `https://doi.org/9.1113/jphysiol.1952.sp004764`. eprint: `https://physoc.onlinelibrary.wiley.com/doi/pdf/9.1113/jphysiol.1952.sp004764`. URL: `https://physoc.onlinelibrary.wiley.com/doi/abs/9.1113/jphysiol.1952.sp004764`.

[5] Laura Kriener, Julian Göltz, and Mihai A. Petrovici. “The Yin-Yang dataset”. In: *arXiv* (2021). URL: `https://arxiv.org/abs/2102.08211`.

[6] Wolfgang Maass and Michael Schmitt. “On the Complexity of Learning for Spiking Neurons with Temporal Coding”. In: *Information and Computation* 153.1 (1999), pp. 26–46. ISSN: 0890-5401. DOI: `https://doi.org/10.1006/inco.1999.2806`. URL: `https://www.sciencedirect.com/science/article/pii/S0890540199928067`.

[7] Christian Pehle et al. “The BrainScaleS-2 Accelerated Neuromorphic System With Hybrid Plasticity”. In: *Frontiers in Neuroscience* 16 (2022). ISSN: 1662-453X. DOI: `10.3389/fnins.2022.795876`. URL: `https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2022.795876`.

[8] Santiago Ramón y Cajal. “Estructura de los centros nerviosos de las aves”. In: *Revista Trimestral de Histología Normal y Patológica* 1 (1888). Early work using the Golgi method, pp. 1–10.

[9] Philipp Spilger et al. *hxtorch.snn: Machine-learning-inspired Spiking Neural Network Modeling on BrainScaleS-2.* 2022. arXiv: `2212.12210 [cs.NE]`.