Comparison of STDP-trained SNNs and ANNs using the MNIST dataset

Neurodynamics - Group 10

Link to GitHub repository

Cornelius Wolff
Osnabrück University
cowolff@uos.de

Leonie Grafweg
Osnabrück University
lgrafweg@uos.de

Paula Heupel
Osnabrück University
pheupel@uos.de

Peter Keffer
Osnabrück University
pkeffer@uos.de

***Abstract*—Artificial Neural Networks (ANNs) are only loosely inspired by the human brain, while Spiking Neural Networks (SNNs) incorporate various concepts of it. In order to obtain a better understanding of SNNs we compare their performance in image classification to Fully-Connected ANNs using the MNIST dataset. During training, we relied on Spike-Timing-Dependent-Plasticity (STDP) as this is one of the most commonly used biologically inspired unsupervised learning rules for SNNs. Our results show that Fully-Connected ANNs outperform SNNs if trained for multiple epochs, while our implementation of the SNN didn't improve much beyond the first epoch. Still, we succeeded in providing a limited baseline comparison between ANNs and SNNs and a basic open source implementation of the latter.**

I. Introduction

Artificial Neural Networks (ANNs) are loosely inspired by the human brain, but still, there are fundamental differences in their structure and way of learning. Spiking Neural Networks (SNNs) consider spiking neurons as computational units and incorporate key aspects of biological processes. Staying close to the actual processes in the brain seems to be more promising, when trying to achieve natural intelligence, than building abstract models [18] [15]. The training of Spiking Neural Networks (SNNs) still remains a challenge in this field [20]. Furthermore, in this context there is still a lack of biologically plausible training methods because most training methods do not take biological plausibility into account and therefore do not reflect the key idea of SNNs. On the contrary, most of the current training methods apply variations of backpropagation, for example, SpikeProp [3], or even just convert classical pretrained ANNs into SNNs.
One unsupervised biologically inspired training method is based on the concept of Spike-Timing-Dependent-Plasticity (STDP) where synaptic weights are manipulated based on temporal correlations of spikes of pre- and postsynaptic neurons [14]. In this paper, we aim at providing a basic python implementation of a Spiking Neural Network using STDP and comparing that implementation to classical ANNs by training it on the MNIST dataset.

A. Research gap

After completing our literature review, we identified multiple research gaps to which we want to contribute in this term paper. For once, we aim at providing a basic and well documented implementation of a Spiking Neural Network based on the leaky integrate-and-fire neuron model (LIF) using Spike-Timing-Dependent-Plasticity as our training method. Furthermore, we want to present a basic comparison between classical Artificial Neural Networks and Spiking Neural Networks in terms of how the different neural architectures perform and how well performance scales with larger networks using a basic classification task.

II. Related work

First, our research focused on finding papers comparing SNNs with more classical ANNs using Dense layers. During this process, we found numerous papers focusing on the efficiency in terms of the required computing performance. Still, most of these works relied on converting classical pretrained ANNs to SNNs and comparisons based on these Neural Networks [8][17]. While this gives some interesting short term insights into the power efficiency of SNNs, it also takes away the opportunity to evaluate the performance of SNNs in terms of accuracy or other standards when the SNN is trained with a training method designed for such neural networks. Yet, such comparisons are important to evaluate the current state of research into the different training methods of SNNs. Furthermore, this approach of converting ANNs into SNNs limits the efficiency metric to the step of inference and therefore does not allow an evaluation of power efficiency or convergence speed during training. Again, this is a significant limitation as training classical ANNs can be very power intensive.

Additionally, we found papers that tried to present baseline datasets for comparing different SNNs against each other, a role that is commonly filled for classical ANNs with datasets such as MNIST or Cifar-10/Cifar-100 [7]. While there is definitely a need for such SNN specific baseline datasets, there is still a need for directly comparing SNNs to ANNs to evaluate strengths and weaknesses of each neural architecture.

Furthermore, while today there is a limited but existing range of libraries for the use of Spiking Neural Networks, there is still a severe lack of basic open source implementation of SNNs available on platforms such as GitHub which adhere to common coding conventions and guidelines [11][10][16]. This also presents a significant difference to classical ANNs and a hurdle for future researchers who want to contribute to basic research in this field.

III. Methodology

A. Data Preprocessing

To compare the classification performance of ANNs and SNNs we relied on the MNIST dataset, which is a widely used baseline dataset to evaluate the performance of classical ANNs [9].
While training the ANNs, image data from the MNIST dataset can be fed directly to the ANN after normalizing the image's RGB values between zero and one and converting those images into tensors. With SNNs on the contrary, the data has to be converted into spike trains. Because of time constraints, we decided to use rate encoding, as this is much easier to implement than the alternative in the form of temporal encoding [12]. Due to extensive training times, we limited the dataset for the SNN to 80 pictures per class, resulting in overall 800 pictures used per epoch.

B. Spiking Neural Networks

We based our implementation of the SNN on the leaky integrate-and-fire neuron model [2][4] and for our learning algorithm we decided to apply the Spike-Timing-Dependent-Plasticity approach introduced by Caporale and Dan [5]. This approach relies on strengthening or weakening the connections between neurons, depending on whether the post-synaptic action potential occurred before or after the pre-synaptic action potential.
To check how well the classification performance of SNNs scales with larger networks, we tested a range of different layer sizes starting with 50 neurons, then doubling the network size until ending up at a network size of 1000 neurons. Each network was trained for 5 epochs. The accuracy of the trained network was then calculated using a separate test dataset, which was not used during training. Concerning most of the other relevant hyperparameters, we relied on values we extracted from existing literature [19]. Only the values for the Spike Drop Rate and the Threshold Drop Rate were determined by ourselves. This resulted in the hyperparameters shown in Table I being used for each Neuron during training. Additionally, we used the hyperparameters shown in Table II for STDP. In order to make sure that our limited dataset of 80 pictures per class does not result in worse performance compared to larger datasets, we ran training once with different dataset sizes and a layer size of 800.

TABLE I
NEURON HYPERPARAMETERS

Hyperparameter	***Value***
Inhibitory Potential	-100
Spike Threshold	-55
Hyperpolarization Potential	-90
Spike Drop Rate	0.8
Threshold Drop Rate	0.4
Min Weight	0.00001
Max Weight	1.0

TABLE II
STDP HYPERPARAMETERS

Hyperparameter	***Value***
STDP Offset	0
Sigma	0.01
Tau-Plus	5
Tau-Minus	5
Mu	0.9

C. Classical Artificial Neural Networks

While there is a lot of literature stating that Convolutional Neural Networks are outperforming pure Dense Networks when it comes to image classification tasks, we decided to stick to relatively simple Artificial Neural Networks only containing one or two dense hidden layers [6]. We chose to implement our ANNs with the popular deep learning framework TensorFlow and to go with this simpler network architecture to achieve better comparability when it comes to scaling [1]. In this instance, we started training with a hidden layer size of 16. To test how well the ANNs classification performances scales with larger and deeper networks, we first tried the same layer size with two hidden layers and afterwards doubled the layer size until we reached a layer size of 256 with two hidden layers.
To optimize our ANN during training, we relied on Stochastic Gradient Descent (SGD) with a learning rate of 0.001 and Sparse Categorical Crossentropy as its loss function. The batch size remained fixed at a value of 64 for all of our tests, and each ANN was trained for 15 epochs.

IV. Results

A. Model performance

First, we present the results from the training of the classical ANNs to provide a baseline for later comparison. With the hyperparameters described previously, we were able to show that relying on an ANN using only Dense Layers is a viable option to achieve good results in terms of accuracy. Furthermore, our results shown in Figure 1 give a good impression of the scalability of classical ANNs. In our use case, performance steadily increased with larger network sizes. Interestingly, in some instances, using two layers of a smaller number of neurons improved performance to a greater degree than just doubling the number of neurons.
On the other hand, the presented graph strongly indicates that

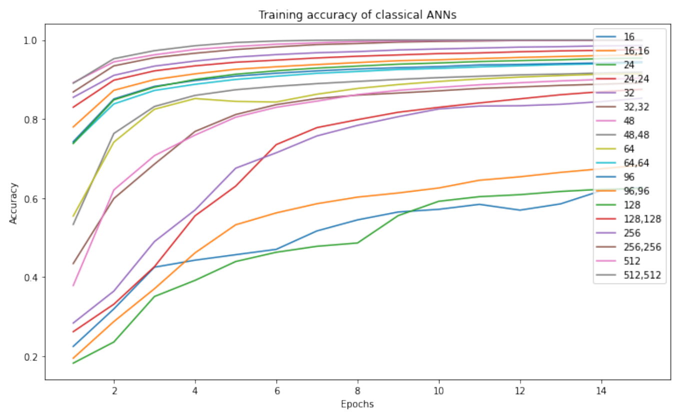

Fig. 1. Comparison of training accuracy of ANNs with different amounts of neurons and hidden layers using the MNIST dataset

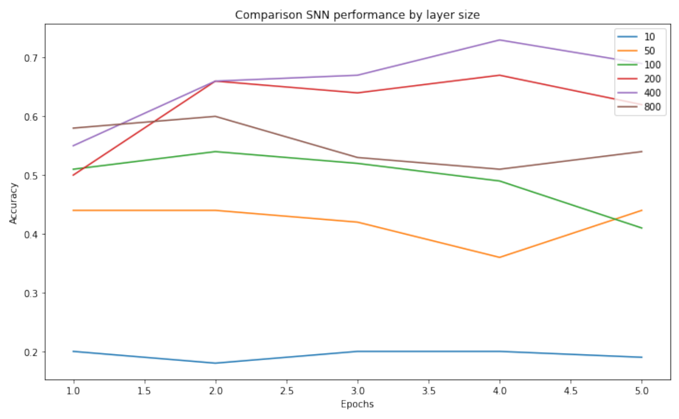

Fig. 3. Comparison of training accuracy of SNNs with different amounts of neurons using the MNIST dataset

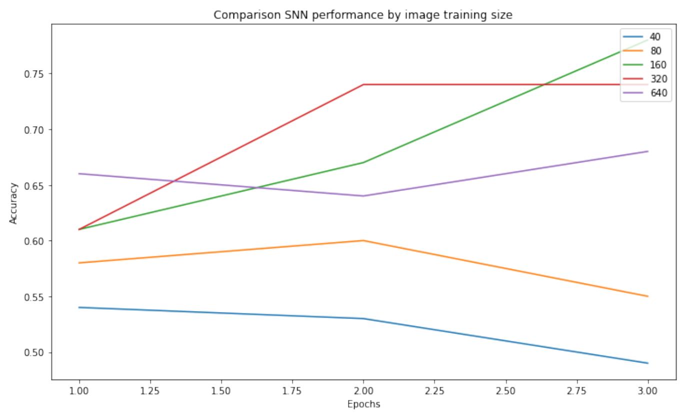

Fig. 2. Comparison of training accuracy of SNNs with different amounts of images and a layer size of 800

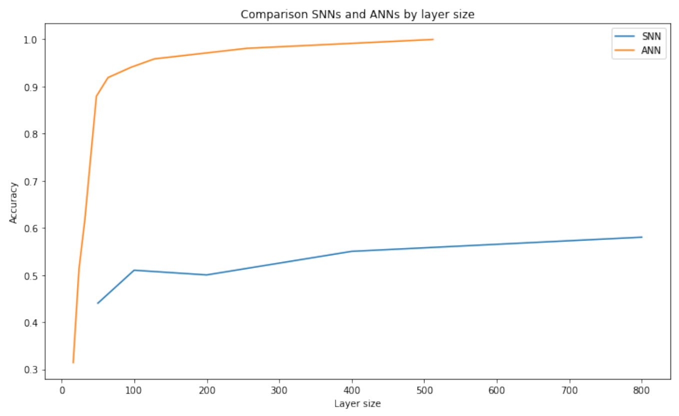

Fig. 4. Comparison of training accuracy of ANNs and SNNs scaled to their number of neurons over one epoch using the MNIST dataset

simply scaling the network up indefinitely soon suffers from vanishing returns since networks beyond a single hidden layer size of 64 perform only marginally better. Larger networks also require a lot more calculating power, as a doubling in the number of neurons also significantly increases the number of weights and activation thresholds that need to be updated.

Figure 3 displays how well the classification performance of SNNs scales with larger layer sizes and a training dataset of 80 images per class. First, it has to be noted that SNNs in our example performed significantly worse than classical ANNs, with its best results reaching an accuracy of 70 percent. Furthermore, the graphs give an indication that our implementation of an SNN doesn't profit from more training epochs as much as classical ANNs. This means that performance is often stuck at levels of 45 to 70 percent after the first epoch. Yet, this also goes to show that SNNs of the kind that we implemented require less training epochs to reach peak classification performance, which can be a significant advantage. In terms of scaling, our results show significant improvements between the range of 10 and 200 neurons, but beyond that, there was no stable long-term advantage, while 10 neurons being too small a network size to become better than slightly above chance level. Furthermore, Figure 2 indicates that training performance doesn't improve when using more than 320 images per class, while training took multiple hours per epoch for 320 images per class or more.

B. *Model comparison*

Figure 4 combines the results from Figure 1 and Figure 3 and makes the network architectures comparable in terms of scaling and classification performance, though for the SNNs we relied on the results gathered after one training epoch. In both metrics, classical ANNs outperform SNNs with ANNs achieving over 90 percent accuracy beyond a hidden layer size of 100 and SNNs never reaching more than 60 percent while also not scaling as well. Even with more training data as presented in Figure 2 the classification performance never exceeded 80 percent.
Additionally, training times per epoch were significantly longer for SNNs, but these training times are hardly comparable as we relied on TensorFlow with its CUDA optimized backend to train our classical ANNs and our own implementation of SNNs with no optimizations.

V. Discussion

A. Challenges

During our research process, we faced multiple major difficulties. One significant hurdle was that a lot of papers in this field apply SNNs to very specific problems, which results in architectures not being easily generalizable or applicable to different problems. At the same time, a lot of those papers took the easier way of converting a pretrained ANN into an SNN and not training the SNN itself. While this is understandable from a time and complexity point of view, it also resulted in us having very few sources to rely on. This problem became even more apparent when searching for other implementations on GitHub, as those few implementations that we were able to find often did not adhere to basic coding standards and were therefore very difficult to comprehend.

Furthermore, we struggled with understanding and picking the ideal learning approaches to implement in our use case, for example separating backpropagation and biologically plausible training. Among the variety of approaches, it was challenging to identify the current state-of-the-art, as there doesn't appear to exist a standard yet.

Additionally, evaluating the results of our project has been a challenge due to the unexpected behavior of the SNN. We were unable to identify this as a reasoning error in our implementation or as a characteristic of STDP-based SNNs. We did, for example, not observe any significant changes in performance when adjusting important-thought hyperparameters, such as the number of training epochs, which could also be related to the robustness of the architecture or the presented dataset.

B. Limitations

One of the biggest limitations of our paper is the fact that we used the MNIST dataset for comparing SNNs and ANNs. While this is an established baseline dataset for ANNs, one could argue that it is also tailored especially towards ANNs, while SNNs are primarily good at event-based operations. Additionally, we only relied on a smaller subset of 80 number of images per class of the dataset, as training on 160 images per class or more, even after performance optimization such as better preprocessing, took at least 1 hour per epoch with a network size of 800. This makes our comparison and the used metrics of accuracy and performance over epochs not very conclusive, even though we were able to show that SNN performance doesn't improve beyond 160 training images. Still, we would argue that such comparisons are important to formally establish strengths and weaknesses of both architectures.

Additionally, our own script suffers from high RAM usage, as the entire dataset is converted into spike trains at the beginning of training. Changing that into a generator setup could significantly improve the required RAM storage. Furthermore, this separate implementation of a dataset generator class could be used to make the script easily applicable to different datasets.

C. Outlook

For further research we suggest a focus on baseline research before specializing in a niche of the field. This groundwork could then be the foundation for more specialized approaches and enable other researchers to gain a basic understanding of the topic before diving into detail. When training our network we experienced high training times and computational cost. In order to efficiently use and work with SNNs, libraries that optimize the training process are required.

Additionally, to make further contributions based on our work, future papers could try to adapt our script to different datasets and also train the SNN on the entire MNIST dataset to check whether this has a significant impact on performance beyond our measurements.

VI. Conclusion

All things combined, we achieved our central research goals, which we laid out in the introduction (I). We created our own publicly available basic implementation of an SNN while following common coding conventions for coding style and documentation. Additionally, we managed to implement and train both ANNs and SNNs on the MNIST dataset after successfully preprocessing the contained image data and compare the acquired results.

Existing work often differed from ours due to varying approaches towards SNNs, e.g., different training methods. We see a need for a commonly used baseline, which the field does not yet seem to have settled on. To this goal we contribute by providing a first baseline comparison, an open source basic implementation and some more insight into potential shortfalls.

From the gathered data, we concluded that STDP-based SNNs have a lower accuracy score than ANNs in classification tasks based on the MNIST dataset. Nevertheless, we came to the conclusion that the results for both network architectures are not properly comparable. Since the two networks are by design suitable for different kinds of data and even more important, we used different sizes of this dataset, their performance on the same task is not necessarily meaningful. Still, building upon our paper and repository, the relevance of such comparisons could be improved by training the SNN on different and complete datasets.

VII. Data availability statement

We trained our neural network on the publicly available MNIST dataset which is available under http://yann.lecun.com/exdb/mnist/ [13]. The implementation of our SNN and all other referenced code is available under https://github.com/cowolff/Simple-Spiking-Neural-Network-STDP. Finally, the SNNs trained by us can be tested by using the SNN.py script available in the Github repository.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] L F Abbott. Lapicque's introduction of the integrate-and-fire model neuron (1907), 1999.

[3] Sander Bohte, Joost Kok, and Johannes Poutré. Spike-prop: backpropagation for networks of spiking neurons. volume 48, pages 419–424, 7 2000.

[4] A. N. Burkitt. A review of the integrate-and-fire neuron model: I. homogeneous synaptic input. *Biological Cybernetics*, 95:1–19, 7 2006.

[5] Natalia Caporale and Yang Dan. Spike timing-dependent plasticity: A hebbian learning rule, 2008.

[6] Rahul Chauhan, Kamal Kumar Ghanshala, and R. C. Joshi. Convolutional neural network (cnn) for image detection and recognition. pages 278–282. Institute of Electrical and Electronics Engineers Inc., 7 2018.

[7] Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke. The heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[8] Simon Davidson and Steve B. Furber. Comparison of artificial and spiking neural networks on digital hardware. *Frontiers in Neuroscience*, 15, 4 2021.

[9] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29:141–142, 2012.

[10] Jason K. Eshraghian, Max Ward, Emre Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D. Lu. Training spiking neural networks using lessons from deep learning. 9 2021.

[11] Hananel Hazan, Daniel J. Saunders, Hassaan Khan, Devdhar Patel, Darpan T. Sanghavi, Hava T. Siegelmann, and Robert Kozma. Bindsnet: A machine learning-oriented spiking neural networks library in python. *Frontiers in Neuroinformatics*, 12, 12 2018.

[12] Nikola Kasabov. *Time-Space, Spiking Neural Networks and Brain-Inspired Artificial Intelligence*. Springer Berlin, 2019.

[13] Yann LeCun, Corinna Cortes, and Chris Burges. Mnist handwritten digit database.

[14] Chankyu Lee, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. *Frontiers in Neuroscience*, 12, 2018.

[15] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10:1659–1671, 1997.

[16] Sujay Pandit. sujay-pandit/spiking-neural-networks-mnist-classification: Pure python implementation of unsupervised mnist classification using spiking neural networks (using stdp), 2022.

[17] Kinjal Patel, Eric Hunsberger, Sean Batir, and Chris Eliasmith. A spiking neural network for image segmentation. 6 2021.

[18] Michael Pfeiffer and Thomas Pfeil. Deep learning with spiking neurons: Opportunities and challenges. *Frontiers in Neuroscience*, 12, 2018.

[19] Gopalakrishnan Srinivasan, Priyadarshini Panda, and Kaushik Roy. Stdp-based unsupervised feature learning using convolution-over-time in spiking neural networks for energy-efficient neuromorphic computing. *J. Emerg. Technol. Comput. Syst.*, 14(4), nov 2018.

[20] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural Networks*, 111:47–63, 2019.