Training Spiking Neural Networks with Synaptic Plasticity under Integer Representation

Shruti R. Kulkarni, Maryam Parsa, J. Parker Mitchell, Catherine D. Schuman
Computer Science and Math Division
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

ABSTRACT

Neuromorphic computing is emerging as a promising Beyond Moore computing paradigm that employs event-triggered computation and non-von Neumann hardware. Spike Timing Dependent Plasticity (STDP) is a well-known bio-inspired learning rule that relies on activities of locally connected neurons to adjust the weights of their respective synapses. In this work, we analyze a basic STDP rule and its sensitivity on the different hyperparameters for training spiking neural networks (SNNs) customized for a neuromorphic hardware implementation with integer weights. We compare the classification performance on four UCI datasets (iris, wine, breast cancer and digits) that depict varying levels of complexity. We perform a search for optimal set of hyperparameters using both grid search and Bayesian optimization. Through the use of Bayesian optimization, we show the general trends in hyperparameter sensitivity in SNN classification problem. With the best sets of hyperparameters, we achieve accuracies comparable to some of the best performing SNNs on these four datasets. With a highly optimized STDP rule we show that these accuracies can be achieved with just 20 epochs of training.

CCS CONCEPTS

• **Hardware** → **Neural systems**; • **Computing methodologies** → *Supervised learning*.

KEYWORDS

Neuromorphic computing, spiking neural networks, synaptic plasticity, classification, hyperparameter optimization, spike timing dependent plasticity

ICONS '21, July 27–29, 2021, Oak Ridge, TN

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

ACM Reference Format:
Shruti R. Kulkarni, Maryam Parsa, J. Parker Mitchell, Catherine D. Schuman. 2021. Training Spiking Neural Networks with Synaptic Plasticity under Integer Representation. In *ICONS '21: International Conference on Neuromorphic Systems, July 27–29, 2021, Oak Ridge, TN*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Neuromorphic computing is a brain-inspired computing paradigm that has the potential to develop energy efficient edge and high performance computing (HPC) systems that perform intelligent operations. Neuromorphic systems can carry out a learning process based on local activity information. This ensures that synaptic strengths between neurons are adjusted without waiting for data from distant memory modules. Most of the emerging neuromorphic hardware and neural network accelerators make use of this principle in their designs [16, 23]. There have been several plasticity learning rules, including Hebb's rule, spike-timing dependent plasticity (STDP), etc., that have shown close to the machine learning state-of-the-art results on several machine learning benchmark datasets [5, 27].

There have been a wide variety of local learning rules proposed in the literature, including several variants of STDP (see Section 2). It is not clear *a priori* how the different learning rules will perform on a task, nor is it clear how the parameters of those learning rules affect performance. In this work, we investigate the performance of a basic STDP learning rule on four different datasets. We perform both grid search and Bayesian optimization to observe the effect of hyperparameters on performance of this learning rule and to determine the optimal hyperparameters for each dataset.

The remainder of the paper is organized as follows. Section 2 presents the prior works that have utilized bio-inspired local plasticity rules to train SNNs. We then introduce our framework and approach to train SNNs with STDP in Section 3. Section 4 presents the results on hyperparameter optimization and the classification accuracies on four UCI datasets. We further present a discussion on the convergence capability and insights from our experiments in Section 5. Finally, Section 6 summarizes the paper and presents the future outlook.

2 RELATED BACKGROUND

One of the earliest learning mechanisms based on neuro-biological studies is Hebbian learning, where the synaptic strength is adjusted based on the correlated activities in the connecting neurons [9]. An advancement over Hebb's rule is the spike timing dependent plasticity (STDP) rule, where the causality of pre- and post-synaptic neuron spike timings plays an important role in adjusting

the strength of the connecting synapse [3]. Typical operation of the STDP rule in SNNs follows an increment in the weight value when the post-synaptic neuron spike follows the pre-synaptic neuron spike temporally (also called potentiation), and decrement in the weight (depression) when the pre-synaptic neuron spike follows the post-synaptic neuron's spike temporally.

Various machine learning tasks such as image classification have been solved by making use of STDP rule to train SNNs [5, 6, 10, 15, 27]. One of the STDP rules presented by Shrestha, et al. shows approximations to the algorithm for hardware realization. They carry out the weight update as bit shift, to reduce the overall computation requirement and overcome the limited precision of the hardware [27]. Some of the other works use STDP as a feature extractor in training deep spiking neural networks, while employing error back-propagation to train the output classifier [12]

3 METHOD

The goal for this work is to investigate the effect of the hyperparameters of STDP learning rules on the supervised training performance on classification tasks. We focus specifically in the realm of neuromorphic hardware implementations in which the weights are integer representations, but we expect that this would translate to hardware implementations with low precision weight values as well. To enable our investigation of this topic, we utilize several existing software and hardware systems, each of which we will describe briefly in the sections below. We utilize the Caspian neuromorphic development platform as our neuromorphic hardware implementation with integer representations. We interface with the Caspian system through the TENNLab neuromorphic computing framework, which also performs input encoding from values in our dataset to spikes. In addition to a simple grid search on a small subset of potential hyperparameters for STDP, we also use a Bayesian hyperparameter optimization approach to investigate larger search spaces of potential hyperparameters. We summarize each of these components briefly in the following subsections. Finally, we explain in detail the precise STDP approach investigated here, including details about how it is implemented within the broader framework, as well as the hyperparameters we are investigating for that rule.

3.1 Caspian Neuromorphic Development Platform

Caspian is a neuromorphic development platform that includes both a software simulator component, as well as an FPGA-based neuromorphic hardware implementation [14]. In this work, we use the hardware-accurate software simulator for the Caspian neuromorphic hardware to run our experiments. The simulator is integrated along with the TENNLab framework. Caspian uses leaky integrate and fire spiking neuron model to realize the SNN, as well as synapses that include synaptic delays. All of the SNN computation is done with an integer representation. The Caspian hardware simulator allows for the ranges of values of thresholds, weights, and delays to be specified. In this case, we use the range of values for the weights and thresholds as a hyperparameter when training the network.

3.2 TENNLab Software Framework

The TENNLab neuromorphic software framework [22] implements a software interface between neuromorphic hardware implementations and applications. The interface includes input encoding and output decoding implementations that turn numerical data into spikes and take output spikes from the neuromorphic implementation and turn them into classification labels or control decisions. The TENNLab implementation also includes a variety of benchmark applications, including several control applications [21]; in this work, however, we focus on classification tasks. The TENNLab framework also supports a variety of backends, including the NEST and Brian2 simulators [11], though we focus on Caspian here. Through the TENNLab interface, users can load networks on neuromorphic hardware or onto a simulation of the hardware, apply spikes as input to the network, simulate the network activity, and extract spikes from the network.

3.3 Input Encoding into Spikes

A key ongoing research question in the field of SNNs and neuromorphic computing is the best way to encode numerical data as spikes. Though there are neuromorphic sensors such as event-based sensors [8], where the data is already in spike-form, most datasets are not natively spiking. There are several different approaches in the literature for encoding real-valued inputs into spikes. The TENNLab framework provides a variety of input encoding approaches, including rate, temporal, and population/binning encoding; it also allows for these approaches to be combined hierarchically to form more complex encoding schemes [24]. In this work, we use rate encoding and population encoding/binning. We briefly describe each of those encoding approaches below.

- Rate encoding: Each value get encoded into a certain number of spikes from the range $[0, max\_spikes]$. Here max_spikes is a hyperparameter.
- Binning or population encoding: Each input value gets mapped across one or more neurons (bins), each of which is assigned a range of values which may overlap. The input value per bin is then mapped to the different neurons through an encoding function (e.g., flip-flop, triangle, etc., which are described in detail in [24]). The number of bins and the binning function are both hyperparameters of these approaches.

3.4 Bayesian Hyperparameter Optimization

Bayesian optimization is well-suited for optimization problems where the objective function is unknown and expensive to evaluate. We chose the Hierarchical Pseudo Agent-based Bayesian Optimization (HPABO) [18–20] approach to optimize the performance (test accuracy) of STDP learning rules on various classification tasks. HPABO is flexible, fast, easy to use, and accommodates step by step analysis. This technique is built upon leveraging current belief (prior distribution), observations (likelihood model), and estimating an updated belief (posterior distribution) based on them.

Single objective optimization using HPABO starts with estimating a Gaussian distribution for initial observations (test accuracy levels) for five sets of random hyperparameters. The search space (all possible combinations of hyperparameters) is then explored and exploited based on optimizing a surrogate model built upon

the likelihood model (observations) and prior distribution. The optimum point of the surrogate model shows the best next set of hyperparameters to explore in the next iteration. Each new observation is added to the likelihood model. The posterior distribution is updated at every iteration based on the updated prior distribution and likelihood model until the total number of Bayesian searches is reached or the surrogate model is converged to zero. For our experiments, we fixed the surrogate model to "expected improvement" acquisition function [2] and Gaussian distribution kernel to "Matern" function. For further details please refer to [19, 20].

3.5 Spike Timing Dependent Plasticity

Several neuro-biological studies have demonstrated the existence of correlation between the timing and causality of spikes arriving on a synapse with the rise in potential being observed in the post-synaptic neuron [3]. This primary mechanism, called spike timing dependent plasticity (STDP), has been abstracted mathematically to adjust the synaptic conductance (or strength) based on a function of the relative timings of pre- and post-synaptic neuron spikes [4]. Here, we use the most common form of STDP where the weight change is exponentially dependent on the difference between the pre and post neuron spike times ($\Delta t = t_{post} - t_{pre}$), [4–6]. The mathematical representation of this rule to update a given synaptic weight is given as:

$$\Delta w = \begin{cases} A_+ \exp\left(\frac{-\Delta t}{\tau_+}\right), & \text{if } \Delta t \geq 0 \\ A_- \exp\left(\frac{\Delta t}{\tau_-}\right), & \text{if } \Delta t < 0. \end{cases} \quad (1)$$

Here, A_+ and A_- are the weighting factors for weight potentiation and depression, respectively. The magnitude of weight change Δw, for either case decays exponentially, with time constants τ_+ and τ_-, as the relative timing difference between the pre and post synaptic spikes grows. The weighting factors and time constants are the hyperparameters while training SNNs with this STDP rule.

3.6 Network Training

We make use of a feed-forward structured SNN, with the number of inputs decided by the spike encoding scheme and the number of outputs representing each class for a given dataset. We then study the impact of hyperparameters on the STDP training behavior, and discuss the convergence trend of our modified STDP rule in the following sections. Our current study is focused on analyzing the behaviour for a two-layered shallow SNN.

The weights of the network are updated the end of every epoch, after the network has been simulated for a certain number of timesteps for all of the training samples. However, we calculate the incremental weight update Δw after simulating the SNN for each sample based on the rule in Equation 1. The Δw computation for each synapse in the network depends solely on the timing difference Δt, which is calculated from spike raster obtained at the end of simulation time period. Similar to the approach by Shrestha, et al. [27], we also introduce supervision to train the network to correctly identify the class of the input. For every label neuron in the output layer that is not issuing spikes, the corresponding synapses receiving pre-synaptic spikes are strongly potentiated, by setting $\Delta t = +1$. Once the correct output starts issuing spikes, the amount of potentiation is reduced by setting Δt to the length of the STDP window, W. Similarly, in scenarios where an incorrect output neuron issues spikes, it is strongly depressed for every synapse which is receiving a pre-synaptic spike, by setting $\Delta t = -1$. For all the synapses not receiving pre-synaptic spikes, they are weakly depressed by setting $\Delta t = -W$. As demonstrated by Mozafari, et al., we also make use of reward based Δw accumulation to avoid over-fitting [15]. The Δw for each synapse is accumulated for every training sample i, rounded to integer representation, and finally the weight at the end of an epoch is adjusted as:

$$w(n+1) = w(n) + \text{round}\left(\eta . \sum_i \Delta w_i\right) \quad (2)$$

Here, η is the learning rate, also a hyperparameter. The output of the network is decided based on the winner-take-all rule, i.e., the output layer neuron that spikes the most represents the class of the input [13].

Table 1: Network, STDP and input encoding hyperparameters.

STDP	Network	Input encoding
τ_+	Leak	Encoder type
τ_-	Initial weights	Binning function
A_+	Simulation time	Max_spikes
A_-	No. of hidden neurons	Encoding interval
Learning rate	Synaptic delay	Overlap interval
Lr_decay	Threshold	No. of bins
Window		

4 RESULTS

We now present our results on four of the UCI datasets: iris, wine, breast cancer and digits [1]. We focus on these simple datasets in this work so that we can run large-scale searches in a reasonable amount of time, but in future work, we will extend to more complicated tasks. The networks trained for each of these datasets have just the input and output layers. We study the sensitivity of our STDP training approach on the combinations of different hyperparameters. The STDP training approach discussed in the previous section adjusts only the synaptic weights to get the maximum classification accuracy on each of these datasets. There are also a number of hyperparameters associated with input encoding, network initialization and the STDP rule itself that need to be set prior to running the training SNN (see Table 1). In the following subsection, we discuss our results on hyperparameter optimization.

4.1 Hyperparameter Optimization

As seen in Table 1, there are a total of 19 hyperparameters we need to initialize before training. However, based on our preliminary hyperparameter sensitivity studies we fixed the values for terms such as threshold, delay, encoding and overlap intervals, to be the same for all the four datasets. Also, the number of hidden neurons is currently kept to 0, which in future studies will be expanded to train multi-layer networks. The remaining hyperparameters varied across the four datasets, motivating our study on optimizing their

Table 2: Network, STDP and input encoding hyperparameters and their range of values giving a total of 16,128 combinations.

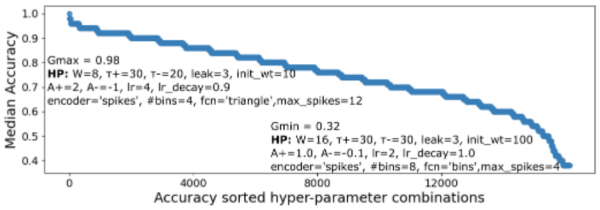
STDP		Network		Input encoding	
τ_+	{20, 30}	Leak	{3, 4}	Encoders types	{'rate', 'spikes'}
τ_-	{20, 30}	Init_wts	{10, 100}	Bins	{2, 4, 8}
A_+	{1, 2}			Max spikes	{4, 8, 12}
A_-	-{0.1, 1}				
Learning rate	{2, 4}				
Lr decay	{0.9, 1}				
Window	{2, 8, 16}				

values for each problem. Our search space has 16,128 combinations of hyperparameter values which are listed in Table 2. We applied both grid search and Bayesian optimization to arrive at the optimal hyperparameter values from this table.

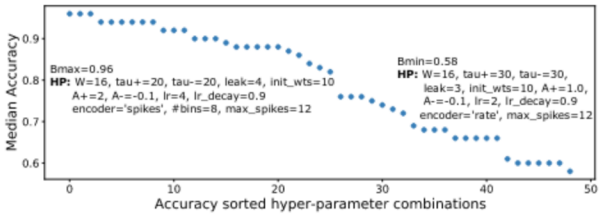
For the Iris dataset, the smallest dataset among the four, we first performed grid search to determine the test accuracy level for all possible 16,128 combinations of hyperparameters given in Table 2. Then, we performed Bayesian optimization search to derive the optimal set of hyperparameters only with 50 different runs. Figure 1a shows the prominent impact hyperparameter combinations have on test accuracy of the trained network. For the grid search experiment, the network is trained with all 16,128 combinations of hyperparameters, and the median test accuracies are plotted in descending order as seen in Figure 1a.

Figure 1b shows that Bayesian search is able to predict close to the maximum performance in terms of test accuracy with only 50 iterations. Figure 1c demonstrates the box plots for test accuracies with Bayesian optimization on iris dataset as the search iterations progress. Note that both the Figures 1b and 1c show the same data, with the former one being sorted based on the median test accuracies and the latter showing the results as they were collected throughout the Bayesian optimization. This shows although the technique starts from random performances in the beginning, toward the end of the Bayesian search, we already know the region (sets of hyperparameters) that lead to maximum performance (exploitation), and we still search for areas for possible better performance (exploration).

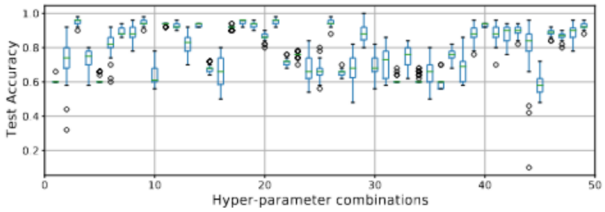
We further carried out the search to find the set of hyperparameters leading to optimum performance (in terms of test accuracy) for the remaining three datasets with Bayesian optimization. Figure 2 presents the box plots for test accuracies with their respective hyperparameter combinations sorted in descending order. It can be seen that as we progress from iris, which is the simplest in terms of number of inputs and outputs (4 and 3, respectively), to the more complex digits dataset, which has 64 inputs and 10 outputs, the correct choice of hyperparameter initialization highly impacts the STDP training.



(a) Sorted median accuracy plot for the 16128 hyperparameter combinations based on grid search.



(b) Sorted median accuracy plot for the hyperparameters explored by the Bayesian optimization process.

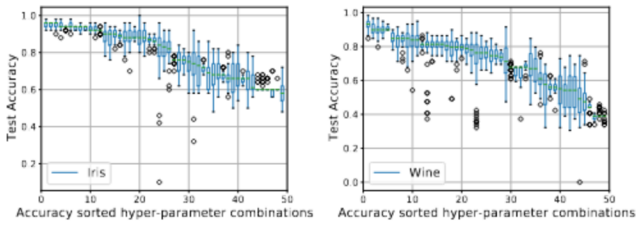


(c) Bayesian optimization carried out for 50 iterations.

Figure 1: Variation of the classification accuracy for the iris dataset as different hyperparameters are selected and evaluated with both grid search and Bayesian optimization. Each box plot here represents the results from 50 repeats of training simulation for a given set of hyperparameters.

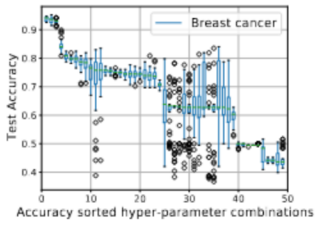
We also present the impact of individual hyperparameters on the classification performance in Figures 3, 5 and 4. In the input encoding hyperparameters, the 'spikes' encoding scheme is the preferred choice. However, as seen in Figure 3a, the box plot for digits has a higher tail end accuracy at 80% with 'rate' encoding scheme, even though the median is higher for 'spikes' encoding scheme. This indicates that the 'rate' encoding scheme along with other combinations of hyperparameters results in the best performance for digits. Note that the 'rate' encoding scheme does not use binning in the input layer. For the cases that use the 'spikes' encoding scheme, the optimal choice of binning function is 'triangle', and a higher number of input bins and input spike rate is preferred across all datasets.

Figure 4 shows the boxplots for training sensitivity with respect to the STDP parameters. For each of these hyperparameters, the search space was restricted to the best possible discrete values based on our initial simulations. For the STDP weighting factors, a higher magnitude of excitatory value, A_+ and lower magnitude of inhibitory value A_- is preferred across all datasets (see Figure 4a and 4b). For both the time constant of the learning window τ_+,

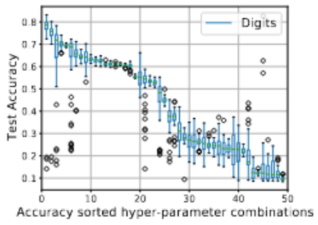


(a) Iris

(b) Wine

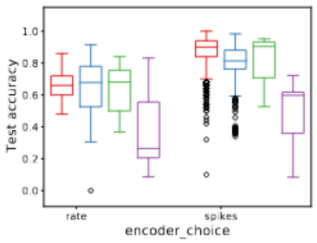


(c) Breast Cancer

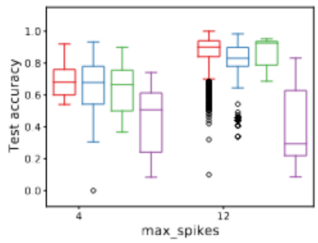


(d) Digits

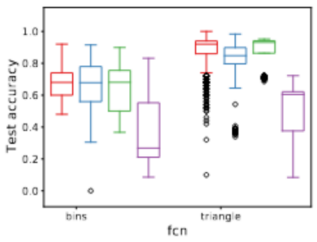
Figure 2: Test accuracy across different datasets, as the hyperparameters are varied by running Bayesian optimization for 50 iterations. Each iteration bar on the plots represents 50 repeats of training simulation. The hyperparameter sets are ordered based on the descending test accuracy.



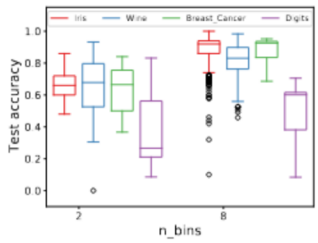
(a) Type of encoder



(b) Max input spikes



(c) Binning function

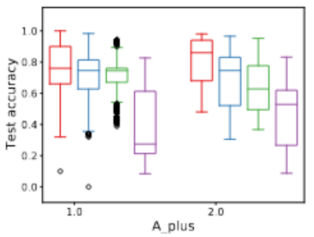


(d) Number of input bins

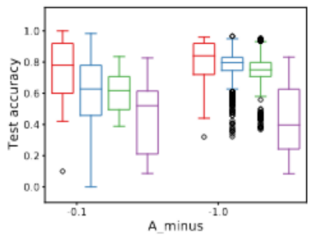
Figure 3: Sensitivity of test accuracy to input encoding parameters.

and the length of the STDP window W, a higher value among their respective options is a preferred choice (see Figure 4c and Figure 4d).

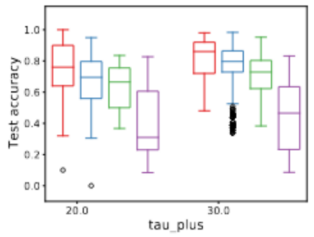
All of the SNN parameters such as neuron threshold, leak, synaptic delay, and the dynamic range of weight values in Caspian use integer representation, whose range can be configured in the beginning. Note that all of these parameters have purely numerical



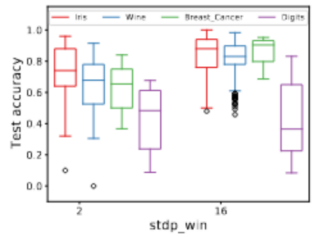
(a) Excitatory weighting factor



(b) Inhibitory weighting factor



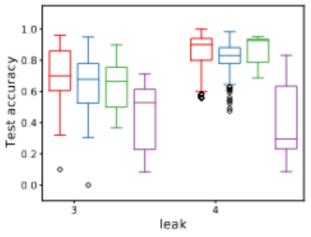
(c) Time constant of STDP learning window.



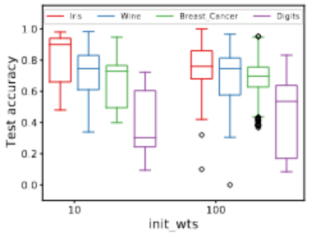
(d) Timing window

Figure 4: Sensitivity of test accuracy to STDP hyperparameters.

representation without any units. In all of our experiments we set the threshold of the spiking neurons to be 255 and the synaptic delay to be 1, which was found to be most suitable for all four datasets used. We included the initial weights and neuron leak in our list of hyperparameters to be optimized. Figure 5 shows the training sensitivity to network initialization. For the neuron leak, a higher value is always preferred (as seen in Figure 5a). We randomly initialize the network weights with integer values distributed uniformly between $[-x, +x]$, with 'x' being a hyperparameter shown in Figure 5b. A small range of values for weights' initialization was sufficient for the simpler datasets - iris, wine and breast cancer. However, we observed that digits required a higher initialization to achieve accuracies above 70%.



(a) Neuron leak



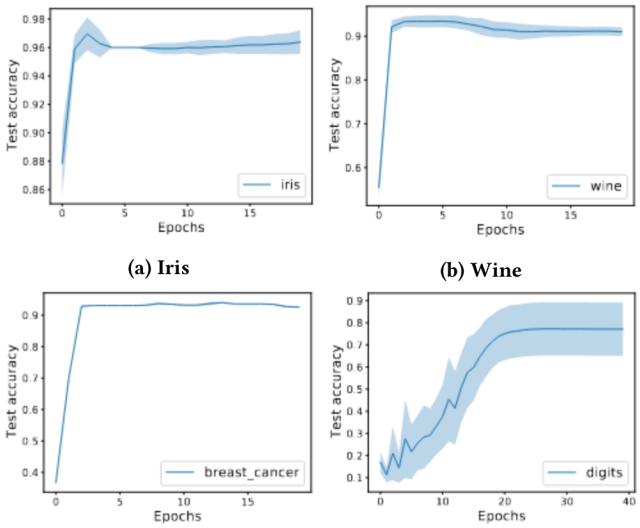
(b) Range for initial weights

Figure 5: Sensitivity of test accuracy to network initialization parameters - spiking neuron leak and range of values for initial weights.

4.2 Classification Performance

Using the best hyperparameters obtained from Bayesian optimization, we train the networks for each dataset with 100 different

initializations. Figure 6 shows the evolution of the training across multiple epochs. It can be observed that the highly optimized STDP training approach with integer representation progresses well starting from a small accuracy to over 80% across all the datasets. It is worth noting that for iris, wine, and breast cancer we observed the training to converge within 20 epochs. However, for digits we had to increase the number of training epochs to 40 for the STDP training to converge.



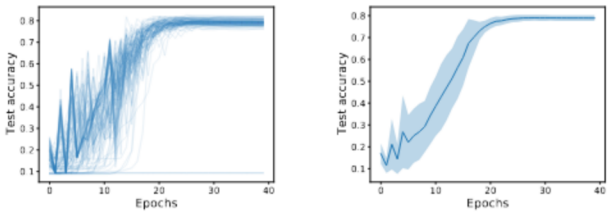
(a) Iris (b) Wine

(c) Breast Cancer (d) Digits

Figure 6: Test accuracy as training progresses through multiple epochs. For iris, wine and breast cancer we observed the training converge within 20 epochs. However, for digits, the convergence was seen after 20 epochs, hence, we trained it for 40 epochs.

It can be seen in Figure 6d, compared to other datasets, training curve for digits has a higher variation. One difference in the case of digits is the higher range of values used in initialization of the synaptic weights. This resulted cases where the resulting weight updates (Δw) became zero after the first 2 or 3 training epochs and did not improve as the epochs progressed. Figure 7a shows the actual line plots for test accuracy while training the digits dataset. While majority of the cases converged to accuracies above 70%, there were 3 cases out of the 100 runs which failed to train.

To mitigate the training failure, we added small random noise to the accumulated weight update term '$\sum \Delta w$' (refer to equation 2), every epoch. The random noise added per synaptic weight are integers chosen randomly in $[-y, y]$, where y was set at 10% of the weight initialization range. With this modification, the failure cases were altogether avoided. Figure 7b shows the results from 100 runs of STDP training for the digits with the additive integer noise in the weight update process. As the other datasets had a smaller range of weight initialization, we did not encounter such similar scenarios. Several works on ANNs have studied the impact of additive noise to improve learning and generalizability [17]. As a future work we would also be exploring in detail the impact of noise on the training behavior for a larger class of problems, and other causes that prevent SNNs from training.



(a) Digits test accuracy per epoch without additive noise while training.

(b) Digits mean test accuracy and variation per epoch with additive noise while training.

Figure 7: Training performance and the impact of adding noise. The training plots in (a) correspond to the same accuracy data shown in Figure 6d, which have been re-plotted as lines to indicate that the larger accuracy variation is caused by a few cases (at the bottom) with classification accuracy remaining at $\sim$ 10%. In (b) we can see the full convergence with very small variation starting after around 20 epochs of training.

5 DISCUSSION

Table 3 compares the mean test accuracy of the four datasets with some of the other spike based training approaches previously explored [25]. It can be seen that the mean accuracy for the relatively simpler datasets with our STDP approach are competitive with the other approaches. Our approach uses native spike-based local plasticity mechanism of STDP to train the SNN, unlike the other approaches such as decision tree or Whetstone method which were mapped from their respective non-spiking versions [26]. Our approach had a lower accuracy on the digits compared to the best reported that used mapping or evolutionary approaches [7, 25]. It is worth noting that our training approach resulted in improvements in accuracy with a simple two-layered SNN topology. In the future, we intend to extend this approach to more complex SNN topologies that can potentially benefit datasets with larger dimensionality. Moreover, the weight update process is local, which makes it highly suitable to realize it on neuromorphic and other edge platforms for real-time, on-chip training.

Table 3: Mean test accuracy across four datasets

Datasets	Supervised STDP (this work)	Other spike-based approaches [25]
Iris	97%	96% (Spike-based Decision Tree)
Wine	93%	90% (Reservoir)
Breast cancer	94%	91% (Spike-based Decision Tree)
Digits	79%	93% (Whetstone)

6 CONCLUSION AND FUTURE OUTLOOK

In this work we have proposed a training approach for SNNs with the basic STDP rule and tailored it for implementation on a neuromorphic hardware with integer representation. This approach also uses supervision to adjust the weights of output neurons. We also show the importance of the right choice of hyperparameters in our study using Bayesian Optimization and grid search techniques, while training SNNs with limited precision supervised STDP approach. Each of these hyperparameters greatly impacts the STDP training behavior based on the problem complexity and our current results can serve as a guide for future more complex problems.

With integer representation of the SNN's parameters and with the optimum selection of hyperparameters, our training approach shows good learning ability and convergence within the first 20 epochs of training. The use of small additive noise during the weight update helps improve training, especially for the cases stuck in local minima.

Going forward, there are several opportunities to study and apply the different local plasticity-based learning rules for SNNs to different real world problems. As discussed in earlier sections, the use of noise to improve SNN training and generalizability without incurring additional overheads is an interesting area we intend to explore. Further, the extension of our simple STDP style rules to train multi-layer or random topology of SNNs will be beneficial in realizing on-line training of networks on neuromorphic hardware for diverse sets of problems.

ACKNOWLEDGEMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under contract number DE-AC05-00OR22725.

This research used resources of the Compute and Data Environment for Science (CADES) at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

REFERENCES

[1] Arthur Asuncion and David Newman. 2007. UCI machine learning repository.
[2] James S Bergstra and et al. 2011. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*. 2546–2554.
[3] Guo-qiang Bi and Mu-ming Poo. 1998. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of neuroscience* 18, 24 (1998), 10464–10472.
[4] Natalia Caporale and Yang Dan. 2008. Spike timing–dependent plasticity: a Hebbian learning rule. *Annu. Rev. Neurosci.* 31 (2008), 25–46.
[5] Peter U Diehl and Matthew Cook. 2015. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience* 9 (2015), 99.
[6] Meng Dong, Xuhui Huang, and Bo Xu. 2018. Unsupervised speech recognition through spike-timing-dependent plasticity in a convolutional spiking neural network. *PloS one* 13, 11 (2018), e0204596.
[7] Daniel Elbrecht, Shruti R Kulkarni, Maryam Parsa, J Parker Mitchell, and Catherine D Schuman. 2020. Evolving Ensembles of Spiking Neural Networks for Neuromorphic Systems. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 1989–1994.
[8] Guillermo Gallego, Tobi Delbruck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew Davison, Jörg Conradt, Kostas Daniilidis, et al. 2019. Event-based vision: A survey. *arXiv preprint arXiv:1904.08405* (2019).
[9] Wulfram Gerstner and Werner M Kistler. 2002. Mathematical formulations of Hebbian learning. *Biological cybernetics* 87, 5 (2002), 404–415.
[10] Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, Simon J Thorpe, and Timothée Masquelier. 2018. STDP-based spiking deep convolutional neural networks for object recognition. *Neural Networks* 99 (2018), 56–67.
[11] Shruti R Kulkarni, Maryam Parsa, J Parker Mitchell, and Catherine D Schuman. 2021. Benchmarking the Performance of Neuromorphic and Spiking Neural Network Simulators. *Neurocomputing* (2021).
[12] Chankyu Lee, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. 2018. Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. *Frontiers in neuroscience* 12 (2018), 435.
[13] Wolfgang Maass. 2000. Neural computation with winner-take-all as the only nonlinear operation. *Advances in neural information processing systems* 12 (2000), 293–299.
[14] J Parker Mitchell, Catherine D Schuman, Robert M Patton, and Thomas E Potok. 2020. Caspian: A neuromorphic development platform. In *Proceedings of the Neuro-inspired Computational Elements Workshop*. 1–6.
[15] Milad Mozafari, Saeed Reza Kheradpisheh, Timothée Masquelier, Abbas Nowzari-Dalini, and Mohammad Ganjtabesh. 2018. First-spike-based visual categorization using reward-modulated STDP. *IEEE transactions on neural networks and learning systems* 29, 12 (2018), 6178–6190.
[16] SR Nandakumar, Shruti R Kulkarni, Anakha V Babu, and Bipin Rajendran. 2018. Building brain-inspired computing systems: Examining the role of nanoscale devices. *IEEE Nanotechnology Magazine* 12, 3 (2018), 19–35.
[17] Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. 2015. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807* (2015).
[18] Maryam Parsa, Aayush Ankit, Amirkoushyar Ziabari, and Kaushik Roy. 2019. Pabo: Pseudo agent-based multi-objective bayesian hyperparameter optimization for efficient neural accelerator design. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.
[19] Maryam Parsa, J Parker Mitchell, Catherine D Schuman, Robert M Patton, Thomas E Potok, and Kaushik Roy. 2019. Bayesian-based hyperparameter optimization for spiking neuromorphic systems. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 4472–4478.
[20] Maryam Parsa, John P Mitchell, Catherine D Schuman, Robert M Patton, Thomas E Potok, and Kaushik Roy. 2020. Bayesian multi-objective hyperparameter optimization for accurate, fast, and efficient neural network accelerator design. *Frontiers in neuroscience* 14 (2020), 667.
[21] James Plank, Charles Rizzo, Kirolos Shahat, Grant Bruer, Trevor Dixon, Michael Goin, Grace Zhao, Jeremy Anantharaj, Catherine Schuman, Mark Dean, et al. 2019. *The TENNLab suite of LIDAR-based control applications for recurrent, spiking, neuromorphic systems.* Technical Report. Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States).
[22] James S Plank, Catherine D Schuman, Grant Bruer, Mark E Dean, and Garrett S Rose. 2018. The TENNLab exploratory neuromorphic computing framework. *IEEE Letters of the Computer Society* 1, 2 (2018), 17–20.
[23] M Prezioso, MR Mahmoodi, F Merrikh Bayat, H Nili, H Kim, A Vincent, and DB Strukov. 2018. Spike-timing-dependent plasticity learning of coincidence detection with passively integrated memristive circuits. *Nature communications* 9, 1 (2018), 1–8.
[24] Catherine D Schuman, James S Plank, Grant Bruer, and Jeremy Anantharaj. 2019. Non-traditional input encoding schemes for spiking neuromorphic systems. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–10.
[25] Catherine D. Schuman, James S. Plank, Maryam Parsa, Shruti R. Kulkarni, Nicholas Skula, and J. Parker Mitchell. [n.d.]. Software Framework for Comparing Training Approaches for Spiking Neuromorphic Systems. ([n. d.]).
[26] William Severa, Craig M Vineyard, Ryan Dellana, Stephen J Verzi, and James B Aimone. 2019. Training deep neural networks for binary communication with the whetstone method. *Nature Machine Intelligence* 1, 2 (2019), 86–94.
[27] Amar Shrestha, Khadeer Ahmed, Yanzhi Wang, and Qinru Qiu. 2017. Stable spike-timing dependent plasticity rule for multilayer unsupervised and supervised learning. In *2017 international joint conference on neural networks (IJCNN)*. IEEE, 1999–2006.