UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Roberta Pärna

Impact of Initialization Methods on Energy Requirements in SNNs

Bachelor's Thesis (9 ECTS)

Supervisors:
Ahmed Abdulmajeed A Sabir, PhD
Rajesh Sharma, PhD
Shirin Dora, PhD

Tartu 2025

Impact of Initialization Methods on Energy Requirements in SNNs

Abstract:

Spiking neural networks (SNNs) are widely recognized as an energy efficient alternative to artificial neural networks (ANNs). A lot of research has been done on how to build the most energy efficient SNNs without sacrificing competitive performance, however initialization of learnable parameters still needs to be studied. In this work several SNNs are built for image classification on the MNIST and FashionMNIST datasets, and the impact of the choice for the initial values of the learnable parameters in the network is analysed by comparing the average number of spikes in the networks. Models initialized with weights from a low range are found to produce significantly less spikes than other models. The initialization of neuronal time constants does not impact the number of spikes produced.

Keywords: Neural networks, energy efficiency

CERCS: P170 Computer science, numerical analysis, systems, control; P175 Informatics, systems theory; P176 Artificial intelligence

Initsialiseerimismeetodite mõju impulss-neurovõrkude energiakulule

Lühikokkuvõte:

Impulss-neurovõrgud on laialdaselt tunnustatud kui energiasäästlikud alternatiivid tavalistele neurovõrkudele. On põhjalikult uuritud, kuidas ehitada võimalikult energiasäästlikke impulss-neurovõrke ilma konkureerivaid tulemusi ohverdamata, kuid õpitavate parameetrite initsialiseerimine vajab veel uurimist. Selles töös ehitatakse mitu impulss-neurovõrku piltide klassifitseerimiseks MNIST ja FashionMNIST andmestikel. Õpitavate parameetrite algsete väärtuste mõju analüüsitakse võrreldes keskmist impulsside arvu võrkudes peale treenimist. Leitakse, et mudelid, mille kaalud initsialiseeritakse madalamast vahemikust toodavad oluliselt vähem impulsse kui teised mudelid. Neuronite ajakonstantide initsialisatsioon impulsside arvu treenitud mudelis ei mõjuta.

Võtmesõnad: Neurovõrgud, energiasäästlikus

CERCS: P170 Computer science, numerical analysis, systems, control; P175 Informaatika, süsteemiteooria; P176 Tehisintellekt

Contents

1. Introduction

The use of artificial intelligence has rapidly increased in recent years and only continues to grow. Many people use AI-powered applications, such as chatbots and image generators, daily, without ever considering the resources that are required to process their requests. Most of this software utilizes artificial neural networks (ANNs) as its backbone. Training and using these ANNs, however, requires a massive amount of energy. Considering the widespread use of ANNs in contrast to their high energy costs, the question arises: is there a more energy-efficient alternative?

Taking inspiration from the most powerful and energy-efficient computer of them all - the brain, spiking neural networks (SNNs) have received a lot of attention as a low energy cost alternative to ANNs. While ANNs were already created to resemble the behaviour of a brain, SNNs take it a step further. They attempt to mimic the behaviour of biological neurons by using abstractions of electrical impulses or spikes as their method of communication. This results in models that can reach a competitive performance in tasks like image classification, with a much lower energy cost.

A lot of work has been done on different techniques to optimize the creation of SNNs to achieve the best energy efficiency but there are still some questions that remain unanswered. This work aims to study how the choices in the initial values of trainable parameters will impact the energy costs of a spiking model after training. To achieve this, several spiking models are trained for the image classification task on two datasets, using different ranges for initial values of some learnable parameters in the networks. They are then compared on classification accuracy and the average number of spikes they produce after training.

Chapter 2 provides the theory behind spiking neurons and the methods used to train SNNs. Chapter 3 describes the details of the used models and experiments. In chapter 4 the results of the experiments are presented and analysed. Chapter 5 provides a conclusion for this thesis.

2. Background

This section will explain the main ideas behind SNNs, as well as explore used datasets and previous research that has been done on the topic.

2.1 Spiking Neural Networks

SNNs are ANNs that aim to more closely mimic the behaviour of naturally occurring neural networks. They consist of spiking neurons that are modelled as abstractions of biological neurons. These spiking neurons transmit information in the form of discrete spikes, with the information being carried in the timing of the spikes. This requires input data to also be encoded into spikes. This can be achieved using either rate encoding, where the information is stored in the frequency of spikes, or time based encoding, where the information is carried in the time during the simulation when the spike is fired.

The sparsity of these binary spikes makes SNNs more energy-efficient by nature as computing them requires less computational resources than the continuous signals of the neurons in an ANN. The energy requirements of SNNs can therefore be compared by analysing the amount of spikes they produce. SNNs can be simulated on CPUs and GPUs but there is specialized neuromorphic hardware such as the IBM TrueNorth chip [1] where the efficiency of SNNs can be observed in practice.

2.1.1 Neuron Models

Biological neurons communicate by accumulating signals from other neurons until a threshold is exceeded, at which point the neuron is triggered to send out its own signal, also called a spike. This process is mathematically described by the Hodgkin–Huxley model [2]. While highly biologically accurate, the complexity of the equations in this model makes it computationally expensive and therefore difficult to use. A more abstract version of this model was introduced as early as 1906 by Lapicque [3] as the integrate-and-fire (IF) model. The IF neuron can predict the evolution of the membrane potential and spike firing rate of the neuron but fails to take into account the decay of the membrane potential, a mechanism observed in the behaviour of natural neurons which occurs due to diffusion.

More complicated iterations of the model aim to combat this problem. The leaky integrate-and-fire (LIF) model presents equations that represent the dynamics of membrane potential and include a term that aims to mimic the loss of potential through time. This model can be used to

derive Formula 1 used to calculate membrane potential at a given time [4]

$$U[t] = \beta U[t-1] + (1-\beta) I_{\text{in}}[t] \tag{1}$$

where U is the membrane potential, t is the discrete time and β is the decay rate. An outgoing spike S_{out} is generated when the membrane potential exceeds the threshold θ as defined in Formula 2 [4].

$$S_{\text{out}}[t] = \begin{cases} 1, & \text{if } U[t] > \theta \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

The spike generation is illustrated in Figure 1 [5]. After a spike is generated, the membrane potential is reset. The two most common strategies for this are reset by subtraction, where the membrane potential is reduced by a value equal to the threshold, and reset to constant, where the membrane potential is set to zero. The leaky integrate-and-fire neuron model is still commonly used in SNNs and has been shown to achieve competitive results [6].

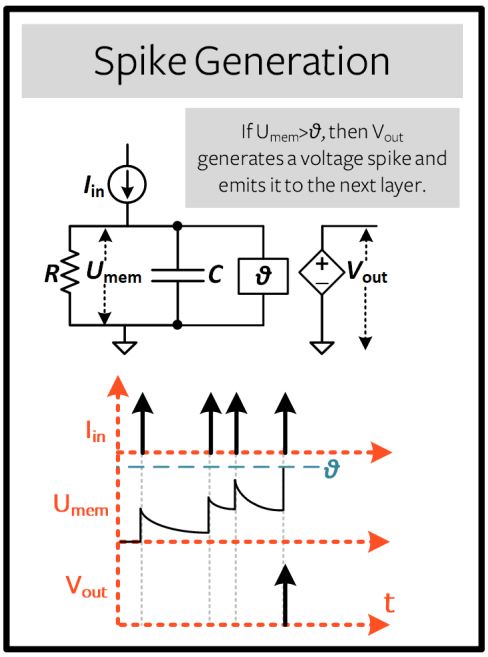

Figure 1. The spike generation of a LIF neuron [5].

2.1.2 Learning Algorithm

SNNs can be trained natively using error back-propagation or by converting a trained ANN into a spiking network. The binary nature of the output of neurons poses a major challenge for training SNNs using back-propagation. While a change in weight would result in a change in membrane potential, unless this causes the potential to reach the threshold, this will not result in a change in the output of the neuron. Also known as the ”dead-neuron” problem, this results in gradients that do not enable learning.

Backpropagation is adapted for SNNs in the backpropagation through time (BPTT) algorithm. BPTT works by taking into account the loss of all previous time steps as well as the current time step. The gradients of all these losses are summed up to find the global gradient. Using BPTT requires employing surrogate gradients to smooth out the spikes. To achieve this, the actual derivative of the spike is replaced by some other function [7]. While spike firing is still required to trigger weight updates, this allows errors to propagate to earlier layers [4]. A large variety of

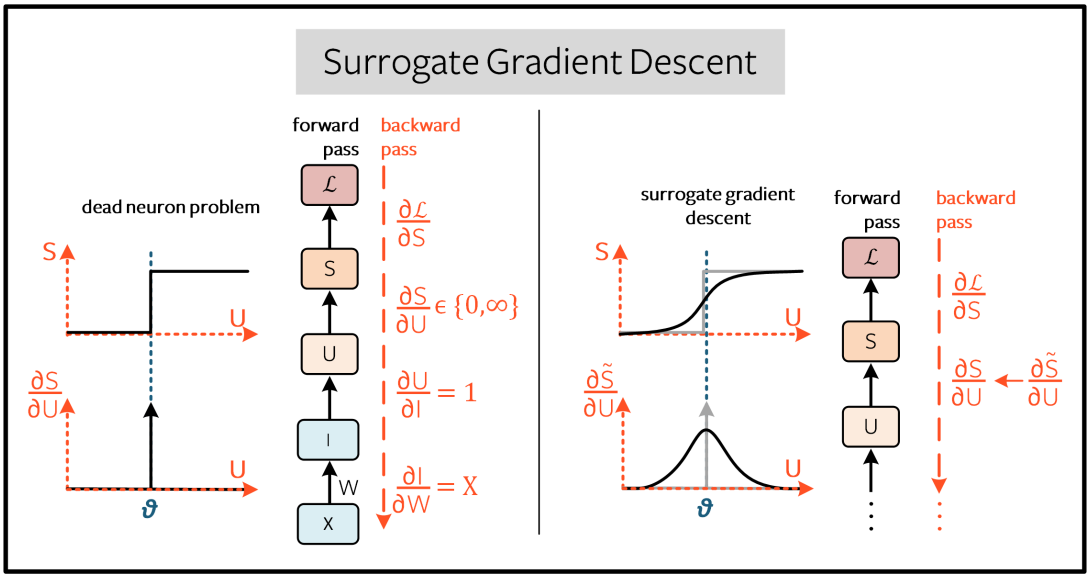

Figure 2. The figure on the left visualizes the dead neuron problem. The figure on the right shows how surrogate gradient descent is used to smooth out spikes [9].

functions can be used as the surrogate gradient with different results, depending on the network and problem [8]. Surrogate gradients help circumvent the ”dead-neuron” problem as illustrated in Figure 2 [9] and are widely used when natively training SNNs [10, 11].

The problem of the non-differentiability of spikes does not need to be addressed when using the method of converting from a traditional artificial network. In this approach, the activation functions in the original network are converted into spiking neurons, while preserving the structure and parameters [12]. One of the main advantages of this method is that the SNN model mostly retains the performance of the original model. This means that it is possible to take advantage of state-of-the-art techniques for training ANNs, such as capsule network designs [13] or mixed sample data augmentation [14] when training the original model and the results could be reflected in the converted model. This approach enables generating SNNs that suffer minimal performance loss compared to their original highly accurate ANN counterparts, but are able to work on much more energy-efficient hardware [15].

2.2 Previous Work

The problem of finding the best method for converting ANNs to SNNs for an energy efficient network has been addressed by Bing Han et al. [16]. They showed that time based encoding resulted in a network that uses significantly less operations than a model converted using rate based encoding while surpassing it in performance on image classification tasks.

The low energy requirements of the converted SNN can also be achieved by training the original network using an approach that specifically aims to minimize the activity in the network after conversion. Martino Sorbaro et al. [17] proposed a training strategy that included energy costs in the loss. They found that it was necessary to use additional techniques such as quantizing network activity to integer activations in order to make this approach plausible. This technique resulted in models that demonstrated much lower energy requirements with only a small loss in performance.

A method for natively training an SNN with low energy requirements and performance that could match state-of-the-art ANNs was first demonstrated by Ana Stanojevic et al. [18] in 2024. They used time-to-first-spike encoding and a complex initialization strategy that ensured the stability of neurons during learning.

Energy efficiency focused training has also been attempted by including a penalty term into the surrogate gradient that discourages high numbers of spikes [19]. The impact of the network architecture on the performance and energy cost was studied by Byunggook Na et al. [20] who also proposed a framework called AutoSNN for automatically generating SNNs that outperform handcrafted networks.

2.3 Datasets

The models presented in this work are studied with inputs from two image datasets: MNIST [21] (see Figure 3) and FashionMNIST [22] (see Figure 4). Both of the datasets are widely available and commonly used to train machine learning models for image classification.

The MNIST dataset consists of 60000 training examples and 10000 test examples of 28x28 pixel images from 10 classes and was first introduced by Yann LeCun et al. [21] in 1998. The images are grayscale and depict handwritten numbers collected from the workers of the United States Census Bureau and high school students. Examples of images from the dataset with their corresponding labels are provided in Figure 3.

FashionMNIST was presented in 2017 by Han Xiao et al. [22] as a response to the popularity of MNIST. Their aim was to create a dataset that could rival MNIST in its accessibility and ease of use while at the same time providing a more challenging classification task. FashionMNIST follows the same structure as MNIST, consisting of 70000 grayscale images from 10 different classes with a size of 28x28 pixels. The images in the FashionMNIST dataset depict different

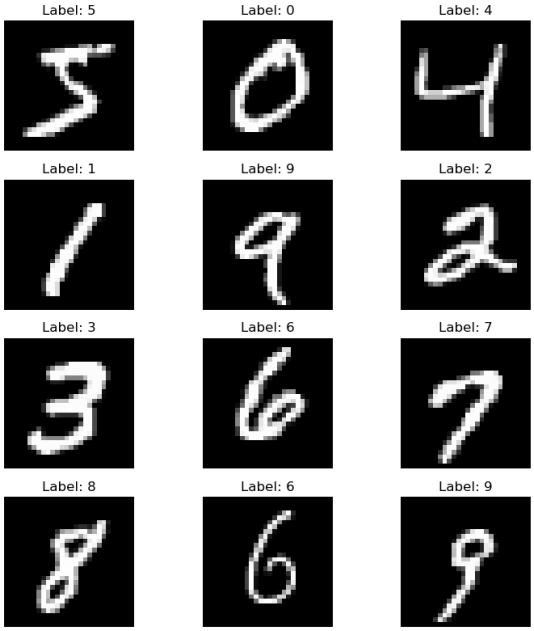

Figure 3. Examples from the MNIST dataset

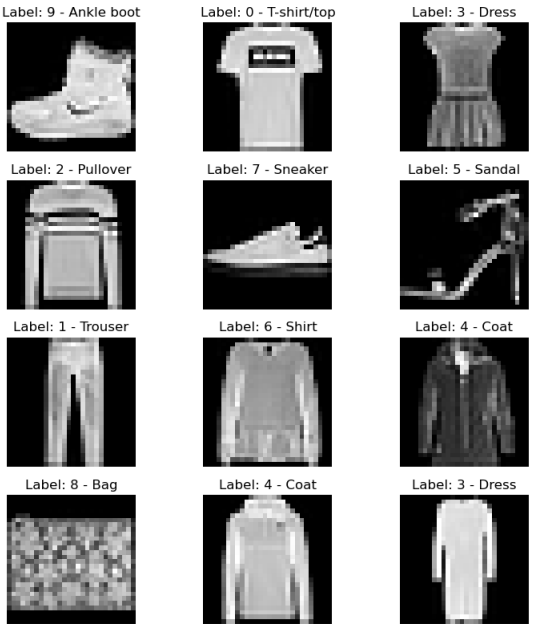

Figure 4. Examples from the FashionMNIST dataset

categories of fashion items collected from product photos in the online store Zalando. Examples of images from this dataset with labels are shown in Figure 4.

3. Method

This section provides the details of all used models and the description of performed experiments.

3.1 Network Architecture

All networks presented in this work use leaky integrate-and-fire neurons and consist of five convolutional layers followed by a fully connected layer as shown in Figure 5. For all the convolutional layers a kernel of size 3×3 is used with $stirde = 2$ and $padding = 1$. The input is encoded into 128 features after which the amount of features is doubled with every convolutional layer up to the last layer where the amount of features is halved. The membrane potential is reset by subtracting the value of the threshold. The class for which the most spikes are generated in the final layer is considered the predicted class.

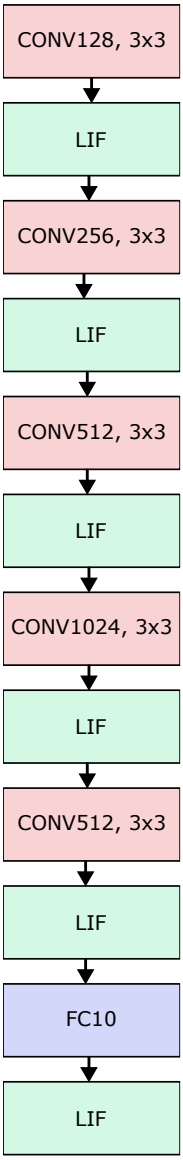

Figure 5. The architecture of proposed networks.

Error backpropagation for SNNs is used to update weights, thresholds and neuronal time constants. The loss is calculated by applying the cross entropy loss function at every time step and accumulating the losses. The cross entropy loss function is a combination of the logarithm of the Softmax function simplified as[1]

$$LogSoftmax(x_i) = \log\left(\frac{\exp(x_i)}{\sum_j \exp(x_j)}\right) \tag{3}$$

and the negative log likelihood loss [23] given as [2]

$$l(x, y) = \sum_{n=1}^{N} \frac{1}{\sum_{n=1}^{N} w_{y_n}} l_n, \quad l_n = -w_{y_n} x_{n,y_n} \tag{4}$$

where x is the input, y is the target, N is the batch size and w is the weight. The gradient of the shifted arc-tan function is used as the surrogate gradient as given in Formula 5 [24] with $\alpha = 2$

[1] https://pytorch.org/docs/stable/generated/torch.nn.LogSoftmax.html (29.04.2025)
[2] https://pytorch.org/docs/stable/generated/torch.nn.NLLLoss.html (29.04.2025)

for all experiments.

$$\frac{\partial S}{\partial U} = \frac{1}{\pi} \times \frac{1}{1 + (\pi U \frac{\alpha}{2})^2} \tag{5}$$

The threshold for membrane potential is initialized as $\theta = 1$ in all experiments.

The weights are initialized from a normal distribution $\mathcal{N}(\mu, \sigma)$, with μ in all models set to 0.1. The impact of initial weights is studied for three values of σ: -0.01, 0 and 0.01. The decay rate of the membrane potential β is studied for three different intervals of uniform distribution: $[0, 0.34]$, $[0.34, 0.67]$ and $[0.67, 1.0]$. These will be referred to as low, medium and high ranges in this work.

3.2 Experiments

For every combination of proposed initial values for β and σ, a network is trained for 50 epochs. The learnable parameters are updated using the Adaptive Movement Estimator (Adam) optimizer. Adam works by assigning parameters individual learning rates and adjusting them with respect to the previous gradients of the parameter [25]. This encourages learning when the gradient of a parameter is moving consistently and discourages learning when the gradient moves chaotically. The learning rate of the optimizer is initially set to $1e-2$ and is multiplied by 0.1 every ten epochs. The input information is encoded into spikes using rate encoding during a simulation length of 20 ms.

The performance of the SNN models is estimated by calculating the classification accuracy of a trained network on the test datasets as

$$accuracy = \left(\frac{N_c}{N_a} \right) \tag{6}$$

where N_c is the amount of correct predictions and N_a is the total amount of predictions. The energy efficiency of the trained models is estimated by the amount of spikes generated per layer by a trained model when classifying images in the test dataset. To enable the comparison to be independent from the sizes of different layers, the amount presented is an average per neuron given as

$$S_l = \frac{\sum_{n,t,i} s_{ln}(t,i)}{N_{img} \times N_n(l)} \tag{7}$$

where S_l is the reported amount of spikes for layer l, N_{img} is the total number of images classified, $N_n(l)$ is the amount of neurons in layer l and $s_{ln}(t,i)$ is the amount of spikes generated by neuron n in layer l during the classification of image i at time step t. The implementation details and source code are provided in Appendices 5.2-5.3.

4. Results

In this subsection the results of the experiments are presented and analysed. The individual results for all experiments are included in the Appendix 5.1. All results are presented as averages over 10 runs with the standard deviation provided in brackets.

4.0.1 Weight initialization

Tables 1 and 2 contain results for different weight initializations averaged across all beta ranges to analyse the impact of initial weights regardless of the initial beta value, obtained on the MNIST and FashionMNIST datasets respectively.

Table 1. Average results obtained on the MNIST dataset for different weight initializations across all beta ranges.

Mean weight	Average number of spikes (layer-wise)	Accuracy (%)
-0.01	0.20, 0.58, 0.91, 2.07, 9.37, 0.10 (0.09), (0.22), (0.27), (0.56), (0.69), (0.00)	97.39 (1.95)
0	0.44, 1.36, 1.61, 2.27, 9.48, 0.10 (0.12), (0.38), (0.17), (0.31), (0.41), (0.00)	98.18 (0.19)
0.01	1.34, 1.24, 1.6, 2.65, 7.28, 0.10 (1.95), (0.87), (0.44), (1.23), (2.87), (0.00)	97.44 (1.26)

Table 2. Average results obtained on the FashionMNIST dataset for different weight initializations across all beta ranges.

Mean weight	Average number of spikes (layer-wise)	Accuracy (%)
-0.01	0.66, 1.02, 1.13, 2.86, 8.86, 0.14 (0.41), (0.46), (0.44), (1.05), (1.01), (0.01)	77.35 (4.39)
0	0.79, 1.75, 2.18, 4.12, 9.36, 0.14 (0.15), (0.25), (0.27), (0.48), (0.43), (0.01)	76.14 (2.17)
0.01	1.46, 2.26, 2.62, 4.67, 9.72, 1.44 (0.62), (0.47), (0.49), (0.45), (0.41), (0.01)	75.64 (1.52)

It can be observed that models with different weight initialization strategies have similar classification accuracies and there is not a lot of variance in the classification accuracy between experiments. The unpaired t-test is used to confirm that the mean accuracy of the low range is not

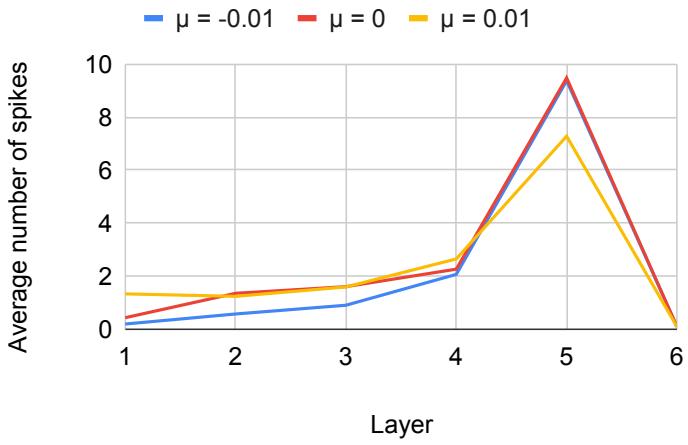

Figure 6. Comparison between different weight ranges across all beta ranges on the MNIST dataset.

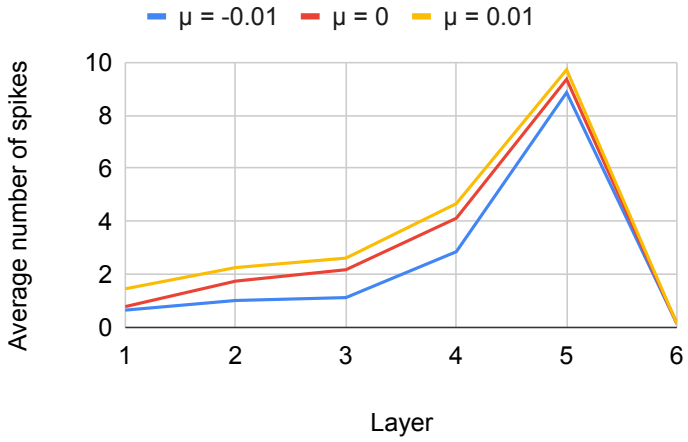

Figure 7. Comparison between different weight ranges across all beta ranges on the FashionMNIST dataset.

significantly different from the mean accuracies of the medium and high ranges at a confidence interval of 95%.

Figures 6-7 present a visual comparison of the results between different initial weight ranges from the experiments. The unpaired t-test is used to compare the results from each layer separately. The null-hypothesis is that the average number of spikes for the low range is not significantly higher than the average number of spikes for the high range.

When comparing high and low ranges at a confidence interval of 95% this hypothesis is rejected for layers 2 and 3 on the MNIST dataset and layers 1 - 5 on the FashionMNIST dataset. On layer 5 of the MNIST dataset the opposite is true - the average number of spikes for the high

range is significantly lower than the average number of spikes for the low range. It should also be noted that the deviation in the number of spikes for the first and fifth layers is much higher in the high range than in other models. When comparing low and medium ranges at a confidence interval of 95% this hypothesis is rejected for layers 1 - 3 on the MNIST dataset and layers 2 - 4 on the FashionMNIST dataset.

It can also be observed in Table 1 that the standard deviation is significantly higher in experiments on the higher weight range. This could indicate that initializing weights with a lower mean results in more predictability in the model after it is trained. These results clearly show that the networks initialized with a lower mean weight produce less spikes after training.

4.0.2 Neuronal Time Constants

Table 3. Average results obtained on the MNIST dataset for different beta initializations across all weight initialization strategies.

Beta range	**Average number of spikes (layer-wise)**	**Accuracy (%)**
Low	0.94, 1.17, 1.48, 2.48, 9.51, 0.10 (1.74), (0.85), (0.51), (1.13), (0.54), (0.00)	97.55 (1.25)
Medium	0.52, 0.96, 1.3, 2.27, 8.08, 0.10 (0.97), (0.45), (0.42), (0.51), (2.44), (0.00)	97.55 (1.91)
High	0.52, 1.05, 1.33, 2.25, 8.54, 0.10 (0.67), (0.61), (0.40), (0.73), (2.27), (0.00)	97.92 (0.71)

Table 4. Average results obtained on the FashionMNIST dataset for different beta initializations across all weight initialization strategies.

Beta range	**Average number of spikes (layer-wise)**	**Accuracy (%)**
Low	0.97, 1.74, 2.03, 4.04, 9.35, 0.14 (0.47), (0.54), (0.64), (0.92), (0.74), (0.01)	75.46 (3.23)
Medium	0.96, 1.59, 1.87, 3.69, 9.31, 0.14 (0.61), (0.67), (0.81), (1.12), (0.73), (0.01)	76.94 (2.55)
High	0.98, 1.7, 2.03, 3.92, 9.28, 0.14 (1.95), (0.87), (0.44), (1.23), (2.87), (0.00)	76.73 (1.26)

Tables 3 and 4 contain results for different beta initialization ranges averaged across all weight ranges to analyse the impact of beta initialization regardless of the weight initialization strategy,

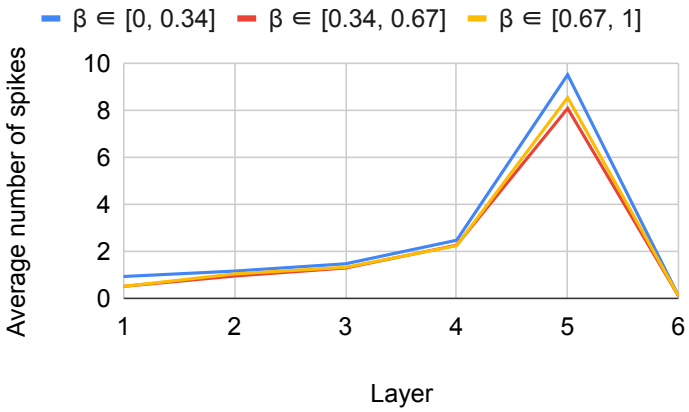

Figure 8. Comparison between different beta ranges across all weight ranges on the MNIST dataset.

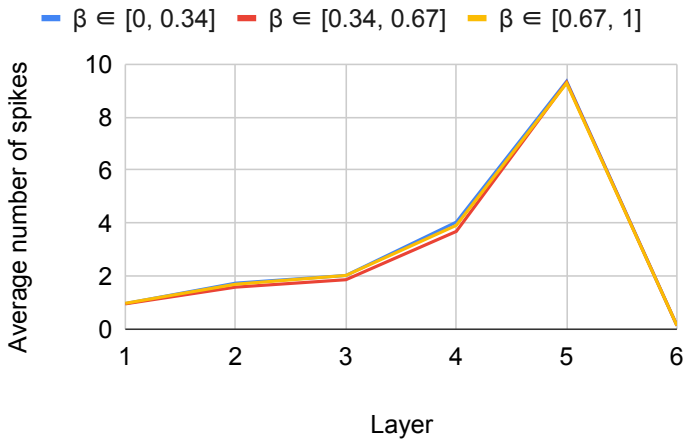

Figure 9. Comparison between different beta ranges across all weight ranges on the FashionMNIST dataset.

obtained from the MNIST and FashionMNIST datasets respectively. It can be observed that models with different initial neuronal time constants result in a similar classification accuracy. The unpaired t-test is used to confirm that the mean classification accuracy of the high range is not significantly different from the mean accuracies of the low nd medium ranges at a confidence interval of 95

Figures 8 and 9 provide a visual comparison of these results. The results are compared using the unpaired t-test. The null-hypothesis is that the average number of spikes for the medium beta range is not significantly lower than than the average number of spikes for the high and low ranges. When comparing the low and medium ranges at a confidence interval of 95% this

hypothesis is not rejected for any layers on either dataset. When comparing the high and medium ranges this hypothesis is also not rejected for any layers on either dataset.

It should be noted that in all experiments the average number of spikes per neuron increases with every layer up until the last classification layer. Neither weight initialization nor time constant initialization seem to have any impact on the classification accuracy of the model or the amount of spikes produced by the last layer of the network. It can be observed that regardless of the initialization method, the models trained for classification on the FashionMNIST dataset produce more spikes than the models trained on the MNIST dataset. This could indicate that a higher complexity of the input data requires the network to use more spikes in order to effectively communicate information between the neurons.

5. Conclusion

In a world where the everyday use of ANNs is massively widespread, and superfluous use of energy resources is often heavily criticized, SNNs could offer an energy efficient alternative to satisfy the demand for artificial intelligence powered software. Attempting to imitate the working of a brain even more closely than ANNs, SNNs have been able to achieve competitive results in many tasks, while requiring less computational resources. There have been many techniques developed for the training of SNNs that aim to optimize the performance and energy costs of the models, but a lot of essential choices made during the creation of a spiking model have not yet been thoroughly studied.

In this work several SNNs are trained with different initialization methods in order to study which initial values for learnable parameters lead to the lowest energy requirements during classification in a trained model. The results presented in this work show that the choice of these initial values has a great impact on the amount of spikes generated during classification. Lower initial weights are shown to result in a significantly more energy efficient model compared to initial weights from higher ranges. The difference in the amount of spikes generated by models initialized with the neuronal time constant β from different ranges are not considered statistically significant.

This work does not consider the impact of used hardware on the results of the experiments. It should be more thoroughly investigated if the hardware used for training has an impact on the energy efficiency of an SNN model. This work also focuses only on the fairly simple task of image classification. The impact initialization methods have on the energy efficiency of SNNs should also be studied on other tasks to determine how the nature of the task affects the energy requirements of an SNN.

In addition there are still a lot of other factors that could impact the energy requirements of a spiking model, such as other parameters, different neuron models or the structure of the network. Further research should focus on determining how all these variables affect the number of spikes a network needs to generate. It should also be studied how all these choices work together and if there are any particular combinations that are able to achieve better results than others.

References

[1] Akopyan F., Sawada J., Cassidy A., Alvarez-Icaza R., Arthur J., Merolla P., Imam N., Nakamura Y., Datta P., Nam G.-J., et al. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE transactions on computer-aided design of integrated circuits and systems* 34.10 (2015), pp. 1537–1557.

[2] Hodgkin A. L. and Huxley A. F. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology* 117.4 (1952), p. 500.

[3] Abbott L. F. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain research bulletin* 50.5-6 (1999), pp. 303–304.

[4] Eshraghian J. K., Ward M., Neftci E. O., Wang X., Lenz G., Dwivedi G., Bennamoun M., Jeong D. S., and Lu W. D. Training Spiking Neural Networks Using Lessons From Deep Learning. *Proceedings of the IEEE* 111.9 (2023), pp. 1016–1054. DOI: 10.1109/JPROC.2023.3308088.

[5] Eshraghian J. K. Tutorial 2 - The Leaky Integrate-and-Fire Neuron. (02.05.2025). https://snntorch.readthedocs.io/en/latest/tutorials/tutorial_2.html.

[6] Diehl P. U. and Cook M. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience* 9 (2015), p. 99.

[7] Neftci E. O., Mostafa H., and Zenke F. Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks. *IEEE Signal Processing Magazine* 36.6 (2019), pp. 51–63. DOI: 10.1109/MSP.2019.2931595.

[8] Zenke F. and Vogels T. P. The Remarkable Robustness of Surrogate Gradient Learning for Instilling Complex Function in Spiking Neural Networks. *Neural Computation* 33.4 (Mar. 2021), pp. 899–925. DOI: 10.1162/neco_a_01367. eprint: https://direct.mit.edu/neco/article-pdf/33/4/899/1902294/neco_a_01367.pdf. https://doi.org/10.1162/neco%5C_a%5C_01367.

[9] Eshraghian J. K. Tutorial 6 - Surrogate Gradient Descent in a Convolutional SNN. (02.05.2025). https://snntorch.readthedocs.io/en/latest/tutorials/tutorial_6.html.

[10] Bellec G., Salaj D., Subramoney A., Legenstein R., and Maass W. Long short-term memory and learning-to-learn in networks of spiking neurons. *Advances in neural information processing systems* 31 (2018).

[11] Huh D. and Sejnowski T. J. Gradient descent for spiking neural networks. *Advances in neural information processing systems* 31 (2018).

[12] Hunsberger E. and Eliasmith C. Spiking deep networks with LIF neurons. *arXiv preprint arXiv:1510.08829* (2015).

[13] Byerly A., Kalganova T., and Dear I. No routing needed between capsules. *Neurocomputing* 463 (2021), pp. 545–553.

[14] Harris E., Marcu A., Painter M., Niranjan M., Prügel-Bennett A., and Hare J. Fmix: Enhancing mixed sample data augmentation. *arXiv preprint arXiv:2002.12047* (2020).

[15] Diehl P. U., Zarrella G., Cassidy A., Pedroni B. U., and Neftci E. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. *2016 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE. 2016, pp. 1–8.

[16] Han B. and Roy K. Deep spiking neural network: Energy efficiency through time based coding. *European conference on computer vision*. Springer. 2020, pp. 388–404.

[17] Sorbaro M., Liu Q., Bortone M., and Sheik S. Optimizing the energy consumption of spiking neural networks for neuromorphic applications. *Frontiers in neuroscience* 14 (2020), p. 662.

[18] Stanojevic A., Woźniak S., Bellec G., Cherubini G., Pantazi A., and Gerstner W. High-performance deep spiking neural networks with 0.3 spikes per neuron. *Nature Communications* 15.1 (2024), p. 6793.

[19] Suetake K., Ushimaru T., Saiin R., and Sawada Y. Spiking Synaptic Penalty: Appropriate Penalty Term for Energy-Efficient Spiking Neural Networks. *arXiv preprint arXiv:2302.01500* (2023).

[20] Na B., Mok J., Park S., Lee D., Choe H., and Yoon S. Autosnn: Towards energy-efficient spiking neural networks. *International conference on machine learning*. PMLR. 2022, pp. 16253–16269.

[21] LeCun Y., Bottou L., Bengio Y., and Haffner P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[22] Xiao H., Rasul K., and Vollgraf R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).

[23] Yao H., Zhu D.-l., Jiang B., and Yu P. Negative log likelihood ratio loss for deep neural network classification. *Proceedings of the Future Technologies Conference (FTC) 2019: Volume 1*. Springer. 2020, pp. 276–282.

[24] Fang W., Yu Z., Chen Y., Masquelier T., Huang T., and Tian Y. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 2661–2671.

[25] Kingma D., Adam J. B., et al. A method for stochastic optimization. *International conference on learning representations (ICLR)*. Vol. 5. 6. San Diego, California; 2015.

Appendices

5.1 Appendix 1 - Results

This appendix contains the results of all experiments on different weight and beta ranges. The average number of spikes is provided in order for every layer of the network along with the standard deviation for the value over 10 experiments in brackets in the row below. The results for the MNIST dataset are provided in Table 5 and the results for the FashionMNIST dataset are provided in Table 6.

Table 5. Results obtained on the MNIST dataset.

Mean weight	Beta range	Average number of spikes (layer-wise)	Accuracy (%)
-0.01	Low	0.17, 0.54, 1.02, 2.14, 9.84, 0.10 (0.09), (0.24), (0.32), (0.50), (0.59), (0.00)	97.47 (0.90)
	Medium	0.2, 0.6, 0.84, 2.2, 8.70, 0.10 (0.06), (0.21), (0.3), (0.75), (1.29), (0.00)	96.81 (3.19)
	High	0.23, 0.6, 0.87, 1.88, 9.58, 0.10 (0.10), (0.22), (0.13), (0.36), (0.44), (0.00)	97.91 (0.73)
0	Low	0.4, 1.17, 1.51, 2.01, 9.17, 0.1 (0.11), (0.39), (0.07), (0.17), (0.31), (0.00)	98.32 (0.12)
	Medium	0.47, 1.47, 1.63, 2.53, 9.76, 0.1 (0.13), (0.35), (0.19), (0.25), (0.43), (0.00)	98.07 (0.20)
	High	0.47, 1.42, 1.68, 2.28, 9.51, 0.1 (0.12), (0.38), (0.17), (0.28), (0.25), (0.00)	97.70 (0.98)
0.01	Low	2.26, 1.8, 1.91, 3.3, 9.52, 0.10 (2.61), (1.11), (0.54), (1.64), (0.49), (0.00)	96.85 (1.73)
	Medium	0.89, 0.82, 1.44, 2.07, 5.78, 0.10 (1.66), (0.18), (0.25), (0.32), (2.81), (0.00)	97.77 (0.74)
	High	0.86, 1.12, 1.45, 2.59, 6.54, 0.10 (1.1), (0.78), (0.30), (1.12), (3.11), (0.00)	97.70 (0.98)

Table 6. Results obtained on the FashionMNIST dataset.

Mean weight	Beta range	Average number of spikes (layer-wise)	Accuracy (%)
-0.01	Low	0.85, 1.28, 1.34, 3.2, 8.87, 0.14 (0.44), (0.51), (0.44), (1.04), (0.94), (0.01)	74.54 (5.44)
	Medium	0.5, 0.77, 0.83, 2.37, 8.80, 0.14 (0.40), (0.35), (0.30), (0.96), (0.91), (0.01)	79.52 (2.60)
	High	0.62, 1.01, 1.22, 3.00 8.91, 0.14 (0.33), (0.39), (0.44), (1.07), (1.27), (0.01)	77.98 (3.36)
0	Low	0.77, 1.72, 2.16, 4.24, 9.41, 0.14 (0.12), (0.25), (0.24), (0.47), (0.46), (0.00)	75.33 (0.86)
	Medium	0.80, 1.86, 2.24, 4.29, 9.41, 0.15 (0.21), (0.21), (0.30), (0.36), (0.47), (0.00)	75.61 (1.04)
	High	0.8, 1.68, 2.14, 3.83, 9.25, 0.14 (0.11), (0.27), (0.27), (0.52), (0.39), (0.01)	77.50 (3.21)
0.01	Low	1.28, 2.23, 2.59, 4.68, 9.76, 0.14 (0.59), (0.37), (0.40), (0.41), (0.48), (0.00)	76.50 (1.07)
	Medium	1.58, 2.14, 2.53, 4.41, 9.72, 0.14 (0.55), (0.37), (0.27), (0.28), (0.45), (0.00)	75.68 (1.42)
	High	1.52, 2.42, 2.73, 4.93, 9.69, 0.15 (0.75), (0.62), (0.72), (0.5), (0.30), (0.01)	74.72 (1.60)

5.2 Appendix 2 - Implementation Details

The experiments were run using a batch size of 1024 with learning rate lr $= 1e{-}2$ on an AMD Radeon RX 7900 XTX GPU with 24 GB of VRAM using DirectML 0.2. For NVIDIA GPU hardware, L4 is used with 24 GB of VRAM and CUDA version 12.5. Also, the NVIDIA A100 GPU with 40 GB of VRAM was used in the experiments with a higher batch size of 2048. All experiments were implemented using the PyTorch version 2.0 and snnTorch version 0.9.1 libraries.

5.3 Appendix 3 - Source Code

The source code for all the experiments described in this paper is available at https://github.com/robertaparna/impact_of_initialization_methods_on_energy_requirements_in_SNNs.

Licence

Non-exclusive licence to reproduce the thesis and make the thesis public

I, Roberta Pärna,

1. grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the digital archives of the University of Tartu until the expiry of the term of copyright, my thesis Impact of Initialization Methods on Energy Requirements in SNNs, supervised by Ahmed Abdulmajeed A Sabir, Rajesh Sharma and Shirin Dora;
2. grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright;
3. am aware of the fact that the author retains the rights specified in points 1 and 2;
4. confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Roberta Pärna

15.05.2025