Automated Design of Neuromorphic Networks for Scientific Applications at the Edge

Catherine D. Schuman
Computational Data Analytics
Oak Ridge National Laboratory
Oak Ridge, TN, USA
schumancd@ornl.gov

J. Parker Mitchell
Computational Data Analytics
Oak Ridge National Laboratory
Oak Ridge, TN, USA
mitchelljp1@ornl.gov

Maryam Parsa
Purdue University
West Lafayette, IN, USA
mparsa@purdue.edu

James S. Plank
Department of EECS
University of Tennessee
Knoxville, TN, USA
jplank@utk.edu

Samuel D. Brown
Department of EECS
University of Tennessee
Knoxville, TN, USA
sbrow109@utk.edu

Garrett S. Rose
Department of EECS
University of Tennessee
Knoxville, TN, USA
garose@utk.edu

Robert M. Patton
Computational Data Analytics
Oak Ridge National Laboratory
Oak Ridge, TN, USA
pattonrm@ornl.gov

Thomas E. Potok
Computational Data Analytics
Oak Ridge National Laboratory
Oak Ridge, TN, USA
potokte@ornl.gov

***Abstract*—Designing spiking neural networks for neuromorphic deployment is a non-trivial task. It is further complicated when there are resource constraints for the neuromorphic implementation, such as size or power constraints, that may be present in edge applications. In this work, we utilize a previously presented approach, EONS, to design spiking neural networks for a memristive neuromorphic implementation for scientific data applications. We specifically use a multi-objective approach in EONS to maximize network accuracy on the scientific data application task, but also to minimize network size and energy. We illustrate that EONS determines both the network structure and the parameters, removing the burden from the user on determining the appropriate spiking neural network structure, and we show that the resulting networks are very different from the layered structure of typical neural networks. Finally, we show that the multi-objective approach produces smaller, more energy efficient networks than the original EONS approach and produces comparable accuracy to a back-propagation style training approach.**

I. Introduction

Neuromorphic computing systems offer the opportunity for energy efficient implementations for machine learning applications [1]. One of the key features for neuromorphic computing systems is that they are low power, energy efficient hardware implementations that can perform machine learning tasks such as data analysis. As such, there has been an increased interest in utilizing neuromorphic systems for applications at the edge in order to implement intelligent data processing in these resource-constrained environments.

However, an open question in the field of neuromorphic computing is how to best define the appropriate spiking neural network (SNN) for a given task and a given neuromorphic implementation. Though there are a variety of training approaches for SNNs, those approaches do not necessarily create SNNs that are directly amenable for deployment into SNN hardware. In particular, many training approaches for SNNs do not take into account the restricted weight resolution that is typical in neuromorphic implementations, especially those that are implemented using emerging devices such as memristors.

Moreover, many of the existing approaches for training neuromorphic systems produce very large SNNs. For example, back-propagation-like training approaches can easily result in SNNs that are on the order of at least hundreds of neurons and thousands to millions of synapses. For a resource-constrained environment, it may not be reasonable to expect that there is either the physical area available nor the power required in order to implement these types of SNNs on a neuromorphic system.

Finally, many of the existing training approaches require a pre-defined network structure to be determined before the algorithm can be used. For example, back-propagation-like algorithms may require the user to define the number of layers and number of neurons per layer or reservoir computing approaches may require the user to define the appropriate reservoir structure. It is non-trivial to determine the appropriate network structure for these tasks and often requires significant human effort to tune these algorithmic hyperparameters before they can be used effectively.

An approach called Evolutionary Optimization for Neuromorphic Systems (EONS) [2] has been previously introduced. EONS addresses some of these issues in that it tailors SNNs directly for the hardware implementation at hand and it designs both the structure and the parameters of the SNN, requiring less human effort in hand-tuning the hyperparameters of the system. EONS also tends to produce smaller networks than those required for back-propagation-like approaches or reservoir computing [3]. An approach that uses EONS to produce even smaller networks by explicitly adding minimizing network size as an objective to the optimization algorithm has previously been presented [4]. We extend that approach here by demonstrating the approach specifically for the automated design of small, energy efficient SNNs for a memristive neuromorphic system for scientific data classification tasks, with an edge deployment scenario specifically in mind. We show that the new approach produces networks that perform equivalently on accuracy, but are significantly smaller and significantly more energy efficient than the networks trained by the original EONS algorithm and networks trained using a back-propagation-based algorithm.

II. Background and Related Work

This work proposes an approach for designing neuromorphic networks for a particular neuromorphic hardware system, using evolutionary optimization. There are many algorithms for training SNNs, including those that use variations of back-propagation or gradient descent [5], [6], [7], [8]. A few approaches target designing for neuromorphic systems specifically, by taking into account constraints such as reduced weight precision [9], [10]. However, as noted above, these approaches can result in very large networks and do not necessarily take into account all of the behaviors of a neuromorphic system based on emerging devices.

For more emerging technologies such as memristors, there are other approaches to deal with training for neural networks that will be deployed onto those types of devices. For example, some issues have been previously addressed by implementing training on-the-chip [11]. Others use noise eliminating training processes [12], variation aware training [13], or training that incorporates the non-idealities of the devices into the training procedure [14]. Other challenges associated with implementing neural networks onto hardware, including maximum fan-in/fan-out of a neuron or limited precision, have been addressed by transforming and retraining under the hardware restrictions [15].

Finally, to help address issues associated with large networks requiring too large of a footprint or too much energy in the physical implementation, a variety of approaches have been tried, including pruning specifically for deployment onto a memristor crossbar array [16], [17], [18], or hyperparameter optimization for co-designing software and hardware [19]. Other approaches include weight pruning and quantization to reduce the size of the network [20]. These types of approaches are post-processing techniques that may have a detrimental effect on the accuracy of the network.

III. Method

In this work we use EONS to train SNNs for a memristive neuromorphic hardware implementation and that perform scientific data classification tasks. In the following subsections, we describe the hardware implementation and the datasets used in our experiments. We also briefly describe the EONS approach, as well the broader neuromorphic software framework in which it operates.

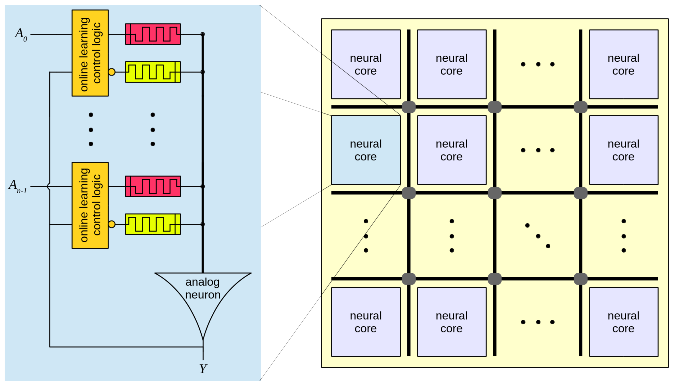

Fig. 1. MrDANNA structure, with neural cores made up of a single neuron and 8 twin-memristor synapses [21].

A. Hardware: MrDANNA

The hardware implementation used in this work is the memristive DANNA or mrDANNA system [21]. It utilizes an integrate-and-fire neuron model that accepts analog weighted current inputs and accumulates charge until an analog reference threshold voltage is achieved, at which point the neuron produces a digital output spike. Each synapse has two memristors connected with opposite polarity such that the effective conductance of the pair can represent positive and negative weights. The memristive devices used in this system are HfO$_x$, with high-resistance states (HRS) of 300 kΩ and low-resistance states (LRS) of 30 kΩ. The synapse delay in this system implementation is capped at a maximum of 7 cycles, and the neuron refractory period is fixed at 1 cycle. We assume a network frequency of 20 MHz. The overall mrDANNA system is made up of tiled neural cores, where each mrDANNA core has a single neuron and eight synapses. The organization of mrDANNA is shown in Figure 1.

To estimate the power consumption of this system, we performed circuit-level simulations of the individual components using 65 nm technology models. Energy estimates were collected for each component in all possible phases of operation. An overall energy estimate was constructed from these numbers based on each network's topology and the time each component spent in its separate operational phases. For the energy estimates used in this work, we use the neuron estimates from [21] and the synapse estimates from [22]. The energy estimates also assume a custom-fabricated circuit for each network.

B. Data

In this work, we use two datasets from the UCI Machine Learning repository [23], the MAGIC Gamma Telescope data set (MAGIC) and the Statlog Landsat Satellite (SAT) data set. Characteristics associated with each of those data are given in Table I. The MAGIC Gamma Telescope data is data simulated using Monte Carlo generation to simulate registration of

TABLE I
DATASET CHARACTERISTICS

	MAGIC	SAT
Training	4755	4435
Testing	14265	2000
Attributes	10	36
Classes	2	6

high energy gamma particles in a ground-based Cherenkov gamma telescope. The classification task for that dataset is to distinguish between gamma rays and hadron (background) rays. The second task is to, given a 3x3 pixel neighborhood of a satellite image, to predict the classification of the center pixel as one of red soil, cotton crop, grey soil, damp grey soil, soil with vegetation stubble, or very damp grey soil. We chose these datasets because they would both potentially benefit from a deployed, low-power neuromorphic solution, for example, at a telescope or on a satellite.

C. Software and Training

We use Evolutionary Optimization for Neuromorphic Systems (EONS) to define the SNNs for the mrDANNA hardware system to solve the two classification tasks described in the previous section. EONS is an evolutionary optimization based approach, described in [2]. To enable using EONS on this task for the mrDANNA hardware system, we utilize the TENNLab neuromorphic research framework [24]. This framework enables the use of EONS on mrDANNA through a common SNN definition and performs conversion on the data from the application into spikes to be included as input to the mrDANNA hardware. In this work, we use the binning and charge injection approaches for input encoding, as described in [25]. We use voting as the output decoding scheme, where the output neuron that fires the most corresponds to the classification. We use eight bins for each input value for each task. The eight bins correspond to different, non-overlapping subsets of the range of potential input values for each of the attributes. When encoding a particular attribute value as input to the network, the neuron that corresponds to the value that the attribute is in will spike a single time at the beginning of network simulation. Each network for the MAGIC task has 80 input neurons (8 bins for each of the 10 attributes) and 2 output neurons (one for each class) and each network for the SAT task has 288 input neurons (8 bins for each of the 36 attributes) and 6 output neurons (one for each class).

The EONS algorithm operates by first defining a population of potential SNNs to solve the given task on a given hardware platform. All of the SNNs in the population have the same number of inputs and outputs as described above, but also have some randomly initialized hidden neurons as well as randomly initialized synapses. The SNNs that make up the initial population are randomly generated. In order to evaluate the SNNs, we define a fitness function, which provides a score for how well the SNN is performing the task at hand on the hardware. Typically, as part of this fitness function evaluation, the given SNN is loaded onto either a simulator of the hardware or the hardware itself, the appropriate input spikes are applied, and the hardware or hardware simulator is run for some amount of time. The spikes of the output neurons are monitored and used to define how well the network performs on the task. In this case, the spiking outputs correspond to classification decisions, which are compared with the correct answers to produce an accuracy score. As noted below, the fitness function can be made more complicated as well, to encourage different types of behaviors.

Once all of the networks in the initial population have been evaluated and scores are assigned, we use a selection algorithm to select SNNs to serve as parents for the next generation based on their scores. For this work, EONS uses the selection algorithm tournament selection. This selection algorithm (like others in the genetic algorithm literature) does not always select the best performing networks in the population, but instead ensures that some diversity remains in the population of networks. EONS then uses the selected parents to produce children through the reproduction operations of duplication, crossover, and/or mutation. Crossover combines multiple parents into multiple children such that each of the children has some components from each of its parents. Mutation makes small changes (e.g., adding or deleting a neuron or synapse, changing a parameter value, etc.) to a given parent network to produce a child. Once the children networks have been created, they form the new population and the evaluation, selection, and reproduction cycle repeats either for a fixed number of generations or a time limit is reached. In this work, we fix the number of generations trained to 1000.

As noted above, to utilize EONS to define an SNN for a hardware platform, we must define a fitness function. Here, we use a multi-objective optimization based approach inspired by how EONS was utilized in [4]. In [4], the fitness function of EONS was expanded to include not only maximizing the network's accuracy on the task, but also minimizing network size and maximizing accuracy of perturbations of the network to increase resiliency. Moreover, that work demonstrated that multi-objective optimization approach out-performed a post-training pruning approach.

Here, we augment the typical accuracy-only EONS function to include minimizing network size and/or minimizing energy usage on the task, in addition to accuracy. In particular, we define the accuracy of the network on the task as $a(net) \in [0, 1]$, which is the fraction of correct classifications in the training set. We define the size of the network as the number of neurons in the network, $n_n(net)$, and the number of synapses in the network, $n_s(net)$. Finally, an energy estimate of how the network will perform is defined as $e(net)$.

Here, we evaluate how EONS will perform in designing mrDANNA networks for scientific applications at the edge by comparing the following four fitness functions. For brevity, we abbreviate accuracy as A, network size as S, and energy as E in the results in Section IV and in the equations below. The first fitness function just maximizes the accuracy of the network and does not consider network size or energy usage.

$$A = f_1(net) = a(net) \tag{1}$$

The second fitness function both encourages maximizing the accuracy of the network, but also minimizing the network size (in terms of the number of neurons and number of synapses).

$$AS = f_2(net) = a(net) - \alpha(n_n(net) + n_s(net)) \tag{2}$$

The α parameter scales down the penalty associated with the network size, and in this work is set to $\alpha = 10^{-7}$, though it is a hyperparameter for this approach. We include both the number of neurons and number of synapses in our calculation of network size and weight them the same way. However, it is worth noting that the actual impact of the physical implementations of neurons and synapses may be very different. As such, it may be appropriate to weight them differently in future work.

The third fitness function encourages maximizing the accuracy of the network and minimizing the energy required to evaluate the training instances.

$$AE = f_3(net) = a(net) - \alpha(e(net)) \tag{3}$$

Finally, the fourth fitness function encourages maximizing accuracy, minimizing network size, and minimizing the energy required.

$$ASE = f_4(net) = a(net) - \alpha(n_n(net) + n_s(net) + e(net)) \tag{4}$$

It is worth noting that though we produce a single score for each of these different approaches, this approach is equivalent to the weighted sum approach in genetic algorithm literature, which is a common way to perform multi-objective optimization in that field [26].

IV. Results

To understand the effect of different fitness functions on the performance of EONS, we ran 100 EONS tests for each fitness value, maintaining the same 100 random number seeds to generate the initial population across all four fitness functions. Thus, any variation in performance between the different fitness functions is due to the change in the fitness function and not due to the differences in the initial populations used. Figure 2 gives the results for these tests for the two datasets, including the accuracy on the testing set, the number of neurons in the resulting networks, the number of synapses in the resulting networks, and estimated energy required to run all of the testing examples, reported in uJ.

As can be seen in Figure 2, by including either network size or energy into the resulting networks, we can see a dramatic reduction in the number of neurons and synapses required to complete the task, as well as the amount of energy to complete the task, with relatively little, if any, impact on the accuracy of the network. As can be seen in the figures, when adding minimizing the number of neurons as part of the fitness

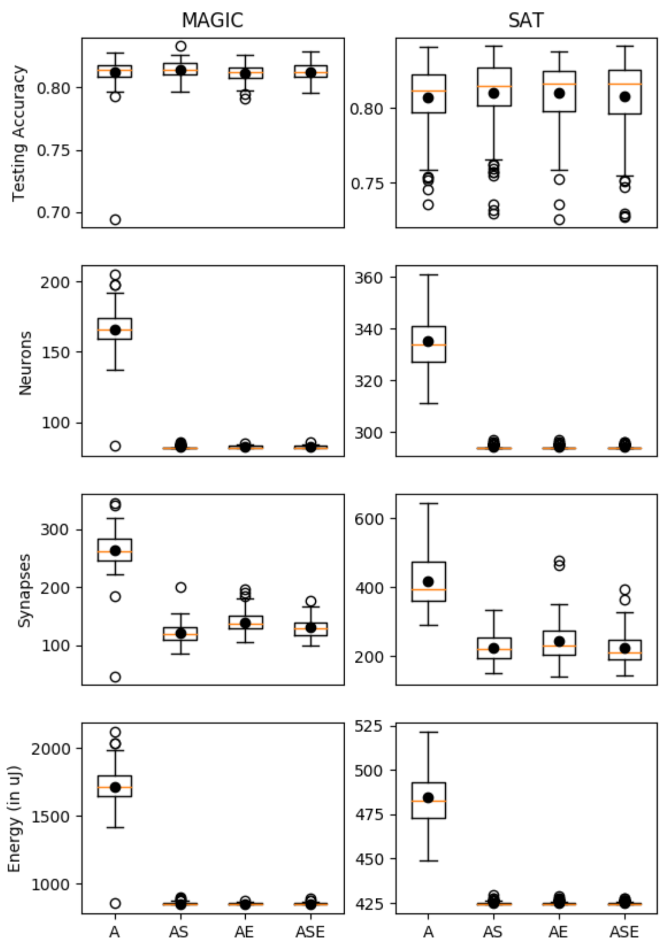

Fig. 2. Results for EONS for maximizing accuracy only as the objective (A), EONS for maximizing accuracy and minimizing network size as the objectives (AS), EONS for maximizing accuracy and minimizing energy as the objectives (AE), and EONS for maximizing accuracy, minimizing network size, and minimizing energy (ASE).

function or minimizing the energy required to perform the task as part of the fitness function, nearly all of the networks produced converge to the same number of neurons, which happens to be the number of input and output neurons in the network. In other words, by leveraging recurrent connections between input neurons, EONS is able to discover networks that solve these tasks with no hidden neurons for each of the two datasets. These sorts of networks cannot be achieved using back-propagation-style training algorithms because of their highly recurrent nature. To illustrate the non-traditional structure of these networks, the best performing EONS trained networks for MAGIC and SAT are shown in Figures 3 and 4. As can be seen in these figures, there are many connections between input neurons (shown in pink).

To understand how EONS performs as compared with a back-propagation-like approach, we compare the results of the best performing EONS network (out of the 100 tests

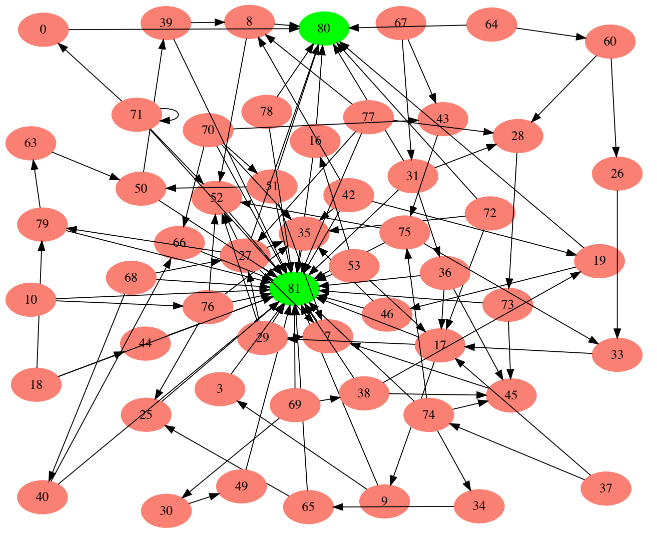

Fig. 3. Best performing EONS network for the MAGIC dataset, obtained from one of the AS runs. Input neurons are shown in pink and output neurons are shown in green.

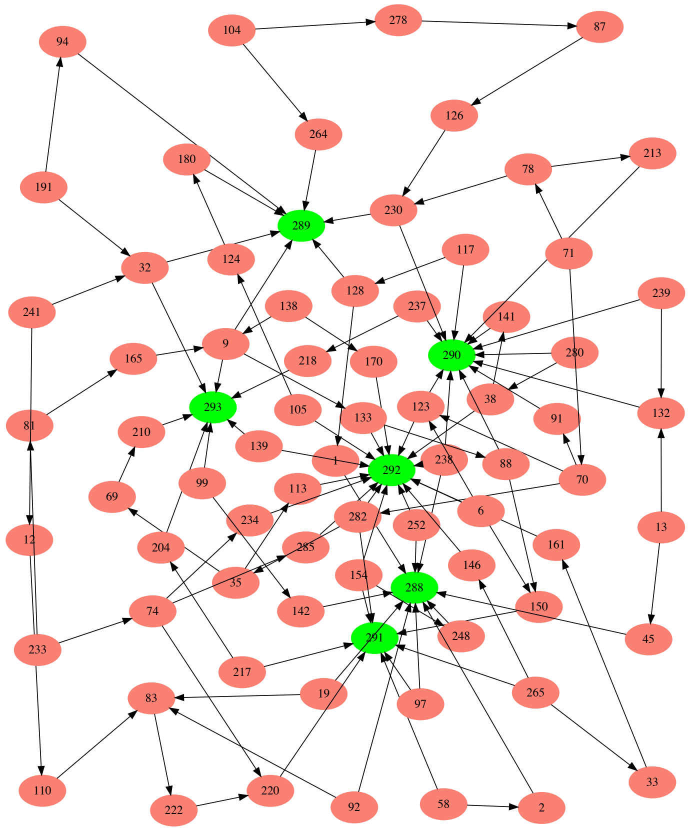

Fig. 4. Best performing EONS network for the SAT dataset, obtained from one of the AS runs. Input neurons are shown in pink and output neurons are shown in green.

TABLE II
COMPARISON OF EONS AND WHETSTONE ON MAGIC

Approach	Testing	Neurons	Synapses	Energy (in uJ)
EONS - A	82.8%	168	257	1766.7
EONS - AS	83.4%	82	128	846.2
EONS - AE	82.6%	82	120	847.2
EONS - ASE	82.9%	82	108	845.5
Whetstone	83.8%	130	3000	-

TABLE III
COMPARISON OF EONS AND WHETSTONE ON SAT

Approach	Testing	Neurons	Synapses	Energy (in uJ)
EONS - A	84.1%	343	381	495.3
EONS - AS	84.2%	294	180	424.6
EONS - AE	83.8%	294	271	424.5
EONS - ASE	84.2%	294	328	424.7
Whetstone	87.0%	196	9600	-

conducted) in Figure 2 for each type of fitness function with a Whetstone [6] trained network in Tables II and III, for the MAGIC and SAT datasets, respectively.

Whetstone is a back-propagation-like algorithm that sharpens the activation function of the neurons from a differentiable function to a non-differentiable binary activation over the course of training. Thus, the resulting networks produced by Whetstone use binary, spike-like communication that is amenable for SNN implementation and can be deployed to neuromorphic hardware. For Whetstone, we use 100 hidden neurons and 10-hot encoding for the outputs. Thus, for the MAGIC dataset, Whetstone has 10 input neurons, 100 hidden neurons, and 20 output neurons (10 for each output class), and for the SAT dataset, Whetstone has 36 input neurons, 100 hidden neurons, and 60 output neurons (6 for each output class). It is non-trivial to map a Whetstone network onto a neuromorphic implementation because of the input encoding and restricted weight resolutions in real neuromorphic systems, so we do not obtain an energy estimate for the Whetstone approach here for mrDANNA.

As can be seen in these tables, EONS achieves similar testing performance as Whetstone, but uses dramatically fewer synapses than the fully-connected networks of Whetstone. In particular, for the MAGIC dataset, Whetstone requires more than 20× as many synapses as the EONS approaches that include network size or energy as a metric. Similarly, for the SAT dataset, Whetstone requires 30-50× more synapses than the EONS approaches. The resulting Whetstone networks deployed on a neuromorphic system would be significantly larger in area and could have a substantial impact on the energy required.

It is worth noting that the performance and size of the networks produced using the AS, AE, and ASE networks are comparable, indicating that size and energy are highly correlated for this neuromorphic implementation and these applications and that in this case, it is sufficient to minimize network size or energy rather than both.

V. Discussion and Conclusion

In this work, we demonstrate that a training approach called EONS can be used to produce SNNs for scientific data tasks for a memristive neuromorphic system. This approach defines both the structure and parameter of the SNNs. It results in non-traditional neural network structures that can produce very small and quite sparse networks. We demonstrate that a multi-objective optimization approach with EONS can generate even smaller networks with lower energy consumption than the original, accuracy only optimization. EONS has been previously demonstrated on several different types of neuromorphic implementations, including digital [27], [28], biomimetic [29], and optoelectronic [30] neuromorphic systems. Moreover, it has been used for a variety of different types of applications, including another high energy physics data application [31], as well different types of control tasks [32]. The approach demonstrated in this work can be used to produce smaller and perhaps more energy efficient SNNs for these different implementations and applications.

The key contributions of this work are:

- A demonstration of a multi-objective optimization approach for designing neuromorphic SNNs for scientific data applications that would benefit from an edge deployment.
- An illustration of how this multi-objective approach can produce substantially smaller and more energy efficient networks for a memristive neuromorphic implementation.
- An illustration of how the training approach EONS produces non-traditional SNN structures for implementation on neuromorphic systems.

Acknowledgment

This material is based in part 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, in part by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U. S. Department of Energy., and in part by an Air Force Research Laboratory Information Directorate grant (FA8750-16-1-0065).

References

[1] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *arXiv preprint arXiv:1705.06963*, 2017.

[2] C. D. Schuman, J. S. Plank, A. Disney, and J. Reynolds, "An evolutionary optimization framework for neural networks and neuromorphic architectures," in *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2016, pp. 145–154.

[3] J. J. M. Reynolds, J. S. Plank, C. D. Schuman, G. Bruer, A. W. Disney, M. Dean, and G. S. Rose, "A comparison of neuromorphic classification tasks," in *International Conference on Neuromorphic Computing Systems*. Knoxville, TN: ACM, July 2018.

[4] M. Dimovska, J. T. Johnston, C. D. Schuman, J. P. Mitchell, and T. E. Potok, "Multi-objective optimization for size and resilience of spiking neural networks," in *2019 IEEE Annual Ubiquitous Computing, Electronics, and Mobile Communication Conference*. IEEE, 2019, p. In press.

[5] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in neuroscience*, vol. 10, p. 508, 2016.

[6] W. Severa, C. M. Vineyard, R. Dellana, S. J. Verzi, and J. B. Aimone, "Training deep neural networks for binary communication with the whetstone method," *Nature Machine Intelligence*, vol. 1, no. 2, p. 86, 2019.

[7] D. Rasmussen, "Nengodl: Combining deep learning and neuromorphic modelling methods," *Neuroinformatics*, pp. 1–18, 2019.

[8] S. Shrestha and G. Orchard, "Slayer: Spike layer error reassignment in time," in *Advances in Neural Information Processing Systems*, 2018, pp. 1412–1421.

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

[10] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in *Advances in Neural Information Processing Systems*, 2015, pp. 1117–1125.

[11] R. Hasan and T. M. Taha, "Enabling back propagation training of memristor crossbar neuromorphic processors," in *2014 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2014, pp. 21–28.

[12] B. Liu, M. Hu, H. Li, Z.-H. Mao, Y. Chen, T. Huang, and W. Zhang, "Digital-assisted noise-eliminating training for memristor crossbar-based analog neuromorphic computing engine," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2013, pp. 1–6.

[13] B. Liu, H. Li, Y. Chen, X. Li, Q. Wu, and T. Huang, "Vortex: variation-aware training for memristor x-bar," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 15.

[14] I. Chakraborty, D. Roy, and K. Roy, "Technology aware training in memristive neuromorphic systems for nonideal synaptic crossbars," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 5, pp. 335–344, 2018.

[15] Y. Ji, Y. Zhang, S. Li, P. Chi, C. Jiang, P. Qu, Y. Xie, and W. Chen, "Neutrams: Neural network transformation and co-design under neuromorphic hardware constraints," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 2016, p. 21.

[16] A. Ankit, A. Sengupta, and K. Roy, "Trannsformer: N eural n etwork transform ation for memristive crossbar based neuromorphic system design," in *Proceedings of the 36th International Conference on Computer-Aided Design*. IEEE Press, 2017, pp. 533–540.

[17] A. Ankit, T. Ibrayev, A. Sengupta, and K. Roy, "Trannsformer: Clustered pruning on crossbar-based architectures for energy efficient neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[18] L. Liang, L. Deng, Y. Zeng, X. Hu, Y. Ji, X. Ma, G. Li, and Y. Xie, "Crossbar-aware neural network pruning," *IEEE Access*, vol. 6, pp. 58 324–58 337, 2018.

[19] M. Parsa, A. Ankit, A. Ziabari, and K. Roy, "Pabo: Pseudo agent-based multi-objective bayesian hyperparameter optimization for efficient neural accelerator design," *arXiv preprint arXiv:1906.08167*, 2019.

[20] G. Yuan, X. Ma, C. Ding, S. Lin, T. Zhang, Z. S. Jalali, Y. Zhao, L. Jiang, S. Soundarajan, and Y. Wang, "An ultra-efficient memristor-based dnn framework with structured weight pruning and quantization using admm," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2019, pp. 1–6.

[21] G. Chakma, M. M. Adnan, A. R. Wyer, R. Weiss, C. D. Schuman, and G. S. Rose, "Memristive mixed-signal neuromorphic systems: Energy-efficient learning at the circuit-level," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 125–136, 2018.

[22] M. M. Adnan, S. Sayyaparaju, G. S. Rose, C. D. Schuman, B. W. Ku, and S. K. Lim, "A twin memristor synapse for spike timing dependent learning in neuromorphic systems," in *2018 31st IEEE International System-on-Chip Conference (SOCC)*. IEEE, 2018, pp. 37–42.

[23] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[24] J. S. Plank, C. D. Schuman, G. Bruer, M. E. Dean, and G. S. Rose, "The TENNLab exploratory neuromorphic computing framework," *IEEE Letters of the Computer Society*, vol. 1, no. 2, pp. 17–20, July-Dec 2018. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/LOCS.2018.2885976

[25] C. D. Schuman, J. S. Plank, G. Bruer, and J. Anantharaj, "Non-traditional input encoding schemes for spiking neuromorphic systems," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–10.

[26] V. Guliashki, H. Toshev, and C. Korsemov, "Survey of evolutionary algorithms used in multiobjective optimization," *Problems of engineering cybernetics and robotics*, vol. 60, no. 1, pp. 42–54, 2009.

[27] A. Disney, J. Reynolds, C. D. Schuman, A. Klibisz, A. Young, and J. S. Plank, "DANNA: A neuromorphic software ecosystem," *Biologically Inspired Cognitive Architectures*, vol. 9, pp. 49–56, July 2016.

[28] J. P. Mitchell, M. E. Dean, G. R. Bruer, J. S. Plank, and G. S. Rose, "Danna 2: Dynamic adaptive neural network arrays," in *Proceedings of the International Conference on Neuromorphic Systems*. ACM, 2018, p. 10.

[29] M. S. Hasan, C. D. Schuman, J. S. Najem, R. Weiss, N. D. Skuda, A. Belianinov, C. P. Collier, S. A. Sarles, and G. S. Rose, "Biomimetic, soft-material synapse for neuromorphic computing: From device to network," in *IEEE 13th Dallas Circuits and Systems Conference (DCAS)*, November 2018. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8620187

[30] S. Buckley, A. N. McCaughan, J. Chiles, R. P. Mirin, S. W. Nam, J. M. Shainline, G. Bruer, J. S. Plank, and C. D. Schuman, "Design of superconducting optoelectronic networks for neuromorphic computing," in *IEEE International Conference on Rebooting Computing*, Tysons, VA, November 2018, pp. 36–42.

[31] C. D. Schuman, T. E. Potok, S. Young, R. Patton, G. Perdue, G. Chakma, A. Wyer, and G. S. Rose, "Neuromorphic computing for temporal scientific data classification," in *Neuromorphic Computing Symposium*, ser. NCS '17. New York, NY, USA: ACM, 2017, pp. 2:1–2:6. [Online]. Available: http://doi.acm.org/10.1145/3183584.3183612

[32] J. S. Plank, C. Rizzo, K. Shahat, G. Bruer, T. Dixon, M. Goin, G. Zhao, J. Anantharaj, C. D. Schuman, M. E. Dean, G. S. Rose, N. C. Cady, and J. Van Nostrand, "The TENNLab suite of LIDAR-based control applications for recurrent, spiking, neuromorphic systems," in *44th Annual GOMACTech Conference*, Albuquerque, March 2019.