DANNA 2: Dynamic Adaptive Neural Network Arrays

J. Parker Mitchell
University of Tennessee
Knoxville, Tennessee
jmitch47@vols.utk.edu

Mark E. Dean
University of Tennessee
Knoxville, Tennessee
markdean@utk.edu

Grant R. Bruer
University of Tennessee
Knoxville, Tennessee
gbruer@vols.utk.edu

James S. Plank
University of Tennessee
Knoxville, Tennessee
jplank@utk.edu

Garrett S. Rose
University of Tennessee
Knoxville, Tennessee
garose@utk.edu

ABSTRACT

Following from the original Dynamic Adaptive Neural Network Array (DANNA) model, we propose a new digital neuromorphic architecture named DANNA 2. Through this paper, we introduce our new hardware design and software simulator, and we explain how DANNA2 can improve network density, training convergence, and element performance as compared to the DANNA model. We propose two network arrays types catering to FPGA and VLSI use cases. Using simulation results for both control and classification problems, we show improved training convergence and simulation performance.

CCS CONCEPTS

• **Computing methodologies** → *Artificial intelligence*; • **Computer systems organization** → *Parallel architectures*; • **Hardware** → *Neural systems*;

KEYWORDS

Neural Network, Neuromorphic, Reconfigurable Logic, Biologically Inspired Computation, Genetic Algorithms

ACM Reference Format:

J. Parker Mitchell, Mark E. Dean, Grant R. Bruer, James S. Plank, and Garrett S. Rose. 2018. DANNA 2: Dynamic Adaptive Neural Network Arrays. In *ICONS '18: International Conference on Neuromorphic Systems, July 23–26, 2018, Knoxville, TN, USA*. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3229884.3229894>

1 INTRODUCTION

There are significant concerns with the ability of traditional Von Neumann systems to continue to scale performance and power efficiency. These architectures have largely depended on increasing density and improving switching speeds of silicon transistors. Unfortunately, we are already seeing the difficulty in this approach.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](https://permissions.acm.org).

ICONS '18, July 23–26, 2018, Knoxville, TN, USA

© 2018 Association for Computing Machinery.

ACM ISBN 978-1-4503-6544-4/18/07...\$15.00

<https://doi.org/10.1145/3229884.3229894>

Coinciding with the difficulty of continued performance and efficiency scaling, machine learning tasks are becoming more popular. There exists an ever increasing need for systems which can learn from real world data, and many of these tasks may require significant computing resources. Just as importantly, there exists a desire for systems which can adapt and respond to unexpected scenarios in a seamless manner.

Neuromorphic computing aims to address these issues. By incorporating concepts from biology, we believe neuromorphic systems will be vastly more efficient while being adaptive to real world problems.

This work explores a new neuromorphic architecture which improves on previous work through increased implementation density, improved training convergence, and enhanced system performance.

2 BACKGROUND

One common machine learning approach is an artificial neural network (ANN). One variation of ANNs is a spiking neural network (SNN). SNNs are typically designed to mimic and thus have similar operating characteristics to biological neurons. As discussed by Schuman et al. in a recent survey of neuromorphic computing, spiking neural networks may have a wide variety of neuron models, but this paper is only exploring integrate and fire style models [13].

Neuroscience Inspired Dynamic Architectures (NIDA) is one implementation of a neuromorphic SNN. NIDA uses three dimensional networks of neurons connected with synapses. Neurons use an integrate and fire model, and synapses carry spikes between neurons with a given weight. Transmitting each fire results in a delay proportional to the length of the synapse. Both neuron thresholds and synaptic weights are expressed as floating point numbers to give a high degree of precision [10].

Following from NIDA, there was a desire and need to develop an architecture which may be implemented as a programmable hardware array on an FPGA or as a VLSI integrated circuit. This led to the creation of Dynamic Adaptive Neural Network Arrays (DANNA). DANNA uses a two dimensional $N \times N$ grid of elements with element connectivity constrained to a 16 nearest neighbor scheme [1]. Each DANNA element may be a neuron or a synapse. Neurons use an accumulate and fire model with 8-bit integer thresholds, and synapses use 8-bit integer weights and up to 256 cycles of configurable delay. Synapses also have long term potentiation and depression (LTP/LTD) to allow for online learning and adaptability. LTP allows for a synaptic weight to be increased if the synapse

fires immediately prior to a post-synaptic neuron firing. Similarly, LTD allows for a synaptic weight be decreased if the synapse fires immediately after a post-synaptic neuron firing [3]. DANNA is now a mature platform complete with an event based simulator [4], multiple FPGA deployments, and a VLSI implementation [2].

Spiking neural networks present a significant challenge to effectively train to solve a task. There are some results with modifications of backpropagation through time algorithms which attempt to deal with the nondifferentiable nature of the activation function [5], but these approaches are difficult and cumbersome to apply to all SNN models.

With NIDA and DANNA, the use of genetic algorithms offers one solution. We use a set of genetic algorithms which we call Evolutionary Optimization (EO). With EO, training occurs by initializing a set of random networks called a population. Each network in the population is evaluated with an application defined fitness function which specifies success for a task. Based on the fitness of each network, the networks can be ranked, and the best networks can be preferentially chosen through tournament selection. Then, a new population can be formed by applying mutations and crossover operations to prior networks. The evaluation of each population constitutes a training epoch. This training process is iterative and continues until either an epoch limit is reached or a network achieves the target fitness value [10].

3 THE MODEL

DANNA2 follows a simple spiking neural network model using neurons and synapses as the network primitives. The model is designed to work well as a digital hardware design or as an event based software simulation.

3.1 Neurons

Neurons follow a discretized accumulate and fire model in which each neuron accumulates charge until a defined activation threshold. Neurons acquire charge based upon incoming spikes and the synaptic weights associated with each spike. Once the activation threshold is reached, the neuron emits a spike to each of its downstream synapses which then will travel to other neurons. Immediately following a fire event, the neuron enters a refractory period in which it may not fire again for some configurable time.

DANNA2's neurons represent activation thresholds as 10 bit unsigned integers up to 1023, and refractory periods may be up to 7 network cycles after the neuron's last fire. DANNA2 also optionally supports linear charge leak in which the neuron's charge may decay back to a resting state (i.e. value of 0) at a programmable linear rate.

The neuron charge function used for DANNA [3] may be updated to take into account the availability of a linear leak, and thus, the function may be expressed as:

$$H_{kj}(t) = \sum_{i=1}^N w_i(t)x_i(t) + H_{kj}(t-1) - L_{kj}(t-1) \quad (1)$$

The neuron activation function is unchanged from DANNA and may be expressed as:

$$a_{kj}(t) = f(H_{kj}(t)) = \begin{cases} 1 & \text{if } H_{kj}(t) \geq \theta(t) \\ 0 & \text{if } H_{kj}(t) < \theta(t) \end{cases} \quad (2)$$

3.2 Synapses

Synapses serve as a mechanism to connect neurons. Each synapse transmits spikes from a pre-synaptic neuron to a post-synaptic neuron. These spikes are sent with a configurable weight value which corresponds with the amount of charge the post-synaptic neuron should accumulate. There is also a configurable delay between the synapse receiving a spike from the pre-synaptic neuron and when the post-synaptic neuron will receive the spike from the synapse.

DANNA2's synapses use signed 9-bit integer weight values from -256 to 255 and may have a delay up to 15 additional network cycles. Each neuron is allowed up to 24 incoming synapse connections, but this number may be increased through the use of charge forwarding with a fan-in element as described later in the paper. Synapses may also leverage a simple Spike Timing Dependent Plasticity (STDP) model in which weight values are dynamically adjusted during runtime as a method of online learning.

The DANNA2 STDP model is a discretized lookup table keyed on the time difference between a synapse fire and a post-synaptic neuron fire.

$$\Delta t = t_{\text{synapse_fire}} - t_{\text{neuron_fire}} \quad (3)$$

This time value is used to look up a configurable synaptic weight adjustment. Potentiation values are located from $\Delta t = -N + 1$ to $\Delta t = 0$ in the table, and depression values are located from $\Delta t = 1$ to $\Delta t = N$ where $2N$ is the temporal window size for STDP. When $\Delta t = 0$ for DANNA2, this means the synapse fire occurred the same cycle that the neuron hit its threshold.

The approach allows for different STDP curves to be implemented, and DANNA2 can be optionally configured to use the same value for each potentiation and depression slot which then works similar to DANNA's single cycle LTP/LTD mechanism.

3.3 Networks

A network is a collection of neurons and synapses which work together to accept input spikes and generate output spikes. DANNA2 networks are typically sparse directed graphs in which the graph nodes are neurons and the edges are synapses. Networks often may have recurrent cyclic patterns.

Each network has a maximum width and height dimension $M \times N$, and each neuron must uniquely exist at a single coordinate pair within this specified grid. DANNA2 typically allocates dimensions with a height to width ratio of 2 to 1.

3.4 Input/Output

Inputs to the network are fire events with scaled weight values from 0 to 255 corresponding the positive weight range of a single synapse. All of a network's inputs are located on the left side of the network. Outputs from the network are valueless spikes. All of a network's outputs are located on the right side of the network.

4 HARDWARE ARCHITECTURE

4.1 Element Design

A DANNA2 element is a tightly coupled collection of 24 synapses with one post-synaptic neuron. The digital implementation of this is shown in Figure 1. A collection of 24 distance registers collect

Figure 1: Depiction of a DANNA2 Element

input fires for each respective synapse. During each element cycle, three synapses are sampled by reading from the corresponding distance registers and loading the appropriate synapse parameters from the synapse table. The synapse units are responsible for sending the correct weight value to the accumulator if the synapse is firing, and it also handles the calculation of spike timing dependent plasticity weight adjustments. It is also important to note that one DANNA element may be either a neuron or a synapse while a single DANNA2 element is a neuron with up to 24 synapses.

4.1.1 Network Cycles. The hardware implementation of DANNA2 uses element cycles and network cycles. An element cycle is a single clock cycle in which the element performs a portion of work. Each element requires 10 total element cycles to process all 24 synapses, fully update the accumulator, and register if there is a fire. These 10 element cycles form a single network cycle. In contrast, DANNA's element accumulator clock is 32 times faster than the network cycle clock. We expect the effective network speed of DANNA2's FPGA implementation to be approximately 10 times faster than DANNA. Some of these improvements may be optionally reallocated to lower power consumption and improved efficiency. This improvement in execution speed is due to logic timing optimization as well as the reduction in the ratio of the element clock to the network clock.

4.1.2 Configurable Leak. DANNA2 neurons may have a linear leak value with per-neuron granularity. The leak value is applied similar to a negative weight as an input to the charge accumulator.

4.1.3 Spike Timing Dependent Plasticity. Synapses have a simplified discrete spiking timing dependent plasticity implementation. Weight adjustment values can be stored in a small look up table in which $t - N + 1 \dots t + N$ values can be stored for a STDP window of $2N$. These values are determined at hardware synthesis but are adjustable for exploration of different weight update curves.

4.1.4 Fan-in Elements. An element can operate in a fan-in mode which allows for the 24 synapses to be forwarded to another neuron at the cost of one additional cycle of delay. This allows for a neuron with high connectivity or for two distant neurons to be connected. This functionality is provided by accumulating charge from the synapses as usual, but the value from the accumulator is forwarded to another neuron and then reset for each network cycle. Each element has maximally four fan-in ports in which it can accept forwarded accumulations.

4.2 Array Design

A DANNA2 neural network is executed on an array. DANNA2 supports two primary array types with different complexity and efficiency trade-offs.

4.2.1 Configurable Grid Arrays. A configurable grid array is implemented as a fixed size rectangular grid of elements. Connectivity of elements is dependent on spatial locality to enable the array to scale to thousands or more of elements. Each neuron has configurable synapses connecting it to 24 of its neighbors in a 5 by 5 connectivity grid pattern centered around the neuron. This array style is similar to DANNA and works well as a general purpose neuromorphic architecture for a programmable VLSI chip. One IC could support a very large number of different network configurations and varying applications. We expect DANNA2 grid arrays to be smaller in overall element count as compared to DANNA for a given set of resources. However, we also expect that the increased capability of a single element will result in an effective gain in overall density because fewer DANNA2 elements will be required to solve a particular task.

4.2.2 Sparse Arrays. A sparse array instead only implements the elements and connectivity required for a given network. The advantage of this approach is connectivity is no longer limited to a fixed grid with spatial locality. Any neuron is able to connect to any other neuron with only the limitation of 24 total input synapses per element. Because unused elements and connections are removed, this approach gives way to increased practical density as well as improved power efficiency. However, this approach is primarily viable for FPGAs because the array is no longer externally programmable at runtime. We expect this to allow for lower size, area, and weight of FPGA based deployments. In the results section, we also explore the effect that more flexible connectivity can have on training performance.

5 APPLICATION FRAMEWORK

In order to support the development and usefulness of hardware implementations, we have developed a software ecosystem to support the simulation, training, and evaluation of neuromorphic networks. DANNA2 is able to take advantage of the TENNLab software

stack which provides a framework for models, applications, and training.[8]

5.1 Simulator

The DANNA2 model can be simulated on a traditional CPU using an event-based simulator. The simulator may be configured with a network either through function calls, conversion from a general graph model, or from a text file serialization. Input spikes are scheduled at specified times, and the simulator is given a cycle count to run. Upon completion, the simulator can return the network outputs and the updated network state (i.e., updated synapse weights from STDP).

The simulator is designed to be very high performance in order to allow for easy training of new networks. The network graph is stored as a key (coordinates) and value (neuron) pair in a hash table. Each fire event is stored in a circular buffer of 16 queues corresponding to the maximum delay. An event is scheduled by inserting into the corresponding queue whose index is relative to the current network cycle time. Fire events within a single network cycle are unordered and do not have intra-cycle dependencies.

After all fires are processed for a given cycle, new fire events are generated for each neuron which has a charge exceeding its threshold as long as the neuron is not in refractory. If a neuron fires, it resets its charge back to its default zero state and enters a refractory period. The simulator simply stores the last time that a given neuron has fired to determine if a neuron is currently in refractory. If leak is enabled, the charge leak is only calculated when the neuron receives a new incoming spike. Likewise, STDP weight adjustments are calculated when a synapse fires and when a post-synaptic neuron fires.

5.2 Training

Networks are trained using genetic algorithms we call Evolutionary Optimization (EO), previously described in [12]. The cited version of EO used model-specific implementations of genetic operations; i.e., DANNA and NIDA each implemented the mutation and crossover operations that the EO uses. In this work, we use an EO implementation which operates on a general neural network graph model. For each epoch, prior networks are converted to general graph models. Then, EO probabilistically performs mutation and crossover operations on the general graph structures to generate new networks. These are converted back to model-specific networks and are evaluated for their performance. Just as in the previous implementation of EO, this process continues until the fitness values reaches a predefined threshold or the epoch limit is reached.

One particular benefit of using the general graph implementation of EO is that improvements to the EO are automatically applied to DANNA, NIDA, DANNA2, and any other model that can convert to and from the general graph representation. Using this implementation also provides a common code base to allow for us to compare varying models.

6 RESULTS

In order to evaluate the DANNA2 model, we benchmarked it against previous models within the same software framework across both control system and classification use cases. For each application, we

ran 100 EO training sessions for each model with different starting seeds for the random number generator. We report the number of epochs, time elapsed, final fitness value, and testing value for each. These benchmarks are intended to be relatively short running tests to demonstrate differences between approaches. Often, final network training will run for more epochs with larger population sizes to obtain maximal accuracy.

All models are run with the default parameters, and the same EO hyper-parameters are used across all models and all applications. DANNA2 is running without any neuron leak and without STDP enabled. Future papers may independently explore the effect of these features and parameters.

6.1 Inverted Pendulum

6.1.1 Application Description. The inverted pendulum or pole-balancing task is a classic control problem described in detail in [11]. The goal is to keep a pole balanced on cart by applying a force to the left or right side of the cart. The cart must stay in the range $[-2.4 \text{ m}, 2.4 \text{ m}]$, and the pole's angle must stay in the range $[-12^\circ, 12^\circ]$. The cart-pole system has four variables describing the state: the position of the cart, the velocity of the cart, the angle of the pole, and the angular velocity of the pole. Every 0.02 seconds, we apply the state to the network, run the network for 100 cycles, and apply the output of the network to the cart.

The network has eight inputs and two outputs. The outputs correspond to pushing right or left; whichever fires more is the direction the cart is pushed. There are two inputs for each state variable—one indicates a positive value and the other negative. The magnitude of the value is rate-encoded by mapping from that value's range to the range $[1 \text{ pulse}, 10 \text{ pulses}]$. The velocity terms don't explicitly have a range, so we use $\pm 2 \text{ m/s}$ and $\pm 2 \text{ rad/s}$.

For training, we use six different starting conditions corresponding to a starting position at $-1.2, 0$, or 1.2 meters and a start angle of -0.15 or 0.15 radians (8.6°). Each run is cut off after 15,000 timesteps (5 minutes) or when the cart hits the wall or the pole falls past 12° . The fitness is the average of the survival time of the six training cases.

The generalization test runs the system in 1927 different test cases, with the position from -2.3 to $+2.3$ meters in increments of 0.1 and the pole angle from -0.2 to $+0.2$ radians in increments of 0.01 . The testing score is normalized by dividing by 15,000.

6.1.2 Analysis. As shown in Figure 2, DANNA2 Sparse and DANNA perform the best in both the training fitness and the generalization testing. However, DANNA2 is able to reach this level of performance with much less training time. Interestingly, DANNA2 Grid appears to require more epochs to reach a suitable fitness for this application. Because of the stark difference between the configurable grid array and the sparse array performance, it seems the connectivity constraints of the grid array negatively impacted training performance. Notably, both DANNA2 arrays use a 5×10 array whereas DANNA uses a 15×15 array.

6.2 Robot Navigation

6.2.1 Application Description. Previous work implemented this application on a physical robot using DANNA in [6] and recommended using it as a benchmark [9]. The goal is to explore an

Figure 2: Benchmark of Inverted Pendulum run with a population of 400 for a maximum of 10 epochs

environment while avoiding obstacles. The robot is equipped with a lidar sensor taking measurements at 5 different angles equally spaced from -60° to $+60^\circ$. It also has two limit switches that detect when there is a dropoff in front of the robot (i.e., the edge of a table or staircase). The lidar measurements are applied to the network by firing a single pulse with the charge linearly mapped such that closer distances have a higher charge value. Each limit switch input receives a single pulse if it is activated. For scoring, the environment is divided into tiles, and the robot's score is the percentage of tiles that were touched.

For the Room benchmark, we simulate the robot in a 6m x 6m room that contains spherical obstacles of various sizes. We train with eight different obstacle distributions plus one training run with an empty room. Each run is two minutes of simulated time. If the robot collides with an obstacle, the score for that run is multiplied by 0.6 as a penalty, and the run is prematurely ended. The fitness is the average score over the nine runs. The Room and Table benchmark additionally simulates the robot on a 6m x 6m table with eight different obstacle distributions. The generalization test simulates the robot with fifty different obstacle distributions, giving the score as the average over these runs.

6.2.2 Analysis. Figure 3 shows DANNA2 outperforming both NIDA and DANNA in final training fitness and testing generalization performance. DANNA2 Grid achieves the highest average fitness and testing score in both benchmarks while also requiring the least CPU time to train. In this application, we believe the connectivity constraints are reducing the search space of possible solution networks, so unlike the inverted pendulum application, the reduced flexibility results in quicker convergence.

For the second benchmark shown in Figure 4, the difficulty of the task in the given number of epochs and time results in little appreciable difference between models for fitness and testing. The key advantage of DANNA2 here is the reduced CPU time required compared to DANNA.

In both benchmarks, DANNA2 is using a 5×10 array compared to DANNA which is using a 15×15 array. This demonstrates a significant decrease in total array size while also showing improved training performance.

Figure 3: Benchmark of RoboNav in a room run with a population of 200 for a maximum of 30 epochs

Figure 4: Benchmark of RoboNav in a room and on a table with a population of 200 for a maximum of 20 epochs

6.3 Classification of Consonants vs Vowels

6.3.1 Application Description. The TIMIT dataset contains audio from American English speakers. Each phoneme is labeled as one of six categories: stops, affricates, fricatives, nasals, semivowels/glides, and vowels. For this benchmark, we consider classifying each phoneme as a vowel or consonant. We follow the method used in [7], which considered stops, affricates, and fricatives to be consonants and did not use the nasals and semivowels/glides. Twelve Mel-frequency Cepstral Coefficients (MFCCs) and the log energy were extracted from the phoneme audio files. They were applied to the network as a series of pulses with more pulses indicating a higher MFCC. Consistent with [7], the networks were trained on 1000 samples from each class (vowel and consonant) and tested on 100 samples from each class.

6.3.2 Analysis. As displayed in Figure 5, all of the models were able to complete the 30 epochs in the allotted time, and none were able to exit early with a high enough fitness score. By and large, DANNA2 Grid and DANNA perform similarly for fitness and testing. However, DANNA2 Sparse is able to, on average, offer a couple percentage points higher fitness which carries over to the testing

Figure 5: Benchmark of Classifying Consonants vs Vowels with a population of 75 for a maximum of 30 epochs

results. In both DANNA2 cases, it again demonstrates a decreased training time as compared to DANNA and is much more on par with a software-only architecture like NIDA.

7 FUTURE WORK

This paper describes the hardware architecture and demonstrates the results of software simulation and testing. For future work, we plan to complete and fully evaluate both arrays types (configurable grid and sparse) on FPGAs to characterize hardware based performance. Currently, we are validating our VHDL implementation. We also expect to scale DANNA2 to multi-chip arrays which will allow for running very large networks.

We also plan to perform an in-depth exploration into the implications of leaky neurons and synaptic plasticity with STDP. In this paper, both features were disabled to compare the base model implementations, but we believe this functionality could have interesting effects on training and testing performance.

8 CONCLUSION

DANNA2 demonstrates a significant improvement in capability, density, simulation speed, and training performance as compared to DANNA. It builds on the concepts of DANNA and NIDA to provide the next generation of DANNA with the hopes of solving problems of increasing complexity. The ability to quickly simulate networks to accelerate the convergence of EO is a key design point. The combination of neurons and synapses into a unified element allows for efficiencies on both the software and hardware implementations leading to higher hardware clock speeds and faster simulations. Additionally, we show the ability to use a configurable grid array with constrained connectivity and a programmable $M \times N$ grid layout, and we also show the ability to use a more flexible sparse array with less constrained connectivity but less programmable hardware structure. Depending on the application deployment criteria, one array type or the other may offer unique benefits. Using this described DANNA2 architecture, we conclude the paper with results from both control and classification problems as well as future work to explore as research continues. In summary, we believe

DANNA2 offers a significant upgrade over our previous DANNA model.

ACKNOWLEDGEMENTS

The authors would like to thank John Reynolds for his advice regarding classification problems and Adam Disney for writing the TENNLab classification application.

Notice: This material is based on research sponsored by the Air Force Research Laboratory under agreement number FA8750-16-1-0065. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory or the U.S. Government.

REFERENCES

- [1] C. Daffron, J. Chan, A. Disney, L. Bechtel, R. Wagner, M. E. Dean, G. S. Rose, J. S. Plank, J. D. Birdwell, and C. D. Schuman. 2016. Extensions and Enhancements for the DANNA Neuromorphic Architecture. In *IEEE SoutheastCon 2016*. Norfolk, VA.
- [2] M. E. Dean and C. Daffron. 2016. A VLSI Design for Neuromorphic Computing. In *IEEE Annual Symposium on VLSI (ISVLSI)*. IEEE. <https://doi.org/10.1109/ISVLSI.2016.81>
- [3] M. E. Dean, C. D. Schuman, and J. D. Birdwell. 2014. Dynamic Adaptive Neural Network Array. In *13th International Conference on Unconventional Computation and Natural Computation (UCNC)*. Springer, London, ON, 129–141.
- [4] A. Disney, J. Reynolds, C. D. Schuman, A. Klibisz, A. Young, and J. S. Plank. 2016. DANNA: A Neuromorphic Software Ecosystem. *Biologically Inspired Cognitive Architectures* 9 (July 2016), 49–56.
- [5] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. 2016. Training deep spiking neural networks using backpropagation. *Frontiers in neuroscience* 10 (2016), 508.
- [6] J. P. Mitchell, G. Bruer, M. E. Dean, J. S. Plank, G. S. Rose, and C. D. Schuman. 2017. NeoN: Neuromorphic Control for Autonomous Robotic Navigation. In *IEEE 5th International Symposium on Robotics and Intelligent Sensors*. Ottawa, Canada, 136–142.
- [7] David Norton and Dan Ventura. 2010. Improving liquid state machines through iterative refinement of the reservoir. *Neurocomputing* 73, 16 (2010), 2893–2904.
- [8] J. S. Plank, G. S. Rose, M. E. Dean, and C. D. Schuman. 2017. A CAD System for Exploring Neuromorphic Computing with Emerging Technologies. In *42nd Annual GOMACTech Conference*. Reno, NV.
- [9] J. S. Plank, G. S. Rose, M. E. Dean, C. D. Schuman, and N. C. Cady. 2017. A Unified Hardware/Software Co-Design Framework for Neuromorphic Computing Devices and Applications. In *IEEE International Conference on Rebooting Computing (ICRC 2017)*. Washington, DC.
- [10] Catherine D. Schuman. 2015. *Neuroscience-Inspired Dynamic Architectures*. Ph.D. Dissertation. University of Tennessee.
- [11] C. D. Schuman and J. D. Birdwell. 2013. Dynamic Artificial Neural Networks with Affective Systems. *PLoS ONE* 8, 11 (November 2013), e80455. <https://doi.org/10.1371/journal.pone.0080455>
- [12] C. D. Schuman, J. S. Plank, A. Disney, and J. Reynolds. 2016. An Evolutionary Optimization Framework for Neural Networks and Neuromorphic Architectures. In *International Joint Conference on Neural Networks*. Vancouver.
- [13] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank. 2017. A Survey of Neuromorphic Computing and Neural Networks in Hardware. *arXiv:1705.06963*. <https://arxiv.org/abs/1705.06963>