Resilience and Robustness of Spiking Neural Networks for Neuromorphic Systems

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

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

J. Travis Johnston
Oak Ridge National Laboratory
Oak Ridge, TN, USA
johnstonjt@ornl.gov

Maryam Parsa
Electrical and Computer Engineering
Purdue University
West Lafayette, Indiana, USA
mparsa@purdue.edu

Bill Kay
Oak Ridge National Laboratory
Oak Ridge, TN, USA
kaybw@ornl.gov

Prasanna Date
Oak Ridge National Laboratory
Oak Ridge, TN, USA
datepa@ornl.gov

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

***Abstract*—Though robustness and resilience are commonly quoted as features of neuromorphic computing systems, the expected performance of neuromorphic systems in the face of hardware failures is not clear. In this work, we study the effect of failures on the performance of four different training algorithms for spiking neural networks on neuromorphic systems: two back-propagation-based training approaches (Whetstone and SLAYER), a liquid state machine or reservoir computing approach, and an evolutionary optimization-based approach (EONS). We show that these four different approaches have very different resilience characteristics with respect to simulated hardware failures. We then analyze an approach for training more resilient spiking neural networks using the evolutionary optimization approach. We show how this approach produces more resilient networks and discuss how it can be extended to other spiking neural network training approaches as well.**

***Index Terms*—neuromorphic computing, spiking neural networks, resilience**

I. Introduction

Neuromorphic computing systems offer great promise for the future of computing. As noted in [1], one of the most frequently cited characteristics of neuromorphic computing systems that make them attractive for real-world applications is that they have inherent fault tolerance, robustness and resilience characteristics. Resilience and robustness characteristics of traditional neural networks have been widely studied, but it is not clear how these results will translate to the types of spiking neural networks (SNNs) that are implemented on neuromorphic systems. In particular, it is not clear what the resilience and robustness characteristics of SNNs are. Moreover, there are several different types of common algorithms for training SNNs, including backpropagation-like algorithms, liquid state machines, and evolutionary optimization. The networks that are produced by these different algorithms have very different network topologies and performance characteristics, and it is not clear how these differences will impact their fault tolerance capabilities.

Specifically for neuromorphic computing, understanding the resilience characteristics of SNNs trained using different algorithms is likely to become increasingly important. In particular, a key area of research in neuromorphic computing is in new devices and materials to implement various components of neuromorphic systems. New and emerging devices that are currently being studied in neuromorphic computing include metal oxide memristors and other non-volatile memory technologies [2], [3], biomimetic and biomolecular materials [4], and optoelectronic [5] implementations. Because these are experimental devices, there are likely to be failures due to hardware-level issues [6]. Thus, when designing SNNs for neuromorphic deployment, understanding how those networks will behave in the presence of corruption in the network due to device failures is important in understanding how neuromorphic systems will behave on real world applications.

In this work, we seek to provide a baseline understanding of the resiliency properties of SNNs trained using a variety of training algorithms. The primary type of failure we focus

Notice: This manuscript has been authored in part 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, world-wide 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).

on in this work is synapse failure, though other types of failures (malfunctioning synapses or neurons, neuron failures, etc.) may also be common. We examine SNNs trained using four different algorithms and how those SNNs perform in the face of synapse failures. We show that these four different algorithms have very different performance with respect to failures in the network. We propose an approach to train for resiliency of SNNs for one particular algorithm type (evolutionary optimization) and we show how it improves the resiliency of the resulting SNNs. We then discuss how this approach can be extended for the other algorithms in future work.

II. Background and Related Work

The resilience, robustness, and fault tolerance capabilities of traditional neural networks have been widely studied over the last few decades, and there have been several algorithms proposed to improve these characteristics in traditional neural networks [7]–[11], as well as deep learning networks [12], [13]. However, because there are fundamental differences in how SNNs perform computation, as well as different types of training approaches, it is not clear that any of these approaches will translate to SNNs.

One of the key technologies used to implement neuromorphic systems is memristors. However, memristors are known to have several different types of faults, so there has been a significant amount of work in addressing those challenges from the perspective of the algorithms and architectures. In [14], memristor faults are addressed by analyzing memristors laid out in a *crossbar* topology. The crossbar array provides a dense layout where every input is connected to every output. Within the crossbar array, some proportion of the memristors were selected to be locked at either a high or low resistance without being able to switch. The array with induced faults was trained using SPICE and MATLAB, and the networks with faulty memristors were still trained. Even in the case that 50% of the memristors were faulty, in several trials the network was trained in fewer than 50 epochs. Hence, the highly connected crossbar topology was (in some instances) resilient with a 50% fault rate for the embedded memristors.

In [6], a two layer neural network that achieves an accuracy of 92.64% on the MNIST dataset is used to illustrate the impact single-bit failures (SBF) can have on a memristor neuromorphic network. The synaptic weight matrix is mapped to the conductances of a memristor array in such a neuromorphic system. Hence, synapse failures in this neural network are in correspondence with SBFs in the memristor-based system. With an SBF rate of 20%, the accuracy drops from 92.64% to 39.4%. To increase the fault tolerance of this network, the normalized accuracy (ACC_{real}/ACC_{ideal}) is introduced, and several protocols to increase the normalized accuracy are examined. First, it is noted that there are significant and insignificant weights. These weights are identified, and an analysis shows that if only insignificant weights are affected the normalized accuracy can drop by as little as 1%. Next, a retraining protocol is introduced where the failed bits are held constant and the other synaptic weights are initialized as in the original model. The network is still trained with failed bits held fixed and all other bits updating in each training step. At a 20% SBF rate, the normalized accuracy can be recovered up to 98.1%. Lastly, a remapping protocol is introduced where a small number of memristor columns in which failure occurs are replaced by new memristor columns. Namely, the columns corresponding to the 5% of the most significant weights which fail. The normalized accuracy is improved to 99.3% after this remapping procedure. It is noteworthy the impact of *which* bits fail has on performance. If the SBF rate is 30% but occurs only on insignificant weights, the normalized accuracy can be brought up to 99.9%. On the other hand, if these failures occur on the most significant weights the normalized accuracy can only be recovered to 95.1%. Hence, identifying critical bits, retraining after bit failures, and replacing the most significant failed bits can go a long way to mitigate faults in memristive neuromorphic systems.

Fault tolerance of Resistive Random Access Memory (RRAM) and RRAM-based Computing Systems (RCS) is provided in [15], [16]. An RCS is a high speed, low energy, in-memory, *multi-bit* (i.e., more information for less hardware) design which utilized memristors in a crossbar topology as RAM. There are some faults, such as Stuck at 0/1 (SA0/1) faults, Transition faults (TF), and Address Decoder faults (ADF) that are common to RAM and RRAM. Hence, many of the protocols for detecting and handling faults in RAM apply to RRAM. On the other hand, the physical mechanism of RRAM gives rise to some unique faults such as Read/Write Disturbance faults (R/WDF) in which a read/write current is applied during a read/write operation, which modulates the current of the cell in unintended ways. Faults in which resistances can no longer be modulated (SA0/1 faults) are called *hard* faults, while faults in which resistances are simply not as intended (R/WDFs) are called *soft* faults. Hard faults can arise when a memristive unit has poor write endurance. That is, when many writes occur on one unit, physical fatigue can occur causing SA0/1 faults. A number of prophylactic procedures are proposed to prevent write fatigue. For example, threshold training in which weights are updated only when they change beyond a certain tolerance to prevent many unnecessary writes. Another principle example is to periodically exchange units which endure frequent writes with units which are written to infrequently. Once SA0/1 faults have occurred, the procedure outlined in [6] of weight identification, retraining, and remapping can be employed. Soft faults such as R/WDF can be dealt with by performing resistance updates on chip. Instead of reading, updating, and writing a resistance, one can update the resistance on chip. Fewer read and write operations reduces the risk of a read/write current being applied during a read/write, thus reducing the probability of a fault.

Resilience of Convolutional Neural Networks (CNN) is analyzed in [17]. It was observed in [12], [18], [19] that errors in some neurons are more critical than others. For example, an error in an output neuron affects only that output, while an error to an early neuron in a small layer will propagate through most of the rest of the network. While one can analyze a network for critical neurons and protect against faults (correcting faults when they occur), this can often be expensive or cumbersome. The authors propose a two step resilience optimization scheme that prevents major faults from occurring. The first component optimizes the architecture. It was noted in [19] that the resilience of a neuron is proportional to the number of neurons in the same layer. However, adding many new neurons can increase cost and energy efficiency to train the network. Hence, they propose adding neurons only to specific bottleneck layers to protect against significant faults in a few critical neurons. The second component optimizes the features. A measure is introduced to identify which features are more critical than others, where the criticality score depends on the weights between layers. Weights are adjusted to normalize the criticality score layer to layer. The network is then re-trained, as changing the weights affects accuracy. The re-trained network retains initial accuracy, but the criticality scores between layers are closer to normalized. The procedure is iterated until the normalization step reduces the accuracy within some small accepted tolerance. Hence, no one feature is more critical than any other. These methods are demonstrated on pre-trained networks which predict MNIST, CIFAR-10, and ILSVRC benchmarks, and the worst case failure rates for a single bit flip for each network are drastically reduced.

Many fault tolerant techniques for DNNs require large overhead and redundancy, while many applications favor low energy use and low latency. In [20], a particular Neural Architecture Search (NAS) is introduced. Specifically, a set of objective functions including error resiliance, energy consumption, latency, and bandwidth based solely on architecture are introduced. Multi-objective NAS is performed with an evolutionary algorithm which is an extension of LEMONADE [21]. Notice that the objective functions depend only on the architecture, and so expensive training steps are not necessary. The networks produced via the multi-objective search are shown to be fault tolerant when evaluated with random bit flip errors. Finally, with many DNNs the training steps and final weights are computed with floating point precision, often times low energy applications (such as cell phones, edge devices, etc.) perform better with 8 bit representations of weights. Converting the floating point weights to a coarser binary representation is known as *quantization*. Two quantization techniques are introduced, and on one set of networks one of the quantized networks is shown to be more error resilient than the other. In particular, fault tolerances are not quantization-invariant, and hence one should consider which quantization method is being used when fault resilience is important.

III. Methods

A. Algorithms

In this work we analyze the resiliency properties of SNNs trained using a variety of different training algorithms. In particular, the four different algorithms we study are a liquid state machine approach, a back-propagation-based approach for training deep neural networks with binary communication appropriate for neuromorphic deployment called Whetstone [22], a back-propagation-based approach for training deep spiking neural networks called SLAYER [23], and an evolutionary optimization-based approach called EONS [24].

Reservoir computing or liquid state machine approaches are often targeted towards time series data classification tasks. For our reservoir computing approach, we randomly create a spiking neural network with 100 hidden neurons, 20 output neurons and approximately 500 synapses. We use a spike-count encoding approach as described in [25], with a maximum of 10 spikes applied over time for each input value. We simulate activity in the network for 1000 time steps for each input value. A vector of the counts of spikes from each output neuron is collected and passed to a linear regression classifier from the scikit-learn Python module, which is the readout layer for our reservoir computing approach.

The second algorithm that we use in this work is Whetstone [22]. Whetstone is an approach that trains deep neural networks with binary communication. These types of networks can be implemented on spiking neuromorphic hardware because a binary communication approach is a simplified version of spikes (i.e., one that does not take into account the timing of spikes as part of the computation). Whetstone trains networks with binary communication by gradually "sharpening" the activation functions of neurons from differentiable functions like ReLUs to non-differentiable threshold functions that are more appropriate for deployment on spiking neuromorphic hardware. Whetstone has primarily been used to train networks for image recognition, for datasets such as MNIST and CIFAR10. We use Whetstone's adaptive scheduling for the sharpening process. We use 10-hot encoding for the outputs, as using multiple output neurons for a single output class is recommended in [22] in order to avoid Whetstone zeroing out an entire class over the course of training. We use a three layer, fully-connected network with 20 neurons in the hidden layer. However, it is worth noting that by optimizing the network structure (number of layers, number of hidden neurons per layer), better performance may be achieved.

The third algorithm we use in this work is Spike LAYer Error Reassignment (SLAYER) [23]. SLAYER is a back-propagation-like approach that not only trains the weights of the spiking neural network, but also trains the delays in the network. As such, SLAYER has primarily focused on datasets with temporal components, including audio signals, spike trains, and dynamic vision sensor (DVS) data. In this work, we convert our non-time series data into spike trains for input into SLAYER, as we do for both the liquid state machine and EONS algorithms. Unlike the other algorithms, we have hand-tuned the SLAYER hyperparameters such as τ, θ, and ρ, to perform well on the datasets we use here. We have done this for the SLAYER approach and not the others because SLAYER fails to train for the task at all without tuning those parameters. As in Whetstone, we use a three layer, fully-connected network with 20 neurons in the hidden layer.

The final algorithm that we use for training is Evolutionary Optimization for Neuromorphic Systems or EONS [24]. EONS is a genetic or evolutionary algorithm-based approach for determining both the structure and parameters of an SNN for neuromorphic deployment. EONS tends to produce relatively small (10s of neurons) and relatively sparse (10s to 100s of synapses) SNNs that are highly recurrent. EONS has been primarily targeted towards control applications, such as those described in [26]. As in the reservoir computing approach and the SLAYER approach, we use spike-count encoding to translate the data from the datasets into spikes. It is worth noting that we do not tune the parameters of EONS to perform well on this task and simply utilize the default parameters. Tuning either the EONS parameters or the input encoding parameters for networks trained using EONS can have significant performance impacts [25].

B. Simulating Failures

Each of the different algorithms described above can produce radically different network structures. For example, we use structures typical of traditional artificial neural networks for the Whetstone and SLAYER algorithms, i.e., networks organized into layers with full connectivity between the layers. Both reservoir and EONS generated networks are relatively sparse and can have recurrent connections. As a result, the size of the networks are often radically different, with SLAYER and Whetstone networks having hundreds to thousands of synapses, reservoirs typically having hundreds of synapses and EONS-generated networks typically having tens of synapses.

Because the networks vary so greatly in size and because we would expect synaptic devices to fail with some probability, we evaluate different failure rates $f_r \in \{0.1, 0.2, 0.3, ..., 0.9\}$. For failure rate f_r, we zero out or remove $f_r \times s_{net}$ synapses from the network, where s_{net} is the number of synapses in the originally trained network. We randomly choose the synapses to zero out for each failure rate individually, so the synapses removed for failure rate 0.1 may be entirely distinct from the synapses removed for failure rate 0.2, etc.

It is worth noting that most of the background and related work in Section II focuses on either a failures for a particular type of hardware (e.g., a bit flip or a memristor fault) or a particular type of algorithm or neural network type (e.g., convolutional neural networks). In this work, we evaluate a single failure type, full synapse failure, that can occur on many different types of neuromorphic hardware, especially those where the synapses are implemented using experimental devices. The intent behind choosing this type of failure is so that we can understand the broader implications of large-scale failures on the performance of SNNs on different types of neuromorphic hardware. Moreover, we analyze the implications of this failure type on a variety of algorithms, as we expect that different algorithms will be utilized for different applications even on the same neuromorphic hardware platform.

C. Datasets and Experimental Setup

Because the four algorithms we are studying target vastly different types of applications, we narrow our focus to classification tasks that are simple enough to be amenable to all of

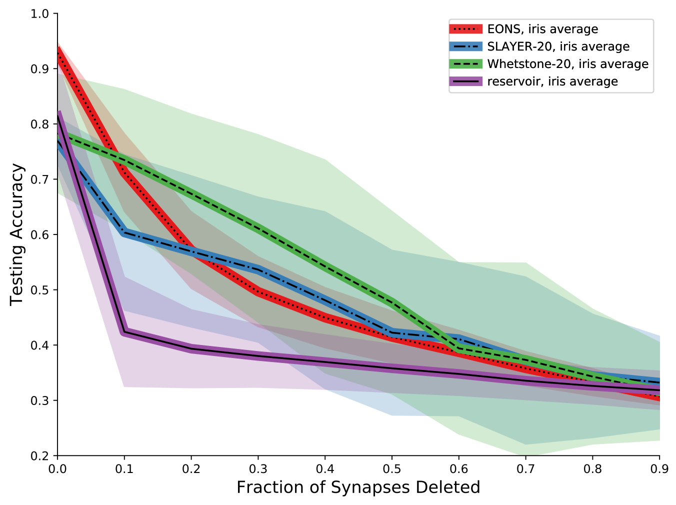

Fig. 1. Iris results. The mean performance across all 100 networks for each algorithm is shown in the bolded line and one standard deviation is shown to visualize the variance in performance for that algorithm.

the algorithms as a baseline comparison, but also give enough variation in performance to understand the impact of failures on the performance of networks produced by these algorithms. In particular, we use three of scikit-learn's toy datasets: iris, wine, and breast cancer. We use a two-thirds/one-third split on the train/test data for each dataset, and the same train/test split it used across all algorithms and all tests. A brief summary of the characteristics of these datasets is given in Table I.

TABLE I
DATASET CHARACTERISTICS

	Iris	**Wine**	**Cancer**
Features (Inputs)	4	13	30
Classes (Outputs)	3	3	2
Total Samples	150	178	569

For each of the algorithms, we train 100 distinct spiking neural networks. These networks are different across the 100 tests because of variations in the initial weights (for Whetstone and SLAYER), reservoir initialization (for the liquid state machine approach), and different initial populations (for the EONS approach).

IV. Resiliency Results

Figures 1, 2, and 3 give the results for the iris, wine, and cancer datasets, respectively. There are a few key observations to be made from these results. The first observation is that the overall trends of how the different algorithms perform holds across all three datasets. That is, though the best performing approach can differ across datasets, the general shape of the resilience "curves" for each of the algorithms is the same across each dataset. For example, the Whetstone resilience curves are more convex across all three datasets, while the EONS curves are more concave.

Second, it is clear that the reservoir computing approach has very poor resiliency characteristics across all three datasets.

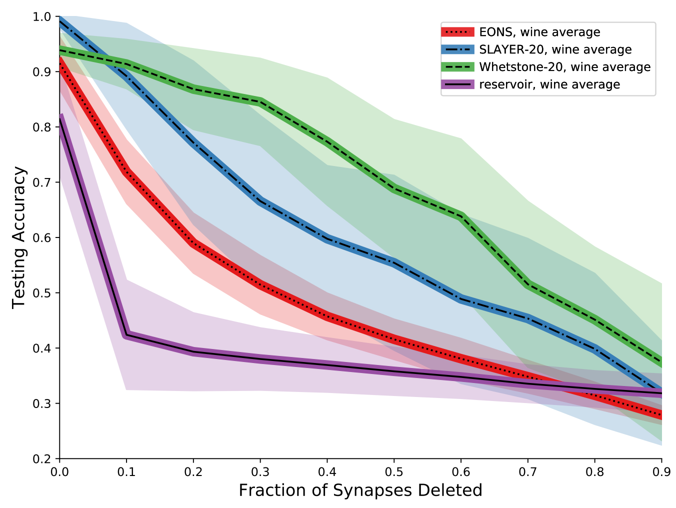

Fig. 2. Wine results. The mean performance across all 100 networks for each algorithm is shown in the bolded line and one standard deviation is shown to visualize the variance in performance for that algorithm.

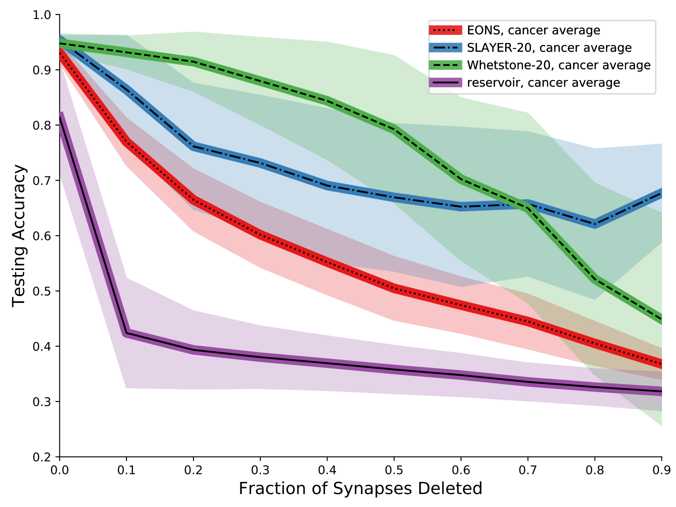

Fig. 3. Breast cancer results. The mean performance across all 100 networks for each algorithm is shown in the bolded line and one standard deviation is shown to visualize the variance in performance for that algorithm.

That is, with even a failure rate of 0.1, the performance of SNNs in reservoir approaches are significantly hurt. The other approaches more gradually become worse over time, and all of the approaches drop to approximately "guessing" behavior at high levels of failures for each dataset (guessing becomes approximately 33% for both iris and wine and approximately 50% for the cancer dataset), except for SLAYER on the cancer dataset, which still maintains a performance of around 70 percent.

In order to compare the performance of different algorithms and different networks with respect to resiliency and overall performance of the network, we define a metric that gives information about both the performance of the network, as well as its performance in the presence of failure, which we call $g_{f_r}(net)$. We calculate this metric as the area under the resiliency curves up to a failure rate of f_r, such as those plotted in Figures 1 - 3. Since we evaluate failure rates for $f_r \in \{0, 0.1, 0.2, ..., 0.9\}$, we approximate the area under the curve as follows:

$$g_{f_r}(net) = \frac{0.1}{f_r} \sum_{f \in \{0.1,...,f_r\}} \frac{1}{2} \left(a_f(net) + a_{f-0.1}(net)\right) \quad (1)$$

where $a_f(net)$ is the accuracy of given network at failure rate f. This value is higher for networks that have both high initial performance on the testing set, as well as high performance in the presence of failures. Using this metric, we can find the best performing, most resilient networks for each of the four algorithms and compare the performance of those different approaches directly. We calculate $g_{0.3}(net)$ for all of the networks trained across each of the four algorithms and find the networks with the highest value for each of the four algorithms.[1] Table II gives a summary of the characteristics of each of these four networks. There are a few items of interest to note in this table. First, the performance of EONS and reservoir computing generated networks tends to be monotonically decreasing as failures occur, while the same is *not* true of Whetstone and SLAYER networks. In fact, some failures in the network can improve the performance of the network in some cases (e.g., the starting accuracy of the SLAYER network on the wine dataset and the Whetstone network on the cancer dataset is lower than the accuracies of these networks in the presence of failures). It is known that sparsifying deep networks can improve the performance in some cases [27], and it appears that the same is true in the case of deep learning-style SNNs.

The second thing to note from Table II is that the size of these networks is radically different across the different algorithms. For example, the EONS networks have an order of magnitude fewer synapses than any of the other approaches. Practically, this means that for a task like the wine classification task, a failure rate of 0.3 for the EONS network results in a network with only 39 synapses, while a failure rate of 0.3 for the SLAYER network results in a network that still has 224 synapses. Moreover, in all cases, the Whetstone networks at a failure rate of 0.9 have more synapses than the *original* EONS generated network for each of the datasets. It is clear that the size of the network has an impact on the resiliency. However, in the case of neuromorphic systems, larger networks also require larger physical hardware and more power to implement them. On real-world applications, there are likely to be tradeoffs in accuracy, network size, power usage, and resiliency that will have to be made.

[1]We chose to focus on a failure rate of 0.3 because we believe it is a reasonable estimate for a relatively large-scale failure in a physical neuromorphic hardware system. This metric could easily be applied for any failure rate, however. We include results in the rest of this work for up to a failure rate of 0.9 to show what would happen in the event of catastrophic failure.

TABLE II
BEST NETWORK STATISTICS

		Accuracy	f_r = 0.1	$f_r = 0.2$	$f_r = 0.3$	$\delta_{0.3}$	Neurons	Synapses
Iris	**Reservoir**	94.0	76.2	59.0	55.1	0.70	124	494
	Whetstone	92.0	88.0	92.0	84.0	0.91	54	680
	SLAYER	84.0	84.0	64.0	64.0	0.74	27	140
	EONS	94.0	89.4	75.2	62.4	0.81	20	19
Wine	**Reservoir**	94.0	76.3	59.0	55.1	0.70	124	494
	Whetstone	98.3	96.6	94.9	93.2	0.96	63	860
	SLAYER	96.6	100.0	88.1	91.5	0.94	36	320
	EONS	96.6	82.4	70.3	66.0	0.78	39	56
Cancer	**Reservoir**	94.0	76.3	59.0	55.1	0.70	124	494
	Whetstone	96.8	97.3	97.3	96.3	0.96	70	1000
	SLAYER	96.3	96.3	93.6	87.2	0.94	52	640
	EONS	94.1	86.7	78.8	74.4	0.83	61	67

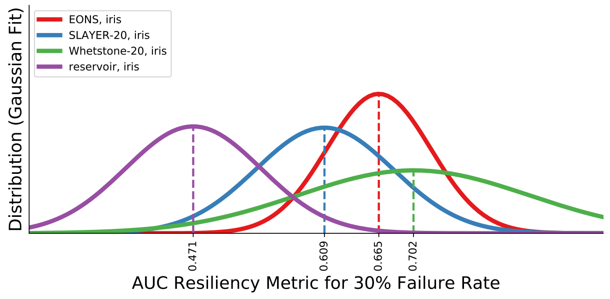

Fig. 4. Gaussian fitted distribution of $g_{0.3}$ for each of the four algorithms on the iris dataset.

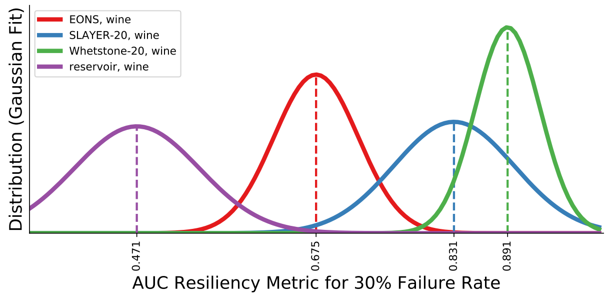

Fig. 5. Gaussian fitted distribution of $g_{0.3}$ for each of the four algorithms on the wine dataset.

Figures 4 - 6 give Gaussian-fitted distributions for the area-under-the-curve metric $g_{0.3}$ for each of the four algorithms. These distributions give a summary of the overall resiliency and performance characteristics of the four different algorithms on each of the datasets. These figures give similar information as those in Figures 1 through 3, but they also elucidate a few other resiliency features. First, as they summarize the statistics across all 100 networks for each dataset, we see that EONS networks have relatively consistent resiliency behavior (across the 100 networks) on all three datasets, whereas Whetstone is most consistent on the wine and cancer datasets and is least consistent on the iris dataset. This indicates that the resiliency performance of the algorithm can greatly depend on the dataset for some approaches. Second, although the Whetstone and SLAYER approaches both use back-propagation-style approaches, it is clear that Whetstone gives more consistent resiliency results. We speculate that this is likely due to the n-hot encoding for output nodes that is used for Whetstone. This output encoding approach is used for Whetstone to prevent dead output neurons during training, but it appears that it can also help deal with failures during inference.

Finally, it is worth that in calculating the performance at each failure rate, we only evaulate one set of potential failures. Figure 7 shows each of the failure curves for each algorithm on the cancer dataset. As we can see in these figures, the performance of Whetstone and SLAYER networks appears to depend dramatically on the synapses selected to fail, as

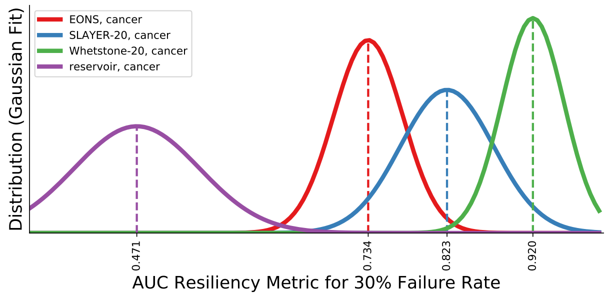

Fig. 6. Gaussian fitted distribution of $g_{0.3}$ for each of the four algorithms on the cancer dataset.

indicated by the vacillating behavior of the failure curves. This is not surprising given previous work that has found that certain neurons are critical to performance in convolutional neural networks (as noted in Section II). In future work, we intend to more fully explore the resiliency of these networks by systemically introducing failures across various combinations of synapses, rather than selecting random sets of synapses to fail as we do in this work.

V. Training for Resiliency

In the previous sections, we demonstrate the performance of different algorithms in the face of synaptic failures. None of these algorithms were adapted to account for future failures. Much of the work in producing more fault tolerant neural networks requires some sort of re-training step, which may

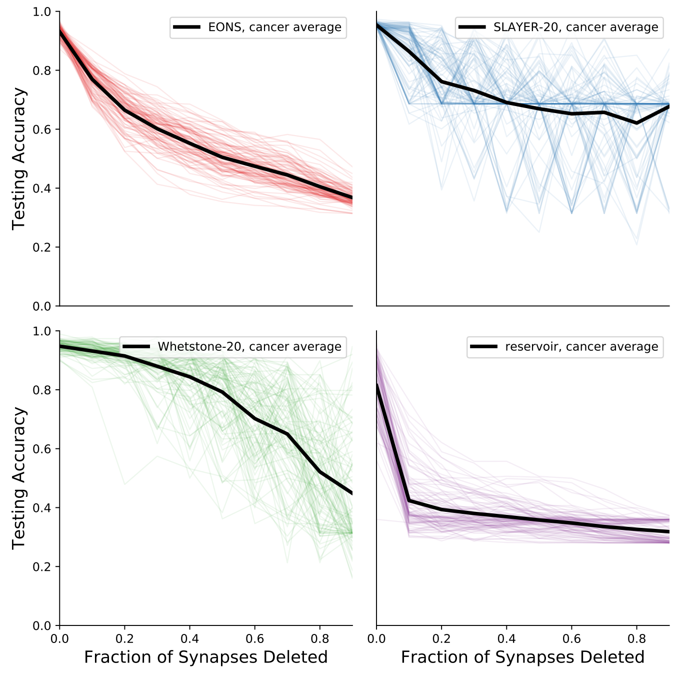

Fig. 7. Failure curves for each of the trained networks on the cancer dataset.

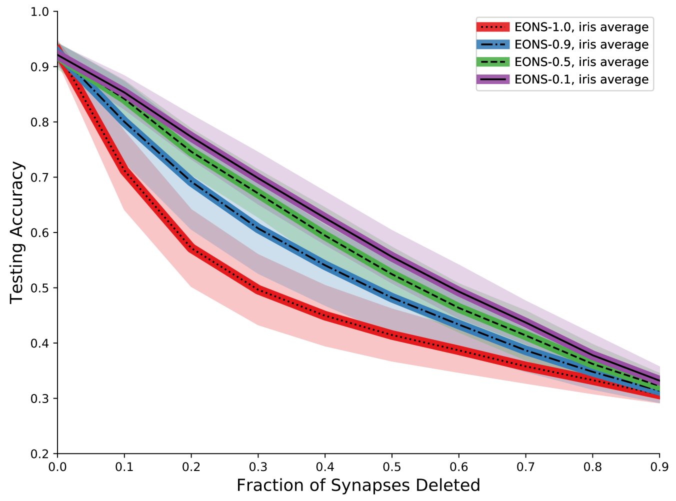

Fig. 8. Iris results for EONS trained with the fitness function described in Equation 2 for varying values of ω. The original EONS results ($\omega = 1$) are also shown in the figure and are the same as those shown in Figure 1.

not be possible when the networks are deployed in real-world environments. We propose instead to adapt the original training algorithms in order to produce initially networks that are more resilient to errors.

A. *Optimizing for Resiliency*

Here, we focus on the EONS algorithm, though, in Section VI, we discuss how this work may be extended for the other training approaches. The EONS approach, as described in Section III-A, is based on evolutionary optimization. In the original EONS algorithm, the structure and parameters of SNNs are optimized to maximize the performance of the network on the application, which is defined as the fitness function in EONS. In the case of classification tasks, the fitness function is typically defined as accuracy on the training set. However, as has been shown in previous work [28], the EONS fitness function can be adapted to include other objectives of interest, including minimizing network size or resiliency to small corruptions in the network. In [28], corruptions in the network were limited to small perturbations in the synapse weight values, which might be a result of a bit-flip in a digital system or drift in an analog system. In this work, we extend that approach by allowing for corruptions to include multiple complete synapse failures.

Our approach is as follows. Rather than defining the fitness function $f(net)$ for EONS as the accuracy of the network $a(net)$ on the training set as was used in the previous baseline results, we also create n corrupted versions of the network we are evaluating. We impose a failure rate of f_r on each of these n networks net_i, randomly selecting the synapses to fail for each type of network. Then, the fitness function is defined as follows:

$$f(net) = \omega a(net) + \frac{1-\omega}{n}\sum_{i=1}^{n} a(net_i) \tag{2}$$

In this equation, ω is a weighting factor that can bias the fitness function to focus more attention on performance on the original network or more on the performance of the corrupted networks. When using this function, EONS requires both the original network to perform well on the task ***and*** the corrupted networks to perform well on the task. The intended goal is to produce networks that will perform better in the face of failures than a network trained without factoring in potential for failures or corruptions in the network. In this work, we use $n = 10$, but we evaluate different values for ω to understand the effect of that parameter on performance.

B. *Results*

The results for EONS with the fitness function given in Equation 2 are shown in Figures 8 through 10 for varying values of ω. Note that for $\omega = 1$, EONS with the fitness function in Equation 2 is equivalent to the original EONS fitness function. As such, the results for EONS with $\omega = 1$ are the same as those shown in Figures 1 through 3.

As can be seen in these figures, by leveraging $\omega \neq 1$, we can improve the resiliency of networks produced by EONS. Moreover, we can see that the value of $\omega = 0.1$ gives the best results of the different values of ω evaluated in terms of resiliency. It is worth noting that using $\omega \neq 1$ can also result in slightly degraded performance of the original network; however, with any failures, all of the $\omega \neq 1$ networks outperform the original EONS approach. The statistics for the best performing networks for $\omega = 0.1$ are compared with the best performing original EONS networks in Table III. As we

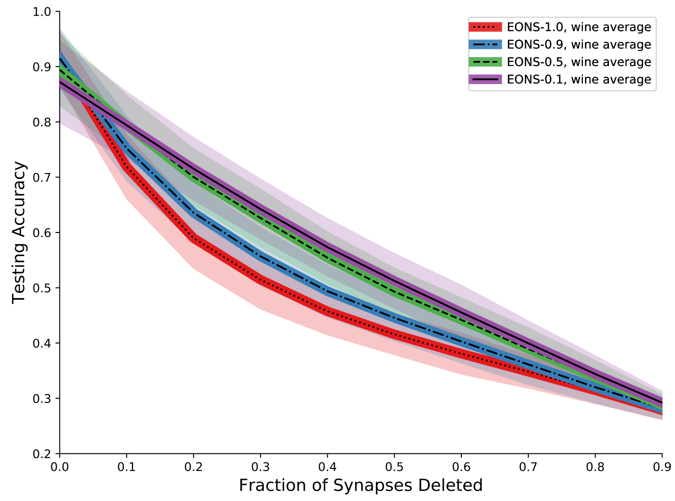

Fig. 9. Wine results for EONS trained with the fitness function described in Equation 2 for varying values of ω. The original EONS results ($\omega = 1$) are also shown in the figure and are the same as those shown in Figure 2.

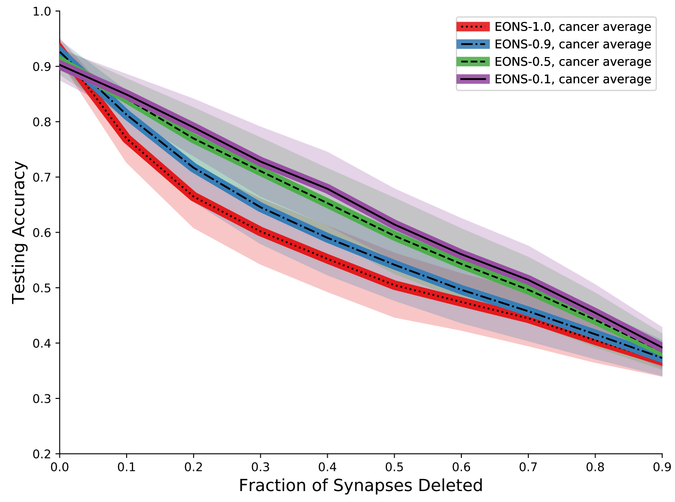

Fig. 10. Cancer results for EONS trained with the fitness function described in equation 2 for varying values of ω. The original EONS results ($\omega = 1$) are also shown in the figure and are the same as those shown in Figure 3.

can see from this table, the networks produced using EONS $\omega = 0.1$ have better resiliency characteristics, comparable testing accuracy with no failures and comparable network size to those produced using the original EONS approach.

We compare the performance of EONS with $\omega = 0.1$ to the original EONS, Whetstone, and SLAYER results in Section IV in Figures 11 through 13. (We omit the reservoir results here because of their poor performance.) As we can see in these figures, the $\omega = 0.1$ EONS approach gives much better performance overall than the original EONS approach. In the case of the iris dataset, the new EONS approach has much better performance than all of the approaches, whereas in the case of the cancer dataset, the approach essentially meets the performance of SLAYER, but is more consistent in its

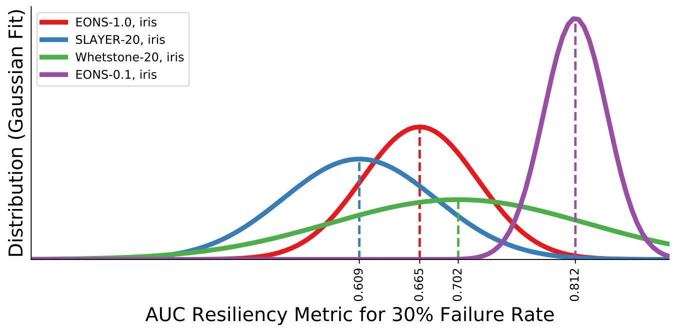

Fig. 11. Gaussian fitted distributions of $g_{0.3}$ for the original EONS approach, SLAYER, and Whetstone, alongside the EONS approach with $\omega = 0.1$ on the iris dataset.

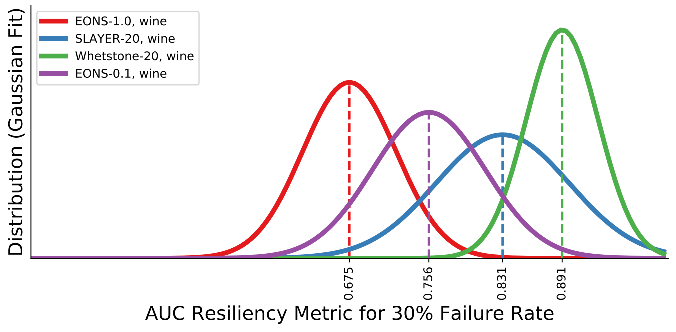

Fig. 12. Gaussian fitted distributions of $g_{0.3}$ for the original EONS approach, SLAYER, and Whetstone, alongside the EONS approach with $\omega = 0.1$ on the wine dataset.

performance than SLAYER. However, in the case of the wine dataset (Figure 12), the new EONS approach still lags with respect to SLAYER and Whetstone, indicating that there is still room for improvement. As noted above, the EONS networks are still at least an order of magnitude smaller than the SLAYER and Whetstone networks. As such, this approach can provide a way to produce well-performing, resilient spiking neural networks that are also size, area, and energy efficient neuromorphic deployments.

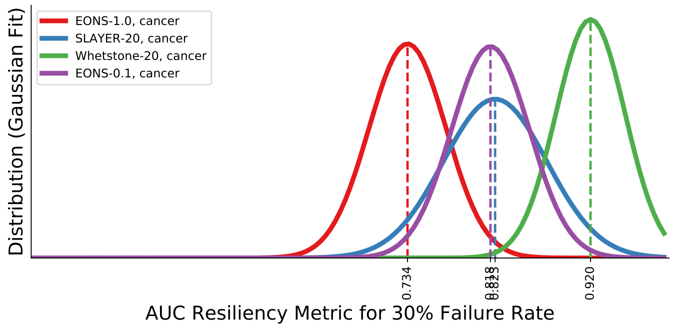

Fig. 13. Gaussian fitted distributions of $g_{0.3}$ for the original EONS approach, SLAYER, and Whetstone, alongside the EONS approach with $\omega = 0.1$ on the cancer dataset.

TABLE III
TRAINING FOR RESILIENCY BEST NETWORK STATISTICS

		Accuracy	f_r = 0.1	$f_r = 0.2$	$f_r = 0.3$	$\delta_{0.3}$	**Neurons**	**Synapses**
Iris	**EONS**	94.0	89.4	75.2	62.4	0.81	20	19
	EONS $\omega = 0.1$	94.0	91.8	86.8	76.0	0.88	24	29
Wine	**EONS**	96.6	82.4	70.3	66.0	0.78	39	56
	EONS $\omega = 0.1$	100.0	88.6	79.7	71.7	0.85	55	49
Cancer	**EONS**	94.1	86.7	78.8	74.4	0.83	61	67
	EONS $\omega = 0.1$	94.7	92.7	89.4	82.5	0.90	74	49

Though there are key advantages in producing more resilient networks by using this new fitness function for EONS, there are also consequences to utilizing this approach. The major issue associated with this approach is that it takes significantly longer to train using this fitness function. In particular, using the original EONS approach, for each fitness function we are only evaluating the accuracy for one network. When using the new approach, the accuracy must be calculated for $n+1$ networks during the fitness function evaluation. Because this accuracy calculation requires simulating the network's behavior on each of the training data instances, it is the most time intensive part of the calculation. As such, it requires approximately $n + 1$ times as long to train the same number of generations using this new fitness function as the original EONS fitness function. Also, as noted above, the testing accuracy in the absence of failures can be slightly reduced using the new approach as well.

VI. DISCUSSION AND CONCLUSION

In this work, we analyze the resilience characteristics of four different types of algorithms for training spiking neural networks for neuromorphic systems. We see that these different algorithms have very different performance characteristics with respect to resiliency. For example, liquid state machine approaches have very poor resilience properties, while other approaches can tolerate some failures and still operate fairly well. As such, though robustness, resiliency, and fault tolerance are common cited reasons for utilizing neuromorphic systems [1], it is clear that this characteristic can depend significantly on the type of algorithm used. We show that the two back-propagation algorithms that use fully connected layers have the best resiliency.

In Section V, we describe an approach for training networks that are more resilient to synapse failures using the EONS framework, an evolutionary optimization-based approach. We adapt the fitness function to include the evaluation of each network with some number of failed synapses as part of the overall performance of the network. As we showed in Section V-B, by adding this as part of the training process, we can improve the resilience of the network to failures. This sort of augmentation is more difficult for other types of algorithms, where there is not an equivalent of a fitness function. However, evolutionary-based approaches have been used as an optimization "outer-loop" to define hyperparameters and network topologies of both liquid state machines [29] and traditional, deep learning-style networks like those used in Whetstone and SLAYER [30]. In future work, we intend to apply the same sort of fitness function adaptation that we use here to optimize for resilience in genetic algorithm or evolutionary algorithm-based training for liquid state machines and the structure and/or hyperparameters of back-propagation-based training approaches.

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, and in part by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC.

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] G. W. Burr, R. M. Shelby, A. Sebastian, S. Kim, S. Kim, S. Sidler, K. Virwani, M. Ishii, P. Narayanan, A. Fumarola *et al.*, "Neuromorphic computing using non-volatile memory," *Advances in Physics: X*, vol. 2, no. 1, pp. 89–124, 2017.

[3] 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 (JETCAS)*, vol. 8, no. 1, pp. 125–136, March 2018.

[4] R. Weiss, J. S. Najem, M. S. Hasan, C. D. Schuman, A. Belianinov, C. P. Collier, S. A. Sarles, and G. S. Rose, "A soft-matter biomolecular memristor synapse for neuromorphic systems," in *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 2018, pp. 1–4.

[5] 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 *2018 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2018, pp. 1–7.

[6] C. Liu, M. Hu, J. P. Strachan, and H. Li, "Rescuing memristor-based neuromorphic design with high defects," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2017, pp. 1–6.

[7] N. C. Hammadi and H. Ito, "A learning algorithm for fault tolerant feedforward neural networks," *IEICE TRANSACTIONS on Information and Systems*, vol. 80, no. 1, pp. 21–27, 1997.

[8] Z.-H. Zhou and S.-F. Chen, "Evolving fault-tolerant neural networks," *Neural Computing & Applications*, vol. 11, no. 3-4, pp. 156–160, 2003.

[9] B. E. Segee and M. J. Carter, "Fault tolerance of pruned multilayer networks," in *IJCNN-91-Seattle International Joint Conference on Neural Networks*, vol. 2. IEEE, 1991, pp. 447–452.

[10] P. Chandra and Y. Singh, "Fault tolerance of feedforward artificial neural networks-a framework of study," in *Proceedings of the International Joint Conference on Neural Networks, 2003.*, vol. 1. IEEE, 2003, pp. 489–494.

[11] C.-T. Chin, K. Mehrotra, C. K. Mohan, and S. Rankat, "Training techniques to obtain fault-tolerant neural networks," in *Proceedings of IEEE 24th International Symposium on Fault-Tolerant Computing*. IEEE, 1994, pp. 360–369.

[12] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G.-Y. Wei, "Ares: A framework for quantifying the resilience of deep neural networks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[13] M. Lee, K. Hwang, and W. Sung, "Fault tolerance analysis of digital feed-forward deep neural networks," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 5031–5035.

[14] C. Yakopcic, R. Hasan, and T. M. Taha, "Tolerance to defective memristors in a neuromorphic learning circuit," in *NAECON 2014-IEEE National Aerospace and Electronics Conference*. IEEE, 2014, pp. 243–249.

[15] M. Liu, L. Xia, Y. Wang, and K. Chakrabarty, "Design of fault-tolerant neuromorphic computing systems," in *2018 IEEE 23rd European Test Symposium (ETS)*. IEEE, 2018, pp. 1–9.

[16] ——, "Fault tolerance in neuromorphic computing systems," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. ACM, 2019, pp. 216–223.

[17] C. Schorn, A. Guntoro, and G. Ascheid, "An efficient bit-flip resilience optimization method for deep neural networks," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1507–1512.

[18] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2017, p. 8.

[19] C. Schorn, A. Guntoro, and G. Ascheid, "Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 979–984.

[20] C. Schorn, T. Elsken, S. Vogel, A. Runge, A. Guntoro, and G. Ascheid, "Automated design of error-resilient and hardware-efficient deep neural networks," *arXiv preprint arXiv:1909.13844*, 2019.

[21] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," *arXiv preprint arXiv:1804.09081*, 2018.

[22] 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.

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

[24] 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.

[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] 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. [Online]. Available: http://neuromorphic.eecs.utk.edu/raw/files/publications/2019-Plank-Gomac.pdf

[27] X. Sun, X. Ren, S. Ma, and H. Wang, "meprop: Sparsified back propagation for accelerated deep learning with reduced overfitting," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 3299–3308.

[28] 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.

[29] J. J. Reynolds, J. S. Plank, and C. D. Schuman, "Intelligent reservoir generation for liquid state machines using evolutionary optimization," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[30] R. M. Patton, J. T. Johnston, S. R. Young, C. D. Schuman, D. D. March, T. E. Potok, D. C. Rose, S.-H. Lim, T. P. Karnowski, M. A. Ziatdinov *et al.*, "167-pflops deep learning for electron microscopy: from learning physics to atomic manipulation," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*. IEEE Press, 2018, p. 50.