

# Neuromorphic Computing for Temporal Scientific Data Classification\*

Catherine D. Schuman,  
Thomas E. Potok, Steven  
Young, Robert Patton  
Oak Ridge National Laboratory  
Oak Ridge, Tennessee  
[schumancd,potokte,pattonrm,  
youngsr]@ornl.gov

Gabriel Perdue  
Fermi National Accelerator  
Laboratory  
Batavia, Illinois  
perdue@fnal.gov

Gangotree Chakma, Austin  
Wyer, Garrett S. Rose  
University of Tennessee  
Knoxville, Tennessee  
[gchakma,awyer]@vols.utk.edu,  
garose@utk.edu

## ABSTRACT

In this work, we apply a spiking neural network model and an associated memristive neuromorphic implementation to an application in classifying temporal scientific data. We demonstrate that the spiking neural network model achieves comparable results to a previously reported convolutional neural network model, with significantly fewer neurons and synapses required.

## CCS CONCEPTS

• **Computing methodologies** → **Neural networks; Genetic algorithms; Hardware** → **Neural systems; Emerging technologies; Applied computing** → **Physics**;

### ACM Reference Format:

Catherine D. Schuman, Thomas E. Potok, Steven Young, Robert Patton, Gabriel Perdue, and Gangotree Chakma, Austin Wyer, Garrett S. Rose. 2017. Neuromorphic Computing for Temporal Scientific Data Classification. In *NCS '17: Neuromorphic Computing Symposium, July 17–19, 2017, Knoxville, TN, USA*. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3183584.3183612>

## 1 INTRODUCTION

Neuromorphic architectures are non-von Neumann computer architectures science that are inspired by biological brains. They are typically made up of many simple processing elements (neurons) that operate in parallel, are connected (via synapses), and that communicate with very simple messages (usually spikes). Key properties of neuromorphic systems include relatively low power consumption, real time performance, and the ability to be deployed

in real environments (i.e., they usually do not require special operating conditions). Spiking neural networks are one common model that is implemented for neuromorphic computing. Spiking neural networks (SNNs) can be implemented in an event-driven way, which can result in lower power consumption. A key component of SNNs is that they include a notion of temporal processing by including recurrent connections and/or delay components on the synapse or neuron implementations. As such, they can be natural platforms for data that has a temporal component.

A key question associated with the field of neuromorphic computing is: what are the applications for which neuromorphic computers are especially suited? In addressing this question, it is worth considering several factors beyond just accuracy on a task. For example, for certain applications, it might be the case that a neuromorphic architecture achieves good but not state-of-the-art performance, but that it is able to achieve this performance on a significantly smaller power budget and footprint than a state-of-the-art solution on other, more conventional hardware. Additionally, there are also potential applications to data sets that contain a significant temporal component. State-of-the-art machine learning techniques such as deep learning often convert temporal data into a representation that they can understand – usually some sort of image representation. Because of their native temporal processing capabilities, temporal data may be represented natively in SNNs, which may result in a better and more efficient usage of the data than an adapted version of the data.

In this work, we present a SNN model and an associated mixed analog/digital hardware implementation that utilizes memristors as applied to temporal scientific data. We apply these spiking and neuromorphic systems to a scientific data set, namely neutrino interaction location identification data from Fermi National Accelerator Laboratory. The data we utilized was created in simulation, but it is representative of the type of data that is collected using their neutrino detector. Though we are training on simulated data, a trained network could be deployed on or near the detector in order to process and classify data as it is being collected in real-time. As such, this application could benefit from a neuromorphic hardware implementation that is small and requires little power to operate, in addition to potentially being a natural type of data for neuromorphic hardware to process. In the following sections, we provide some context for the background of spiking and neuromorphic systems. We also describe the SNN architecture, the memristive neuromorphic architecture, and the training method for both that we apply to the neutrino interaction data. We present preliminary

\*This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (<http://energy.gov/downloads/doe-public-access-plan>).

Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the United States government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

*NCS '17, July 17–19, 2017, Knoxville, TN, USA*

© 2017 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.

ACM ISBN 978-1-4503-6442-3/17/07...\$15.00

<https://doi.org/10.1145/3183584.3183612>

results on the application of these systems to the neutrino interaction data set and show that the results are comparable with those achieved by convolutional neural networks.

## 2 BACKGROUND AND RELATED WORK

Spiking neural networks (SNNs) have previously been described as the “third generation” of artificial neural networks, following networks the first generation (networks with threshold units) and the second generation (networks with activation functions such as sigmoid in the neurons) [19]. Basic SNN models that are not attempting to model biological systems tend to use forms of integrate-and-fire neurons, though more complex models such as the Izhikevich spiking model have also been used [14]. Training or learning for SNNs is more difficult than the traditional feed-forward neural network with sigmoid activation function because of the additional complexity of the neuron models and/or the topology of the network. There have been back-propagation-like algorithms developed for SNNs [6, 36], but these require relatively restricted network topologies (feed-forward neural networks). Unsupervised learning techniques including both Hebbian learning and spike-timing-dependent plasticity (STDP) have been used to determine weight values in a network [13, 32], but these approaches are somewhat limited in training for more complex tasks. Evolutionary algorithms have also been used to train SNNs [20, 22, 34]. They have the advantage that they can train a variety of aspects of the network, including network topology, to suit a given task. In this work, we utilize an evolutionary approach for training two similar SNN models.

Many neuromorphic computing systems implement SNNs. One of the popular components included in today’s neuromorphic systems are memory-resistors, or memristors, because they can operate with low energy and have the potential for high density [15]. Moreover, memristors are typically used in synapse implementations because of their behavioral similarity to biological synapses [18]. In particular, the resistive switching characteristics of memristors allows the “weight” value of the memristive synapse to change as a result of a learning process such as STDP [30]. Memristors can be made from a variety of materials, including metal oxides [25], polymers [7], and organic materials [12].

In this work, we apply SNNs and neuromorphic networks to temporal data. This is a natural use of SNNs because of their inherent temporal processing capabilities. SNNs have previously been applied to spatiotemporal data such as object recognition in video [2, 3, 16, 21] and spatio-spectral data such as EEGs [17, 24, 31]. It is also worth noting that neuromorphic systems have been applied to high energy physics problems in the past, including particle collision detection [9, 35]. Both of these implementations are for non-spiking multi-layer perceptrons implemented on field-programmable gate arrays (FPGAs), primarily to achieve the speed required for particle collision detection tasks.

## 3 SPIKING NEURAL NETWORK MODEL: NIDA

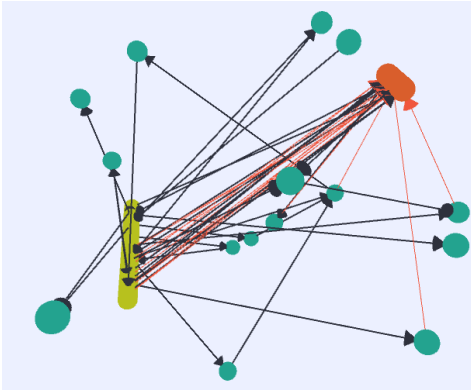
The spiking neural network (SNN) model that we first tested with the neutrino detector data is the Neuroscience-Inspired Dynamic Architecture (NIDA) spiking neural network model [27]. This model uses very simple neuron and synapse models and thus is well-suited

to implementations in hardware. Implementations in hardware that are based on the NIDA model include an FPGA-based model [10], a forthcoming fully digital custom chip implementation [8], and an implementation that includes memristors in the synapses, which is discussed further in Section 4. We perform initial experiments with NIDA because it can give a good idea of what the performance characteristics will be for the system, but with a faster simulation time than is possible for some of our hardware implementations. It also allows us to explore the capabilities of a spiking model when the connectivity in the network is not restricted due to physical hardware limits. However, it is worth noting that, if the resulting networks are relatively sparse, it is likely that an equivalent model can be built using additional neurons in hardware with connectivity restrictions. The only restriction on the topology of NIDA networks is that they must be simple directed graphs; that is, loops (a synapse connecting a neuron to itself) are not allowed, nor are multiple synapses with the same pre- and post-synaptic neurons.

NIDA neurons have two parameters: a threshold and a refractory period. The threshold value of the neurons in the network governs how much charge is required for the neuron to fire, and the refractory period of the neuron defines the period of time after that neuron has fired during which it cannot fire again. In this work, we train the threshold value for the neurons, but the refractory period is fixed to a single value for all neurons in the network. NIDA synapses have two parameters: weight and delay. NIDA networks are embedded into a three-dimensional Euclidean space, and currently, the distance between the pre- and post-synaptic neurons in the 3D space defines the delay between two neurons. As such, the delay value is not explicitly programmed, except in the placement of the pre- and post-synaptic neuron. The weight value, however, is trained in two ways. First, weights can be set using the evolutionary optimization training method that is used to train the rest of the characteristics of the network (described in Section 5). Second, weights can be refined using long-term potentiation and long-term depression mechanisms, which adjust the weight values based on the firing activity of the post-synaptic neuron. An example of a NIDA network is shown in Figure 1.

## 4 MEMRISTIVE HARDWARE IMPLEMENTATION

The memristive hardware implementation is synchronous in nature, and it has digital peripheral circuits whereas the core computation is analog in nature. This mixed-signal implementation is more power and area efficient because we are leveraging nano-scale memristive devices to hold synaptic weights and analog neurons to compute integration of charges. The memristive implementation of the synapse utilizes a twin memristor architecture, which helps to achieve both positive and negative synaptic weights. Here, the synaptic weight is proportional to the current flowing through the memristors and hence depends on the memristance of the twin memristors. The synapses are also capable of online learning. The synapse control block generates driving voltage for the memristors and also controls the online learning based on long term plasticity. The online learning is based on the NIDA learning rule. According to the potentiation and depression rule, the synapses contributing to the post-synaptic fire would be potentiated and the rest of



**Figure 1: An example NIDA network. Hidden neurons are shown in teal, input neurons are shown in yellow, and output neurons are shown in red. Excitatory synapses are shown in dark blue, and inhibitory synapses are shown in red. The visualization tool used to display the network is described in [11].**

synapses arriving immediately after the post synaptic fires would be depressed. The neuron model used in the mrDANNA architecture is based on a leaky integrate and fire neuron model. The neurons work in two different phases: the accumulation phase and the firing phase. When the neuron is in the accumulation phase, it acts as an integrator and accumulates charge based on the supplied current through the synaptic weights. On the firing phase the neuron generates a post-synaptic fire if the accumulated charge is higher than a specified threshold. The neuron also resets itself after generating post synaptic fire and starts from the initial condition.

For the simulation of mrDANNA, we use two types of simulators: a high-level C++ simulator and a low-level Spectre simulator. The high-level simulator takes a network generated using evolutionary optimization (described below) and the components of the high-level simulator are built following the transistor-level neuron and synapses. The low-level simulator is used to build circuit components and simulate in real time. Since it is time consuming to simulate the neural networks in the low-level simulator, we have used the high-level simulator to help build network in a time efficient way. The high-level simulator has also been helpful to verify the functionality of the Spectre simulator.

## 5 TRAINING METHOD: EVOLUTIONARY OPTIMIZATION

For both NIDA and mrDANNA, we utilize a training method that is based on evolutionary optimization (EO) [29]. The EO method determines the topology of the network (the number and connectivity of the neurons and synapses), as well as the parameters of the network (i.e., weights and delays of the synapses and thresholds of the neurons). The basic EO method utilized begins with a randomly initialized population of networks for both NIDA and mrDANNA, where each network in the population has the same input and output neurons. Each network in the population is evaluated using a fitness function. The fitness function for the classification of the

data in this work is given in Section 6. Higher performing networks are preferentially selected to serve as parent networks using tournament selection. Once two parent networks have been selected, crossover and mutation operations are probabilistically applied. Both crossover and mutation are custom operations created for the representations of NIDA and mrDANNA. Crossover relies on the spatial structure of the direct representation of the NIDA network in the three-dimensional Euclidean space and the two-dimensional spatial layout of mrDANNA networks. Mutation operations are similar for both NIDA and mrDANNA, and they include updating parameter values, inserting or deleting neurons, and inserting or deleting synapses.

There are several key properties for why EO was selected as the training method. The first is that it is model and topology agnostic. In other words, EO can be applied to any network model and any hardware implementation; this differentiates EO from an algorithm like back-propagation, which relies on a particular network structure and model. We take advantage of this characteristic by employing a very similar EO method for both NIDA and mrDANNA, which are similar in model but have different operating characteristics. The second is that it can deal with and optimize temporal processing components (such as delay values on the synapses) in the network, which is an important factor for training SNNs. The third reason for utilizing EO as a training method is that it can be parallelized and scaled. We have previously implemented a parallel EO framework and demonstrated that utilizing a larger number of nodes can allow for a shorter time-to-solution [28]. In this work, we will discuss scaling this parallel EO framework to a very large number of nodes (18,000 nodes) on the application described below. Though this application (and applications in general) tend to benefit from scaling up to a larger number of nodes, in general we have found that simply utilizing as few as between 10 and 100 nodes over a longer period of time can give similarly good results.

## 6 APPLICATION: NEUTRINO DATA

The data used in this work is neutrino scattering data that was collected at Fermi National Accelerator Laboratory using the MINERvA detector [4]. The MINERvA detector collects a variety of information, including energy levels and the times at which certain events occur. This data may be analyzed to discover at point in space neutrino interactions occurred (vertex reconstruction). In a previous work, convolutional neural networks have been applied to some of this data for vertex reconstruction by converting energy level information into images, which are then classified [33]. There are three “views” provided in the MINERvA detector: an X view (in this work, we use 127x50 measurements from the X view, though additional measurements are taken), a U view (127x25 measurements in this work), and a V view (127x25 measurements in this work). There are also additional measurements taken in the outer region of the detector. We restrict our attention to the X view to reduce input size for our neuromorphic systems, but we plan to expand to include the U and V views in future work. The physical space that is being analyzed is divided into segments labeled from 0 to 10 (inclusive). The goal is to take information from the X view and to determine in which segment the neutrino interaction occurred (the vertex).

In [33], the input data to the convolutional neural network is an image, where every pixel in the image represents the mean value of the energy at that location. In this work, we utilize a different aspect of the data. In particular, part of the data collected by the detector is the time at which energy value at each location exceed a very low threshold. These times are recorded for each of the 127x50 measurements for the X-view.

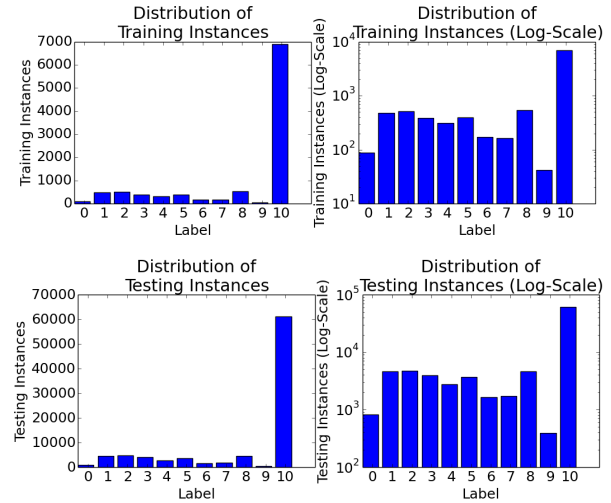
We utilize a training set of 10,000 instances generated by the GENIE Neutrino Monte Carlo Generator[5] and the Geant4 toolkit [1], so ground-truth information is available. This is a relatively small subset of the overall number of instances (over 1,000,000), so we utilize another 90,000 instances for the testing set. However, we keep the training set small in order to keep the fitness evaluation time relatively short. The number of training and testing instances for each label are shown in Figure 2. As can be seen in this figure, a majority of the instances are label 10. This label is applied to instances that fall within the physical segment labeled 10, but it is also applied to instances in which interactions occur downstream.

One of the key factors that must be addressed in order to utilize neuromorphic systems for applications is how the input information is encoded. In our data set, each instance is a 127x50 array of values  $A$ , where each value in the array corresponds to a time offset from a specific event occurring. We normalize the temporal values so they are greater than or equal to 0, and set all values for which no events were measured to -1 to create a matrix  $B$ . We initialize our networks to have 50 inputs neurons (corresponding to the dimension of 50 in the input data), and eleven output neurons (one corresponding to each possible segment label). Then, for each value  $i = 1, \dots, 127$  and for each value  $j = 1, \dots, 50$ , if  $B_{i,j}$  is non-negative we create a fire event at time  $B_{i,j}$  on input neuron  $j$ . Thus, input neuron  $j$  may have up to 127 input fire events for any given input instance, assuming all of the events for that dimension occurred at unique times. We count the number of times all 11 output neurons fire over the course of simulation, and the output neuron that fires the most defines the label. If none of the output neurons fire, the decision for the network is set as label 10, which also applies to instances where the interaction occurs downstream.

## 7 PRELIMINARY RESULTS

In this section, we present preliminary results of applying both NIDA and mrDANNA networks to classifying MINERvA data. These networks were trained using both the basic EO, which is meant to be run on a multi-core, shared memory device, and the large-scale parallel EO, which is meant to be run on a multi-node, distributed memory supercomputer or cluster. For the large-scale parallel EO method, we used Oak Ridge Leadership Computing Facility’s Titan<sup>1</sup>, which has 18,688 compute nodes and 299,008 CPU cores. The basic EO method utilizes a single population, while the parallel EO method utilizes subpopulations on each node with communication between nodes in a hierarchical master-slave configuration, described in [28].

Table 1 shows the maximum fitness value achieved by a NIDA network for the training set when training using the basic EO method on a single node of Titan for one hour as compared with the performance of one hour of evolution with 100 nodes, 1,000 nodes



**Figure 2: Breakdown of training (top) and testing (bottom) instances per label, with the log-scale plot shown on the right of both.**

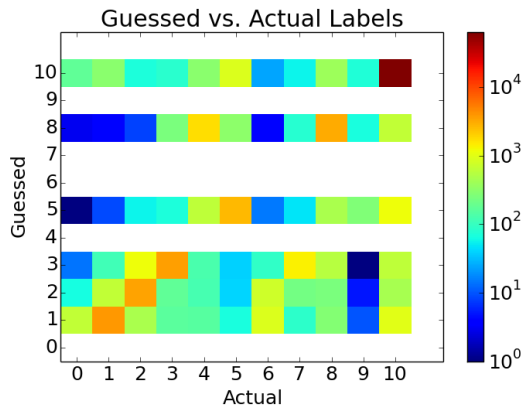
**Table 1: Parallel EO training after one hour**

Number of Nodes	Maximum Fitness Value
1	69.05%
100	70.82%
1000	72.6%
10000	79.11%

and 10,000 nodes for one hour utilizing the parallel EO method. Obviously, the best result was achieved by the 10,000 node method. It also produced the best result to date for our method. Additionally, we tested a 3,750 node run for 24 hours, which produced a network that achieves 81.57 percent classification accuracy on the training set. We used that network to seed several 50 nodes runs to further train the networks. The resulting best network achieved 81.06 percent accuracy on the 100,000 data instances of the combined training and testing set, with 81 percent accuracy only on the testing set. The results reported in [33] for a convolutional neural network using just the X-view produced 80.42 percent classification accuracy on the full data set. The convolutional neural network is made up of four convolution layers, four pooling layers, three fully-connected layers, and two dropout layers. In contrast, the resulting NIDA network (shown in Figure 1) only contained 90 neurons (including inputs and outputs) and 86 synapses. There are fewer synapses than neurons, thus, there are some input and/or output neurons that are ignored. It is worth noting that we utilize a high performance computer to quickly get to a good solution using the EO distributed method. The EO benefits both from massively parallel simulation and a large population size, but similar results can be achieved on a desktop, though they require significantly longer to train.

Figure 3 shows the guessed labels from the best performing NIDA network. As can be seen, the NIDA network never guessed the labels

<sup>1</sup><https://www.olcf.ornl.gov/computing-resources/titan-cray-xk7/>



**Figure 3: Heatmap showing the labels guessed by the NIDA network and the actual labels in the data set. Note that the colorbar is shown on a log-scale (because of the skewed instances towards label 10). White on the heatmap corresponds to no values.**

0, 4, 6, 7, or 9. In Figure 2, we can see that those are the classes with the fewest instances in both the training and testing set. This result shows that the EO process has favored networks that focus on categorizing as many training instances as possible correctly (as specified by the fitness function), and since those labels have the fewest instances in the training set, the EO focuses on optimizing classifying the most frequently used labels. To overcome this in the future, we can adjust the fitness function to favor networks that better cover all of the different labels.

We also trained mrDANNA networks for the neutrino task, using just the basic EO method on a desktop. The fitness function evaluation for mrDANNA requires more time than the NIDA evaluation time, and thus requires longer to evolve. The current resulting mrDANNA network achieved approximately 76.14 percent classification accuracy on the training set and 73.59 percent classification accuracy on the combined training and testing set, but its training trajectory was similar to that of NIDA network training on a desktop. We expect that allowing additional training time for mrDANNA will yield a network with similar performance as the NIDA network shown above. We are able to estimate the energy usage of the mrDANNA network using our combination of low- and high-level simulators. This network requires approximately  $1.66 \mu\text{J}$  per calculation. To calculate the energy per calculation we determined the energy per neuron in accumulation, fire and idle phase and the energy per synapse in active and passive phase using the low level simulator. We utilized our high level simulator to simulate neural networks and to get an estimation of the number of neurons and synapses in different phases. Then we multiplied each energy number with the total number of neurons and synapses to get an estimated energy per calculation.

## 8 DISCUSSION AND FUTURE WORK

There is much work left to do on the neutrino interaction data discussed in this work. There are two other views of data (U view

and V view) that can also be used in classifying where interactions occur. The best results for the convolutional neural network (>90 percent accuracy) were achieved using a combination of the three views. There are two ways to approach this: include all three views as input to one network or build three different networks (one for each view) and combine the results to produce the output label. Our previous experience indicates that the latter will be the better approach, so we intend to pursue that approach moving forward.

We are encouraged by the results on the neutrino interaction data, and we intend to explore other scientific data applications, including other high energy physics data, materials science data, and medical data where there is a temporal component that maps well to spike-based systems. We also plan to explore how to combine convolutional neural networks (which do very well at processing spatial data) and SNNs in order to understand how we may leverage the strengths of each system as applied to scientific data sets.

We utilize evolutionary algorithms as a way to train spiking neuromorphic systems. We plan to explore other training algorithms and/or network models that can leverage spiking neuromorphic systems as well, including liquid state machines, which have been shown to perform well on temporal data analysis [26]. Liquid state machines leave the spiking neuromorphic component “untrained”, and rely on training a readout layer using back-propagation or some other gradient descent approach.

We also intend to explore other neuromorphic device implementations. The EO, our models (NIDA and mrDANNA), and the neutrino interaction data are implemented in a single software framework [23]. This framework allows us to easily include a new neuromorphic device model (e.g., a digital neuromorphic device or a non-memristive analog or mixed analog/digital neuromorphic device), and to compare performance across different implementations. It also allows us to easily incorporate in new applications (such as other temporal scientific data applications discussed above).

## 9 CONCLUSION

In this work, we present preliminary results on the application of spiking neuromorphic systems to temporal scientific data. We show that spiking neuromorphic systems can achieve a comparable result to a convolutional neural network applied to the same “view” of a neutrino interaction data set, but with a much smaller network. The goal of this work is to demonstrate that by utilizing a spike-based encoding of temporal information (which is a “native” data type for spiking neuromorphic systems), we can achieve similar results to other, state-of-the-art machine learning systems, such as convolutional neural networks.

We also show that the memristive implementation requires less than  $2 \mu\text{J}$  per classification, indicating that this network may be realistically deployed in an energy efficient device. We expect that the best results for processing scientific data will use multiple machine learning techniques, including both convolutional neural networks and other deep learning techniques such as long short-term memory network. We seek to emphasize that SNNs and spiking neuromorphic systems have the potential to process temporal data in an efficient and accurate way and that they should be considered in the landscape of machine learning mechanisms for processing scientific data.

## ACKNOWLEDGEMENTS

This material is based upon work supported in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under contract number DE-AC05-00OR22725. Research sponsored 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. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. We would like to thank the MINERvA collaboration for the use of their simulated data and for many useful and stimulating conversations. MINERvA is supported by the Fermi National Accelerator Laboratory under US Department of Energy contract No. DE-AC02-07CH11359 which included the MINERvA construction project. MINERvA construction support was also granted by the United States National Science Foundation under Award PHY-0619727 and by the University of Rochester. Support for participating MINERvA physicists was provided by NSF and DOE (USA), by CAPES and CNPq (Brazil), by CoNaCyT (Mexico), by CONICYT (Chile), by CONCYTEC, DGI-PUCP and IDI/IGIUNI (Peru), and by Latin American Center for Physics (CLAF).

## REFERENCES

- [1] S. Agostinelli et al. 2003. GEANT4: A Simulation toolkit. *Nucl.Instrum.Meth.* A506 (2003), 250–303. [https://doi.org/10.1016/S0168-9002\(03\)01368-8](https://doi.org/10.1016/S0168-9002(03)01368-8)
- [2] Himanshu Akolkar, Cedric Meyer, Xavier Clady, Olivier Marre, Chiara Bartolozzi, Stefano Panzeri, and Ryad Benosman. 2015. What can neuromorphic event-driven precise timing add to spike-based pattern recognition? *Neural computation* (2015).
- [3] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, et al. 2015. TrueNorth: Design and Tool Flow of a 65mW 1 Million Neuron Programmable Neurosynaptic Chip. (2015).
- [4] L Aliaga, L Bagby, B Baldin, A Baumbaugh, A Bodek, R Bradford, WK Brooks, D Boehnlein, S Boyd, H Budd, et al. 2014. Design, calibration, and performance of the MINERvA detector. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 743 (2014), 130–159.
- [5] Costas Andreopoulos, A Bell, D Bhattacharya, F Cavanna, J Dobson, S Dytman, H Gallagher, P Guzowski, R Hatcher, P Kehayias, et al. 2010. The GENIE neutrino monte carlo generator. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 614, 1 (2010), 87–104.
- [6] Sander M Bohte, Joost N Kok, and Han La Poutre. 2002. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing* 48, 1 (2002), 17–37.
- [7] Yu Chen, Gang Liu, Cheng Wang, Wenbin Zhang, Run-Wei Li, and Luxing Wang. 2014. Polymer memristor for information storage and neuromorphic applications. *Materials Horizons* 1, 5 (2014), 489–506.
- [8] Mark E Dean and Christopher Daffron. 2016. A VLSI Design for Neuromorphic Computing. In *VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on*. IEEE, 87–92.
- [9] Bruce Denby, Patrick Garda, Bertrand Granado, Christian Kiesling, Jean-Christophe Prévotet, and Andreas Wassatsch. 2003. Fast triggering in high-energy physics experiments using hardware neural networks. *Neural Networks, IEEE Transactions on* 14, 5 (2003), 1010–1027.
- [10] Adam Disney, John Reynolds, Catherine D Schuman, Aleksander Klibisz, Aaron Young, and James S Plank. 2016. DANNA: A neuromorphic software ecosystem. *Biologically Inspired Cognitive Architectures* 17 (2016), 49–56.
- [11] Margaret Drouhard, Catherine D Schuman, J Douglas Birdwell, and Mark E Dean. 2014. Visual analytics for neuroscience-inspired dynamic architectures. In *Foundations of Computational Intelligence (FOCI), 2014 IEEE Symposium on*. IEEE, 106–113.
- [12] Victor Erokhin, Tatiana Berzina, Anteo Smerieri, Paolo Camorani, Svetlana Erokhina, and Marco P Fontana. 2010. Bio-inspired adaptive networks based on organic memristors. *Nano Communication Networks* 1, 2 (2010), 108–117.
- [13] Stefano Fusi. 2002. Hebbian spike-driven synaptic plasticity for learning patterns of mean firing rates. *Biological cybernetics* 87, 5 (2002), 459–470.
- [14] Eugene M Izhikevich. 2003. Simple model of spiking neurons. *IEEE Transactions on neural networks* 14, 6 (2003), 1569–1572.
- [15] Sung Hyun Jo, Ting Chang, Idongesit Ebong, Bhavitavya B Bhadviya, Pinaki Mazumder, and Wei Lu. 2010. Nanoscale memristor device as synapse in neuro-morphic systems. *Nano letters* 10, 4 (2010), 1297–1301.
- [16] Nikola Kasabov, Kshitij Dhoble, Nuttapod Nuntalid, and Giacomo Indiveri. 2013. Dynamic evolving spiking neural networks for on-line spatio-and spectro-temporal pattern recognition. *Neural Networks* 41 (2013), 188–201.
- [17] Dhireesha Kudithipudi, Qutaiba Saleh, Cory Merkel, James Thesing, and Bryant Wysocki. 2015. Design and Analysis of a Neuromemristive Reservoir Computing Architecture for Biosignal Processing. *Frontiers in Neuroscience* 9 (2015), 502.
- [18] Bernabé Linares-Barranco and Teresa Serrano-Gotarredona. 2009. Memristance can explain spike-time-dependent-plasticity in neural synapses. (2009).
- [19] Wolfgang Maass. 1997. Networks of spiking neurons: the third generation of neural network models. *Neural networks* 10, 9 (1997), 1659–1671.
- [20] Liam P Maguire, T Martin McGinnity, Brendan Glackin, Arfan Ghani, Ammar Belatreche, and Jim Harkin. 2007. Challenges for large-scale implementations of spiking neural networks on FPGAs. *Neurocomputing* 71, 1 (2007), 13–29.
- [21] Garrick Orchard, Xavier Lagorce, Christoph Posch, Steve B Furber, Ryad Benosman, and Francesco Galluppi. 2015. Real-time event-driven spiking neural network object recognition on the SpiNNaker platform. In *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on*. IEEE, 2413–2416.
- [22] NG Pavlidis, OK Tasoulis, Vassilis P Plagianakos, G Nikiforidis, and MN Vrahatis. 2005. Spiking neural network training using evolutionary algorithms. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, Vol. 4. IEEE, 2190–2194.
- [23] 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.
- [24] Anvesh Polepalli, Nicholas Soares, and Dhireesha Kudithipudi. 2016. Digital neuromorphic design of a Liquid State Machine for real-time processing. In *Rebooting Computing (ICRC), IEEE International Conference on*. IEEE, 1–8.
- [25] Mirko Prezioso, Farnood Merrikh-Bayat, BD Hoskins, GC Adam, Konstantin K Likharev, and Dmitri B Strukov. 2015. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* 521, 7550 (2015), 61–64.
- [26] Benjamin Schrauwen, Michiel DăĂZhaene, David Verstraeten, and Jan Van Campenhout. 2008. Compact hardware liquid state machines on FPGA for real-time speech recognition. *Neural networks* 21, 2 (2008), 511–523.
- [27] Catherine D Schuman, J Douglas Birdwell, and Mark E Dean. 2014. Spatiotemporal classification using neuroscience-inspired dynamic architectures. *Procedia Computer Science* 41 (2014), 89–97.
- [28] Catherine D Schuman, Adam Disney, Susheela P Singh, Grant Bruer, J Parker Mitchell, Aleksander Klibisz, and James S Plank. 2016. Parallel evolutionary optimization for neuromorphic network training. In *Proceedings of the Workshop on Machine Learning in High Performance Computing Environments*. IEEE Press, 36–46.
- [29] Catherine D Schuman, James S Plank, Adam Disney, and John Reynolds. 2016. An evolutionary optimization framework for neural networks and neuromorphic architectures. In *Neural Networks (IJCNN), 2016 International Joint Conference on*. IEEE, 145–154.
- [30] Teresa Serrano-Gotarredona, Timothée Masquelier, Themistoklis Prodromakis, Giacomo Indiveri, and Bernabe Linares-Barranco. 2013. STDP and STDP variations with memristors for spiking neuromorphic learning systems. *Frontiers in neuroscience* 7 (2013), 2.
- [31] Juncheng Shen, De Ma, Zonghua Gu, Ming Zhang, Xiaolei Zhu, Xiaoqiang Xu, Qi Xu, Yangjing Shen, and Gang Pan. 2016. Darwin: a neuromorphic hardware co-processor based on Spiking Neural Networks. *Science China Information Sciences* (2016), 1–5.
- [32] Sen Song, Kenneth D Miller, and Larry F Abbott. 2000. Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nature neuroscience* 3, 9 (2000), 919–926.
- [33] Adam M. Terwilliger, Gabriel N. Perdue, David Isele, Robert M. Patton, and Steven R. Young. 2017. Vertex Reconstruction of Neutrino Interactions using Deep Learning. In *Neural Networks (IJCNN), 2017 International Joint Conference on*. In Press.
- [34] Andres Upegui, Carlos Andrés Pena-Reyes, and Eduardo Sanchez. 2005. An FPGA platform for on-line topology exploration of spiking neural networks. *Microprocessors and microsystems* 29, 5 (2005), 211–223.
- [35] Salvatore Vitabile, Vincenzo Conti, Fulvio Gennaro, and Filippo Sorbello. 2005. Efficient MLP digital implementation on FPGA. In *Digital System Design, 2005. Proceedings. 8th Euromicro Conference on*. IEEE, 218–222.
- [36] Jianguo Xin and Mark J Embrechts. 2001. Supervised learning with spiking neural networks. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, Vol. 3. IEEE, 1772–1777.