N-MNIST Handwritten digit classification using a simulated Spiking Neural Network

January 10, 2025

Abstract
This project demonstrates a simple Spiking Neural Network that performs a handwritten digit classification. For this the N-MNIST data set is used. A feed-forward network with Leaky-Integrate-and-Fire (LIF) neuron model is used. The network is trained using the mean square error count loss. L2 regularization was implemented to prevent overfitting. A specific arctan function was applied to minimize the effects of the *dead neuron problem*. A Spiking Neural Network model with 800 hidden neurons, trained in 15 epochs was found to achieve the best test accuracy of 97.054%

Contents

1 Introduction **3**

2 Theory **3**
2.1 Spiking Neuron Model 3
2.2 Dead Neuron Problem 5
2.3 Surrogate Gradient 5

3 Methods **6**
3.1 Data 6
3.2 Transforming input events 7
3.3 Architecture 8
3.4 Objective function: MSE Count Loss 8
3.5 Technical implementation of spiking neurons 9
3.6 Training 9
3.7 Hyper-parameter search 9

4 Results **11**

5 Discussion **11**

6 Conclusion **12**

1 Introduction

The efficiency of human brain, which only requires $\approx 12-20W$ of power to accomplish complex tasks, has been an inspiration for Machine Learning [1]. The branch of Neural Networks is a prime example of that, as it is based on the structure of neurons and their connections [2]. However, recreating the utter process of information passage through the brain is not an easy task, which lead to the creation of a variety of Neural Networks, that differ in their complexity.

Spiking Neural Networks (SNN) have lately received a lot of interest due to their biological plausibility and their possible implementation on neuromorphic hardware such as the Intel Loihi Research Chip [3]. Extreme energy efficiency, high performance and low latency may be expected when implemented on this neuromorphic hardware [4] [5] [6]. For this project it is chosen to train an SNN to classify handwritten digits from the N-MNIST dataset.

As opposed to a regular Artificial Neural Network (ANN), an SNN incorporates a concept of time. Whereas an ANN performs a propagation cycle on a static input activation to retrieve static output activations, an SNN operates on a time-dependent input spike train that influences intermediate membrane potentials and neuron activations to produce an output spike train. Intermediate SNN operations are built on a construction of individual neurons that fire a spike to the connected neurons if this neurons' membrane potential reaches a specific threshold value due to incoming spikes. In order to respectively produce and interpret the input and output spike trains, adequate sample encoding and label decoding methods are required.

The objective of the project was not to reproduce a biologically plausible neural network, nor generating an efficient SNN designed to execute on a neuromorphic chips. Instead, this work should be interpreted as a demonstration of a computer-simulated SNN and its capability of performing the N-MNIST handwritten digits classification task.

2 Theory

In order to classify the N-MNIST dataset using an SNN, one requires some insight in the underlying mathematics that govern the dynamics of spiking neurons. How neurons spike, which problems may arise during training and how to mitigate these problems is explored in the following section.

2.1 Spiking Neuron Model

Before analysing Spiking Neural Networks, one should acquire a basic understanding of the biological process that inspired this branch of Machine Learning, namely the information passage through neurons. Due to the structure, the human brain is capable of efficiently processing large amount of data coming from the environmental stimuli. As discussed by Jason K. Eshraghian et al. in [1] there are three significant reasons behind that.

Firstly, human attention is more event-based rather than state-based. In other words, a change in the environment is considered more relevant in comparison to a continuous stimulus. The reduction of information, that has to be processed at a given time, leads to the second reason behind brain efficiency. Due to the limited data, most of the neurons do not need to stay active, which decreases the amount of energy required by the brain. Finally, when the neurons are to forward a certain stimuli, they do so by emitting short electrical signals, which are referred to as spikes. An important factor is the timing at which the spike is generated.

By incorporation of these notions, a Neural Network can be modified to be more neuromorphic. This is done by inclusion of a spiking neuron model, which represents the biological process behind the information passage in the brain. Such a system is referred to as a Spiking Neural Network.

There is a variation of spiking neuron models, that differ in their complexity and biological plausibility. In this project a Leaky Integrate and Fire Neuron Model (LIF) was utilized, which is one of the more prevalent choices [1]. This model is based on the research conducted by Louis Lapicque in 1907, where the mechanism of a spiking neuron was compared to a circuit with a resistor R and capacitor C [7]. Let us explain these dynamic through more mathematical terms.

A membrane potential $U(t)$ of a neuron is its electrical charge at time t. It consists of the sum of the incoming input stimuli $I_{in}(t)$, which are labeled as the input spikes. In the LIF model the membrane potential is '*leaking out*', or in other words, decaying with a rate β. Additionally, it is assumed that all neurons have the same membrane potential threshold, which is denoted by θ.

Let $\tau = RC$ denote the time constant for the resistor-capacitor circuit. Then the membrane potential dynamic can be describes as

$$\tau \frac{dU(t)}{dt} = -U(t) + I_{in}(t)R. \tag{1}$$

As described in [1], the Forward Euler method can be used to approximate the solution to equation 1. Then the membrane potential at time t is equal to

$$U(t) = \beta U(t-1) + (1-\beta)I_{in}(t). \tag{2}$$

A neuron only emits an output spike S , if the membrane potential reaches the threshold θ. For the LIF model the value of the output spike is set to 1. This can be expressed through the following function

$$S(t) = \begin{cases} 1, & \text{if } U(t) > \theta \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

After an output spike has been emitted, the value of the membrane potential has to be reset by the subtraction of the threshold value. The reset can be performed by the addition of the term $-S(t-1)\theta$ to the expression 2. Then the modified equation becomes

$$U(t) = \beta U(t-1) + (1-\beta)I_{in}(t) - S(t-1)\theta. \tag{4}$$

Finally, let us rephrase the second term of the sum $(1-\beta)I_{in}(t)$ as $WX(t)$, where $X(t)$ denotes a single input and W is a weight parameter, that includes $(1-\beta)$. This change is beneficial, as weight W is a learnable parameter. This leads to the equation,

$$U(t) = \beta U(t-1) + WX(t) - S(t-1)\theta, \tag{5}$$

which describes the membrane potential dynamic in a Leaky Integrate and Fire Neuron Model.

2.2 Dead Neuron Problem

According to Eshraghian et al. [1], one of the biggest challenges of training a Spiking Neural Network, comes from the non-differentiability of the output spikes.

Recall, the output spike, defined in equation 3, is a step function

$$S(t) = \begin{cases} 1, & \text{if } U(t) \geq \theta \\ 0, & \text{otherwise,} \end{cases}$$

which is not differentiable due to its discontinuity at the threshold θ. Only at point θ the derivative $\frac{\partial S}{\partial U}$ is equal to $+\infty$, while at any other point it is 0. This is depicted in Figure 1.

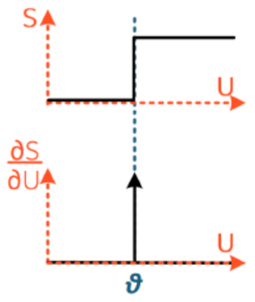


Figure 1: Non-differentiability of output spikes (Image from Eshraghian et al. [1])

The objective of training is to obtain a weight W of a neuron that minimizes a loss function $\mathcal{L}$. This can be analyzed through consideration of the gradient of loss $\mathcal{L}$ with respect to the neuron weight W [1].

By the use of the chain rule, the relation between the gradient of loss and the derivative of the spike function can be formulated as

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial S} \frac{\partial S}{\partial U} \frac{\partial U}{\partial I} \frac{\partial I}{\partial W}. \tag{6}$$

However, as the derivative of the spike function $\frac{\partial S}{\partial U}$ is 0 everywhere except at the threshold θ, the value of the loss gradient $\frac{\partial \mathcal{L}}{\partial W}$ is also diminished to 0, with an exception of one point. In practice this interferes with the error backpropagation process, as the weights of neurons cannot be re-evaluated. This is referred to as the *dead neuron problem* [1].

2.3 Surrogate Gradient

In order to minimize the effects of the *dead neuron problem* a surrogate gradient method is used. As defined by Eshraghian et al. in [1], the main idea behind this method is to use a differentiable approximation of the spike function $\tilde{S}$ during backpropagation. This modification results in a continuous derivative $\frac{\partial \tilde{S}}{\partial U}$, that makes the gradient of loss $\frac{\partial \mathcal{L}}{\partial W}$, as defined in equation 6, computable. It is important to note, that the original spike function S is still used for the forward-pass, as the approximation $\tilde{S}$ is only used for the backpropagation step.

An example of the approximation $\tilde{S}$ and its derivative $\frac{\partial \tilde{S}}{\partial U}$ is visible in Figure 2. The grey functions in the plots correspond to the original spike function S, as shown in Figure 1.

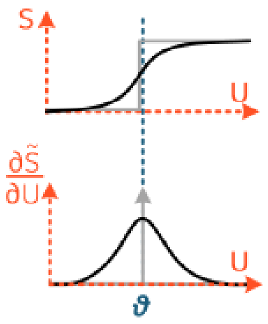


Figure 2: Surrogate gradient of output spikes (Image from Eshraghian et al. [1])

The surrogate gradient function used in this project was introduced by Neftci et al. in [8], where the approximation function is

$$\tilde{S} = \frac{1}{\pi} arctan(\pi U)$$

and its derivative is given by

$$\frac{\partial \tilde{S}}{\partial U} = \frac{1}{\pi} \frac{1}{(1 + (\pi U)^2)}.$$

According to Eshraghian et al. [1], this arctan function is the standard choice for the surrogate gradients, due to its efficiency. For that reason, it is used in this project.

Although the use of the surrogate gradient method vastly reduces the impact of the *dead neuron problem*, it does not fully prevent it. The approximation $\tilde{S}$ is only used when a spike occurs, which implies that only the weight of spiking neurons is re-evaluated. If a certain neuron does not activate for a prolonged period of time, it becomes a *dead neuron* and is not used for the duration of the training. A possible solution is enforcing the inactive neurons to spike occasionally, so that their weight can be re-evaluated [1].

This concludes the theoretical explanation behind Spiking Neural Networks. For a more detailed description one should refer to the paper written by Eshraghian et al. [1].

3 Methods

In order to utilize the theory of a single spiking neuron to construct an SNN that correctly classifies the N-MNIST dataset, there is multiple steps to be executed. In this section it is explained how the MNIST data-set is converted to spike-trains, what the SNN architectures looks like, and the training method with its respective loss-function is explained. The section is concluded with remarks about finding the best hyper-parameter values.

3.1 Data

The dataset used is the N-MNIST dataset [9] used for classifying hand-written digits. This dataset contains 60.000 training and 10.000 testing samples at a 34×34 pixels resolution, each linked to the appropriate class. As opposed to the original MNIST dataset [10], input samples are not a static image. Each input sample is a 100ms recording of a saccadically moved sample from the original MNIST dataset captured by an ATIS asynchronous time-based image sensor [11]. Whereas a regular camera collects complete frames at a constant interval of a scene, an asynchronous time-based camera only captures changes in the scene. An illustration of this concept can be found in Figure 3.

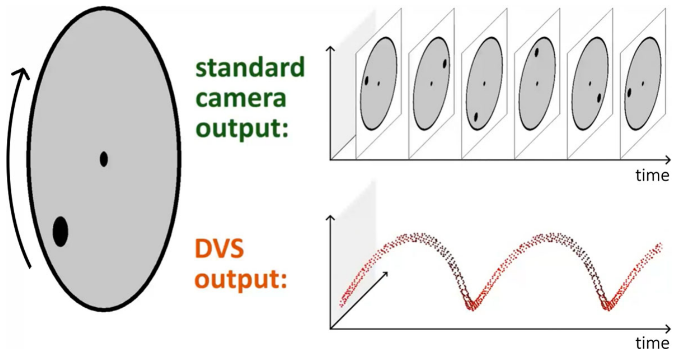


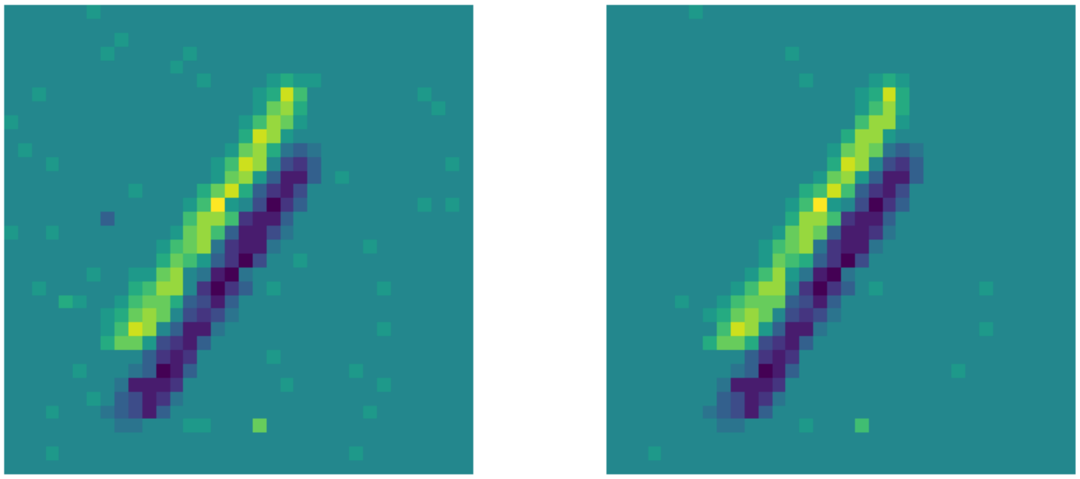
Figure 3: Comparing the output of a standard camera and an asynchronous time-based image camera when pointed at a spinning disk with a black circle. (Image from Rebecq et al.[12])

Applying this idea on a saccadically moved MNIST dataset, individual pixels can collect an on-event where the pixel goes from white to black, or an off-event, where the pixel goes from black to white. By interpreting the individual pixels as a pair of input neurons. 1 for both on- and off-events. The on- and off-events are respectively causing the on- or off-neuron to spike.

3.2 Transforming input events

The data N-MNIST consists of data describing spikes or ‘events’. All spikes within 1 microsecond are collected into observations. In total there are 100.000 of these observations. Hence, each sample has 100ms of footage. These observations are what make up the N-MNIST data set. For the input data to be usable for the spiking neural network (and at the same time reduce the number of time steps at the input), we collect the observations into frames by binning the observations. Hence, all observations in a specified time window, with a window length t_{window}, are binned resulting in a frame. This results in $\frac{100.000}{t_{\text{window}}}$ frames for each sample. Where each frame is non-overlapping and equally spaced in time. (Window length t_{window} is specified in subsection 3.7.)

As well as putting the events into frames, the events are also ’denoised’. This is done by removing events that are isolated for a long enough time. If there are no other events in a 1-pixel neighborhood around the isolated event for 10.000 micro seconds, it is removed.
Figure 4 depicts an example of ’denoising’ digit ’1’.


(a) A frame with a very large time windows, showing all events in one saccade movement. On-events are more black and off-events are more white/yellow. Shows the handwritten digit '1'.

(b) The same frame as (a) but denoised.

Figure 4: 'Denoising' of handwritten digit '1'

3.3 Architecture

A fairly simple architecture was used which is similar to a multilayer perceptron with 1 hidden layer. There are $34 \times 34 \times 2 = 2312$ input neurons, one for every pixel and polarity of the pixel. There are 10 output neurons, one for each classification target (written digit 1 to 10).

For the hidden layer we used n_{hidden} neurons. This number is varied in the hyper-parameter search (subsection 3.7).

Layers are densely connected and use the LIF model for the neurons. Neurons use '*arctan*' surrogate gradient (subsection 2.3) such that the backpropogation algorithm can work properly on the neurons. The membrane potential decay rate β is also varied in the hyper-parameter search.

3.4 Objective function: MSE Count Loss

Mean Square Error Spike Count Loss is used as the objective function for the Spiking Neural Network. It is defined by Eshraghian et al. in [1] as the Mean Square Error applied to the *spike count* and the *target spike count*, for both the correct and incorrect classes. To elaborate, a *correct spiking count* for each neuron is the amount of spikes that are accurately linked to a class, while the rest is accounted in the *incorrect spiking count*. The goal is to achieve a certain *target spike ratio* between the two. The optimal value for the *target ratio* is found through a hyper-parameter search discussed in subsection 3.7.

Enforcing a certain amount of the incorrect spiking behaviour of a neuron reduces the risk of it becoming a *dead neuron* [1]. This ensures the re-evaluation of the weight of the neuron. The occurrence and dangers behind the *dead neuron problem* are as explained in subsection 2.2.

3.5 Technical implementation of spiking neurons

The spiking neurons are implemented using the *torch* library combined with the *SNNtorch* library [13]. A standard ANN will be altered such that it can represent a spiking neural network architecture.

At first, we have to alter the forward pass of the training phase. Recall that one sample consists of 100ms of footage which is collected into many frames. One of the main differences between a normal ANN and a spiking ANN is that neurons are replaced by LIF neurons. This is implemented in the activation. When a new observation is shown to the network, all weights are set to θ. After this, the charge of the neuron is described by Equation 5. The neuron will pass on a spike following Equation 3. The number of spikes in the last layer will be counted and used for calculating the loss. Hence, the activation function is the one that ensures the neuron has a "memory".

This all is implemented in python 3.11.3 [14]. The spiking data is sourced from *tonic* [15]. To be able to bring down computation times, the high-performance computer cluster of the RUG, Hábrók, is used.

3.6 Training

Weight initialization for each layer is drawn from an uniform distribution

$$\mathcal{U}\left(-\sqrt{\frac{1}{n_{\text{prev}}}},\sqrt{\frac{1}{n_{\text{prev}}}}\right), \tag{7}$$

with n_{prev} the number of hidden units in the previous layer.
To prevent overfitting, regularization techniques are applied. We have used L2-regularization where the following term is added to the cost function

$$L2 = \lambda \sum_i w_i^2 \tag{8}$$

where we set $\lambda = 10^{-5}$.

Adam [16] is the optimizer we have used. This optimizer is currently the industry standard, and for more information about this optimizer we refer to: [16].

Lastly, the batch size has to be determined. The batch size determines the number of observations we pass through the network before adjusting the weights. A too-small batch size might result in overfitting and is generally slower. A too-large batch size can cause underfitting as a network is not able to learn from individual samples. The batch sizes we consider are 64, 128 and 256.

3.7 Hyper-parameter search

To determine the settings that let the model best fit the data, a hyper-parameter search is performed. What is important to note is that one epoch costs about 1.5 hours of computation time. Therefore, the hyper-parameter search is a costly operation. To limit the computational time, a relatively small search is conducted. The number of epochs is limited to decrease computation times while still being able to test all different settings. A set of different models has been parallelly trained on Hábrók for 3 epochs with a 5-fold cross-validation. We will perform a full grid search with the following settings: $\beta \in \{0.7, 0.9, 0.95\}$, the number of hidden units $n_{\text{hidden}} \in \{100, 200, 400, 800\}$, $t_{\text{window}} \in \{500, 1000, 2000\}$, and target ratio $\in$ {0.8/0.2, 0.95/0.05}. The batch size is fixed to 128.

Accuracy	β	n_{hidden}	t_{window}	Target ratio
0.9784	0.9	800	2000	0.8/0.2
0.9760	0.9	800	500	0.8/0.2
0.9742	0.7	800	2000	0.8/0.2
0.9741	0.9	800	1000	0.8/0.2
0.9734	0.7	800	1000	0.95/0.05
0.9725	0.9	800	2000	0.95/0.05
0.9725	0.7	800	500	0.95/0.05
0.9717	0.7	800	500	0.8/0.2
0.9716	0.7	800	1000	0.8/0.2

Table 1: Settings of the top nine mean accuracies of the first "small" parameter search on the validation sets. The accuracy is after 3 epochs with a 5-fold cross-validation. With $\beta \in \{0.7, 0.9, 0.95\}$, $n_{\text{hidden}} \in \{100, 200, 400, 800\}$ the number of hidden units, $t_{\text{window}} \in \{500, 1000, 2000\}$, and target ratio $\in$ {0.8/0.2, 0.95/0.05} Batch size is set to 128.

Table 1 presents the results of the small parameter search. The number of hidden units is the most important factor for high accuracy on the validation set, as for the top 10 highest accuracies, there are always 800 neurons. Secondly, it seems that a target ratio of 0.8/0.2 is a good setting. A β of 0.9 seems to have the best overall performance, and lastly, the number of events per frame seems to not have that much of an influence. Therefore, a value of 2000 seems to be a solid choice. Note that a lower value for t_{window} increases the number of frames per sample and with that also increases the training times.

After this first small hyper-parameter search it is a bit more clear which parameters are most important. A bigger hyper-parameter search is performed based on these initial results. This bigger parameter search is attempted for 15 epochs with 5-fold cross-validation. However, this took to much computational time and power. And therefore this search failed. To reduce training time, a switch is made to perform this hyper-parameter search again for 15 epochs, but with 2-fold cross-validation. 2-fold cross-validation is analogous to a simple split of the total training dataset into training data and validation data. However, rather than some 80/20 split, here the split is 50/50 and the model is trained on both sets. Although this is less optimal, and the models performance is now more prone to individual outliers, this choice is still supported due to the huge amounts of not-that-high variance data.

The bigger parameter search uses the following settings:
$\beta \in \{0.8, 0.9, 0.95\}$, $n_{\text{hidden}} \in \{400, 800, 1200\}$, batch size $\in \{64, 128, 256\}$, t_{window} fixed to 2000 and target ratio fixed to 0.8/0.2. The mean accuracy is shown in Table 2.

What can already be seen is that the accuracy of correct classification on the validation set is very high. The accuracies of the best performing models are very close together regardless of the parameter settings, except that $\beta = 0.95$ does not appear in the top ten. The same behaviour is expected when choosing parameter setting for the final model and it's result on the test dataset.

Accuracy	β	n_{hidden}	Batch size
0.9890	0.9	800	64
0.9889	0.8	1200	128
0.9885	0.8	800	128
0.9885	0.8	1200	256
0.9875	0.8	800	64
0.9873	0.8	400	64
0.9863	0.8	800	256
0.9857	0.8	400	128
0.9849	0.9	1200	64
0.9843	0.9	400	64

Table 2: Settings of the top ten mean accuracies of the bigger parameter search on the validation sets. Accuracy and variance are taken at 15 epochs with a 2-fold cross-validation. With $\beta \in \{0.8, 0.9, 0.95\}$, $n_{\text{hidden}} \in \{400, 800, 1200\}$, batch size $\in \{64, 128, 256\}$, t_{window} fixed to 2000 and target ratio fixed to 0.8/0.2.

4 Results

Following the hyper-parameter search, one set of parameters can be chosen for the final model. A model with 800 hidden neurons, a beta of 0.9, a window length of 2000 and a wrong/false ratio of 0.8 : 0.2 is trained using a batch size of 128 on the whole training dataset of N-MNIST for 15 epochs. After training, the model is evaluated on the N-MINST testing dataset. The accuracy on the testing dataset is 97.054%.

This test accuracy is slightly lower than the validation accuracies from the hyper-parameter search. Presumably this may be caused by the big amount of models in the hyper-parameter search, from which one could be 'lucky' and perform particularly well on the way the data was split.

As of writing this paper, the highest published accuracy achieved on the N-MNIST dataset is 0.996, published in the year 2020 [17]. So the final model described in this paper is slightly less accurate. This is not a surprise as the MLP-like architecture is also kept relatively simple.

The N-MNIST (as well as the MNIST) dataset lends itself very well to machine learning tasks with neural networks. The high accuracy achieved in this paper also affirms that.

Looking back at the hyper-parameter search, Table 1 and Table 2, the accuracies of the different models are all very close together. When the benefit on accuracy are so marginally dependent on these parameters, one could also consider other factors to choosing the "best model". For example, using less amount of neurons in the hidden layer is more economical, it gives faster training and less energy use.

5 Discussion

One should keep in mind that 'regular' digital computers and hardware was used in this project. Spiking Neural Networks are best used on specialized hardware, such as the Intel's Loihi chip [3]. When using specialized hardware, Spiking Neural Networks can be much more efficient in terms of (electric) energy used [3]. As students, this hardware was not available to us and the aim of this project is more along the lines of demonstrating the use of Spiking Neural Networks on digit classification.

In this project a Leaky Integrate-and-Fire neuron model has been used. However there is a variety of different spiking neuron models that were not considered. A further study could result in finding a more suitable model for the given task. Nonetheless, due to the high accuracy of the final results, it can be deduced that the use of LIF neuron model is sufficient for achieving the goal of this project.

All training and testing data in the N-MNIST dataset [9] was used. Leading to long training times for every epoch. Using a subset of the data for the hyper-parameter search might have been beneficial such that a bigger parameter space could have been considered in the same or less time.

Speaking of the hyper-parameter search, only one option for the L2-regularization term λ was considered (subsection 3.6). One could vary this in the hyper-parameter search as well to regulate the flexibility of the model and prevent overfitting.

It is typically a good thing to implement an early-stopping technique in machine learning tasks. Early stopping helps prevent overfitting by stopping the training of a model once the validation error is no longer improving (according to some criteria). In this project it was observed that the validation errors for the best performing models were always increasing for the duration of the training (very slightly at later epochs), so no early stopping was implemented.

Often, dimension reduction plays a role in a machine learning pipeline. In this project, the binning of spikes into frames (subsection 3.2) could be viewed as dimension reduction (Where a larger time window t_{window} reduces the dimensionality more). Other forms of dimension reduction have not been considered, as it is hard to retain the spiking properties of the data while also decrease the number of dimensions.

In this project we have chosen to go for a simple MLP architecture, as suggested in the project discussion. Convolutional Neural Networks (CNNs) are neural networks that work especially well on image data [18]. There has been showed that using convolutional layers within a NN can boost the performance of the network. This has also already been shown in combination with spiking neural networks as shown in [17]. Hence, adding convolutional layers to this MLP architecture are likely to improve the model.

6 Conclusion

Spiking Neural Networks (SNN) are a more neuromorphic extension of Neural Networks. They incorporate the concept of the spiking behaviour of information passage, which biologically takes the form of a short electrical signal that is passed between the neurons. This makes it not only biologically plausible, but also extremely energy efficient when implemented on neuromorphic hardware [1].

The goal of this project was demonstrating the efficiency of a Spiking Neural Network through classification of handwritten digits from the N-MNIST dataset. For that a Leaky Integrate and Fire Neuron Model (LIF) has been used, which describes the mechanics of a spiking neuron. To counter the *dead neuron problem*, as defined in subsection 2.2, an '*arctan*' surrogate gradient was utilized.

The input data coming from the N-MNIST dataset has been binned into frames. Through hyper-parameter search it has been found that a suitable window length t_{window} for each frame is 2000. The architecture used resembles a multilayer perceptron with 1 hidden layer. The optimal value for the number of neurons in the hidden layer has been found as 800, while value for the membrane potential decay rate β equals 0.9. Mean Square Error Spike Count Loss has been used as the objective function. Additionally, a *target spike ratio*, found to equal 0.8 : 0.2, has been enforced to further truncate the *dead neuron problem.* To prevent overfitting L2-regularisation has been used. Finally, through hyper-parameter search, the batch size, which determines the number of observation passed through the network before weight re-evaluation, has been found as 128.

The Spiking Neural Network was implemented in python 3.11.3 with use of the *torch* library and *SNNtorch* library. The N-MNIST data has been sourced from *tonic.* The high-performance computer cluster Hábrók has been used to shorten the computation time of the hyper-parameter search.

The final model, using the optimal parameters found, has been trained on the N-MNIST dataset for 15 epochs. The accuracy on the testing dataset is 97.054%. In view of the high accuracy it can be concluded that the Spiking Neural Network has been successfully trained to classify the handwritten digits of the N-MNIST dataset. Through this simple classification task one can fathom the potential of Spiking Neural Networks. Considering the probable implementation on neuromorphic hardware, there is a vast amount of research that can be conducted on this branch of Machine Learning.

Acknowledgement

We thank the Center for Information Technology of the University of Groningen for their support and for providing access to the Hábrók high performance computing cluster.

References

[1] 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.

[2] A. Özer, "A brief history of the neural networks," Oct 2023.

[3] 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.

[4] R. Massa, A. Marchisio, M. Martina, and M. Shafique, "An efficient spiking neural network for recognizing gestures with a dvs camera on the loihi neuromorphic processor," in *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–9, 2020.

[5] A. Viale, A. Marchisio, M. Martina, G. Masera, and M. Shafique, "Carsnn: An efficient spiking neural network for event-based autonomous cars on the loihi neuromorphic research processor," in *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–10, 2021.

[6] K. Buettner and A. D. George, "Heartbeat classification with spiking neural networks on the loihi neuromorphic processor," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 138–143, 2021.

[7] N. Brunel and M. van Rossum, “Lapicque’s 1907 paper: From frogs to integrate-and-fire,” *Biological cybernetics*, vol. 97, pp. 337–9, 01 2008.

[8] E. O. Neftci, H. Mostafa, and F. Zenke, “Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks,” *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 51–63, 2019.

[9] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, “Converting static image datasets to spiking neuromorphic datasets using saccades,” *Frontiers in Neuroscience*, vol. 9, 2015.

[10] Y. LeCun, C. Cortes, and C. Burges, “Mnist handwritten digit database,” *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[11] C. Posch, D. Matolin, and R. Wohlgenannt, “A qvga 143 db dynamic range frame-free pwm image sensor with lossless pixel-level video compression and time-domain cds,” *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 259–275, 2011.

[12] H. Rebecq, G. Gallego, E. Mueggler, and D. Scaramuzza, “Emvs: Event-based multi-view stereo—3d reconstruction with an event camera in real-time,” *International Journal of Computer Vision*, vol. 126, 12 2018.

[13] J. K. Eshraghian, M. Ward, E. 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.

[14] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[15] G. Lenz, K. Chaney, S. B. Shrestha, O. Oubari, S. Picaud, and G. Zarrella, “Tonic: event-based datasets and transformations.,” July 2021. Documentation available under https://tonic.readthedocs.io.

[16] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017.

[17] A. Samadzadeh, F. S. T. Far, A. Javadi, A. Nickabadi, and M. H. Chehreghani, “Convolutional spiking neural networks for spatio-temporal feature extraction,” 2021.

[18] M. Valueva, N. Nagornov, P. Lyakhov, G. Valuev, and N. Chervyakov, “Application of the residue number system to reduce hardware costs of the convolutional neural network implementation,” *Mathematics and Computers in Simulation*, vol. 177, pp. 232–243, 2020.