Understanding Selection and Diversity for Evolution of Spiking Recurrent Neural Networks

Catherine D. Schuman
Computational Data Analytics
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831–6085
Email: schumancd@ornl.gov

Grant Bruer, Aaron Young, Mark Dean, and James S. Plank
Department of Electrical Engineering and Computer Science
University of Tennessee
Knoxville, Tennessee 37996
Email: [gbruer, ayoung48, markdean, jplank]@utk.edu

***Abstract*—Evolutionary optimization or genetic algorithms have been used to optimize a variety of neural network types, including spiking recurrent neural networks, and are attractive for many reasons. However, a key impediment to their widespread use is the potential for slow training times and failure to converge to a good fitness value in a reasonable amount of time. In this work, we evaluate the effect of different selection algorithms on the performance of an evolutionary optimization method for designing spiking recurrent neural networks, including those that are meant to be deployed in a neuromorphic system. We propose a selection approach that utilizes a richer understanding of the fitness of an individual network to inform the selection process and to promote diversity in the population. We show that including this feature can provide a significant increase in performance over utilizing a standard selection approach.**

I. Introduction

Spiking recurrent neural networks (SRNNs) are known to be a theoretically powerful computational tool [1], [2]. Moreover, they are an increasingly popular computational model implemented in neuromorphic hardware [3]. Perhaps the most intriguing question associated with these types of neural networks is how to design and train them effectively to solve real problems. A variety of approaches have been proposed for determining the weights of spiking neural networks, including back-propagation [4], [5] and spike-timing dependent plasticity or other biologically-inspired plasticity mechanisms [6], [7], but those approaches do not give guidance on determining various aspects of the network (e.g., network topology or delays), nor are they necessarily customized for neuromorphic hardware.

One approach for training neural networks is evolutionary optimization or evolutionary algorithms, an approach that is sometimes called neuroevolution [8], [9]. There are a variety of attractive reasons for using evolutionary approaches for training spiking recurrent neural networks: there are no restrictions on the topology of the networks or functionality of the network, they are applicable to a wide variety of applications, they can operate within the characteristics and constraints of neuromorphic systems, and they can define all aspects of the network, including network topology and parameters such as synaptic weights and delays. Evolutionary optimization and genetic algorithms have not been widely applied to spiking neural network in the past because of a few key issues. First and most importantly, they can be slow to converge for some applications and may get stuck in local minima, further delaying convergence, and second, it can be difficult to define how the evolutionary optimization or genetic algorithm will operate on the particular spiking neural network structure.

In this work, we seek to address the first issue and improve the performance of an evolutionary optimization method as applied to SRNNs for neuromorphic systems by examining the role of selection algorithms and population diversity in training performance, with the goal of finding the selection approach or approaches that are most likely to help with training convergence. We restrict our view to the selection aspect of EO because any improvements or alterations that occur in defining how selection operates are applicable to any application and any neuromorphic model. Furthermore, any insight gained in how selection should be implemented can be extended to other neuroevolution methods as well. In the following sections, we discuss previous work done on evolutionary optimization for SRNNs and note both the key advantages and disadvantages of utilizing them for SRNNs and neuromorphic networks. In Section III, we briefly describe the two SRNN models used in this work, noting the key differences in the two models that can result in significant differences in training performance. In Section IV, we discuss our overall evolutionary optimization approach for training SRNNs and note the role of selection and diversity in that process. We present the results for both SRNN models on two different applications in Section V, specifically noting the difference in performance for different selection algorithms and how diversity in the population and final performance are correlated. We also present an approach for utilizing a richer definition of fitness to improve the selection procedure. We conclude with discussion of the results and of potential future directions for this work.

II. Background and Related Work

Evolutionary optimization (EO) and/or genetic algorithm (GA) approaches for training neural networks have been used for the last three decades and are sometimes referred to as neuroevolution [10], [9]. EO and GAs have been used simply for weight training of fixed structures networks [11], for determining network topology and weights [12], [8], [13], or for topology or hyperparameters while traditional algorithms such as back-propagation are used for weight training [14], [15].

Spiking neural networks in general, and spiking recurrent neural networks (SRNNs) in particular, present an intriguing issue for training, as they add an additional dimension beyond topology and parameters to optimize in the form of the time dimension. There are a variety of proposed ways to train SRNNs, including back-propagation, spike timing dependent plasticity, and liquid state machines, but none of these methods define a satisfactory approach for determining all aspects of an SRNN. Back-propagation based methods will require certain restrictions on the topology for the network structure, but it is not clear how many layers, layer types (if utilizing convolutional neural network-based structures), number of neurons per layer, and connectivity between layers should be used for a given task. Certain topologies have been useful for utilizing STDP, but for general applications, it is not clear how to determine the topology. Reservoir computing or liquid state machines overcome this approach by providing certain rules for the structure of the reservoir and leaving the weights of the reservoir untrained [16], [17]. However, there are restricted use cases for reservoir computing and the size of the reservoir required can be quite large, which may not be suitable for some neuromorphic implementations. In addition to those methods, EO and GAs have been applied to SRNNs successfully in the past in a variety of works [18], [19], [20], [21]. EO and GAs have also been used to construct spiking neural networks and SRNNs for neuromorphic systems [21], [22], [23], [24].

There are a variety of reasons for using EO and GAs for designing SRNNs. EO and GAs provide the flexibility to optimize all aspects of SRNNs, which is a great advantage for these relatively complex models. They can optimize and customize the topology, parameters, and hyper-parameters of the network for both the particular application and for a particular neuromorphic architecture. For neuromorphic systems in particular, in addition to optimizing topology, parameters, and hyper-parameters of the SRNNs, the training method must work within the constraints of the neuromorphic device, which typically include restrictions on all aspects of the network. Yao notes several key properties of EO and GAs that make them useful for training neural networks, including: they do not rely on gradient information (which may be difficult or impossible to calculate), they can be applied to any neural network architecture or model, they are much less sensitive to initial conditions (though there is still sensitivity to the initialization), and they always search for global optima, though they may also get stuck in local optima [10]. EO and GAs are very flexible with respect to the applications to which they can be applied, because they rely on a fitness score rather than an error calculation. As such, they can be applied to design networks for control applications, where error information may not be readily available.

Though there are many advantages to utilizing EO and GAs for SRNN training, there are several issues that may prevent researchers from utilizing them. The first is that a representation of the network in the population must be defined, which is a non-trivial issue for spiking neural networks. We utilize a direct representation, which means that every aspect of the network is defined in the representation. As discussed further below, this means that there are typically a tremendously large of potential solution networks that are representable in the population. Moreover, the solution space itself is very complex, as tweaking of single elements in the representation can result in changes in topology or parameters that will radically affect the performance of the network. We have also observed the tendency for our EO approach to become stuck in a local minima because the population becomes dominated by one particular style of solution network, which is exacerbated by the competing conventions problem, in which many different networks are functionally but not structurally equivalent. Thus, though our network representations are distinct, they produce the same or very similar behavior, often resulting in a relatively homogeneous population. One way to mitigate this issue is to adjust the selection algorithms utilized for determining which networks in the population to select to produce offspring. Though selection algorithms themselves have been studied extensively with respect to EO and GAs as a whole [25], [26], it is not clear which selection algorithms will actually perform best in producing the complex network solutions required for evolving SRNNs, nor is it clear what the relationship is between population diversity and performance for this particular type use case of EO.

III. Spiking Recurrent Neural Networks

In this work, we examine selection approaches for evolutionary algorithms as applied to spiking recurrent neural networks. In particular, we examine the effect of selection and diversity on the performance of evolution-based training for spiking recurrent neural networks. Throughout, we utilize two spiking recurrent neural network frameworks: neuroscience-inspired dynamic architecture (NIDA) networks [27] and dynamic adaptive neural network array (DANNA) networks [28]. Both NIDA and DANNA are bounded spiking neural networks (there are a maximum number of neurons for each) that utilize integrate-and-fire neurons and synapses with delays and plasticity mechanisms. The key differences between NIDA and DANNA are that NIDA is a software-based spiking architecture and has fairly unlimited connectivity between neurons, while DANNA is a hardware-based spiking architecture that has relatively strict limits on network topology. As such, these two architecture types give a nice spectrum of how selection algorithms operate in both more restricted (DANNA) and more unrestricted (NIDA) environments.

IV. Evolutionary Optimization for Neuromorphic Systems (EONS)

Our evolutionary approach for designing spiking recurrent neural networks for neuromorphic implementation is called Evolutionary Optimization for Neuromorphic Systems (EONS). For any given instance of EONS, the goal is to produce a spiking recurrent neural network (SRNN) to be deployed on a particular neuromorphic implementation that performs well on a particular application. In particular, the goal is to design all aspects of the SRNN, including the topology (number of neurons and synapses and connectivity between them) and parameters of the network and its elements (weights of the synapses, threshold of the neurons, etc.) To achieve this, the first step of EONS is to initialize a population of networks. This is followed by evaluation, selection, and reproduction operations that operate on the population to produce a new population, which will replace the old one (Figure 1). The evaluation procedure of EONS is dependent upon the application for which a network is being optimized. There are many such applications, including those in control, classification and anomaly detection, and the way evaluation is done can vary significantly from application to application. Though there is innovation to be done in the evaluation, this innovation is likely limited to a relatively small subset of applications. The reproduction procedure depends on the particular neuromorphic model on which the network will be deployed; once again, any innovation to be made here will likely be relatively specific to the particular neuromorphic model or device. In previous works we have discussed a variety of applications and their associated evaluation procedures [29], [30], [31], as well as particular models and their associated reproduction operations [21]. In this work, we focus on the selection portion of the EONS method, which is generic to both the application and the neuromorphic model. Selection algorithms are used in genetic and evolutionary algorithms to produce a "survival of the fittest" effect. There are a variety of selection algorithms to choose from, some of which have one or more parameters. Though there has been work done to analyze the performance of different selection algorithms (as noted in Section II), it is not clear which ones will be most appropriate for EONS, because EONS is a non-traditional evolutionary approach, as described below. Since the selection procedure is application and model-agnostic, innovation in the selection algorithm has the potential to improve performance for a variety of applications, as well as a variety of neuromorphic models. Moreover, because we have implemented a software stack that allows for separate models and applications to be easily compiled with our full software stack [31], these improvements are immediately applicable to different models and applications.

EONS optimizes over a non-traditional genomic representation for members of the population. EONS utilizes a direct representation of the network, meaning that every parameter and aspect of the network is explicitly accounted for in the genome representation. For example, for one of the neuromorphic models explored in this work (NIDA), the network representation is that of a graph, where the nodes are the neurons and the edges are the synapses. EONS determines how many nodes and edges are required in the graph, as well as their associated parameters. Since we are concerned with neuromorphic hardware implementations, all of our neuromorphic models have physical limits on the maximum number of neurons and synapses that are allowed. In addition, every parameter has a fixed number of values that can be attained. However, there is still a tremendously large solution space over which EONS is searching to find a particular network. For example, for another neuromorphic model explored in this work (DANNA), each DANNA device has a fixed number of elements, where each element is set as unused, a neuron, or a synapse. As such for a DANNA with 100 elements, there are on the order of 3^{100} different topologies that can be attained (though some of those are not legal topologies). This number does not take into account the parameters that are associated with each element, each of which can take on a variety of values (up to 256 different values for some parameters). With this level of complexity, as well as the large number of local minima that we expect in the fitness landscape, it is not clear which selection algorithm will perform best in maintaining enough diversity in the population or which one will be most useful in mitigating getting stuck in local optima.

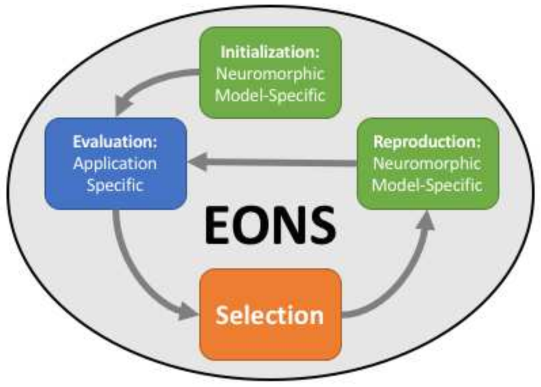


Fig. 1. Overall view of the operation of EONS. This work focuses on the selection portion of the EONS workflow.

V. Results

A. Applications

In this work, we focus our attention on two applications: the canonical pole balancing task that we have previously described in [27] and playing the Atari game Pong. For pole balancing, the SRNN is given information about the current state of the cart and pole: the cart's position and velocity and the pole's position and velocity. The output of the SRNN is calculated every 0.02 seconds, and gives a decision as to whether the cart should move to the left or to the right. As in many control applications, we evaluate the SRNN's ability on the task in several different test cases, which usually corresponds to different starting conditions. In the case of pole balancing, we evaluate six different starting conditions (different initial positions of the cart and the pole). We measure fitness by how many time steps the SRNN is able to successfully keep the pole from falling and the cart from hitting the two endpoints of the track. We cap the maximum number of time steps for each test case at 15,000. The fitness for each test case is calculated as follows:

$$f_i = \frac{\text{number of hits for test case } i}{15000} \tag{1}$$

The overall fitness for the SRNN for the pole balancing task is then:

$$f = \frac{\sum_{i=1}^{6} f_i}{6} \tag{2}$$

For Pong, the SRNN is given information about the current state of the game: position and velocity of the player's paddle, horizontal and vertical distance to the ball, horizontal and vertical velocity of the ball, and the position and velocity of the opponent's paddle. The output of the SRNN is calculated every 0.02 seconds, and gives a decision as to whether the player's paddle should be moved up or down or to stay in the same position. Similar to the pole balancing task, we evaluate the SRNN's ability on the task in multiple test cases. In the case of Pong, these different starting conditions are different starting angles of the ball. Unlike several other control applications that we have evaluated, including the pole balancing task, Flappy Bird [32], and robotic navigation [30], we have found that when training SRNNs for Pong, EONS is very likely to get stuck in a local optima. In particular, we observed that EONS was likely to optimize to do very well on one or two of the test cases we have selected, and perform relatively poorly on the other examples, indicating that the test cases actually require very different performance from the neuromorphic system and indicating that the problem is non-trivial for SRNNs (especially those that are restricted) to solve. As such, Pong served as an excellent test case for observing the effect of selection and diversity on overall performance of EONS. In the Pong application, we use four test cases. We measure fitness by how many times the player (the SRNN) is able to successfully hit the ball back to the "ideal" player, which calculates where the ball is going and moves its paddle to that spot accordingly. For each of the test cases, we cap the maximum number of hits at 55. The fitness for each test case is calculated as follows:

$$f_i = \frac{\text{number of hits for test case } i}{55} \tag{3}$$

Then the overall fitness for the SRNN is then:

$$f = \frac{\sum_{i=1}^{4} f_i}{4} \tag{4}$$

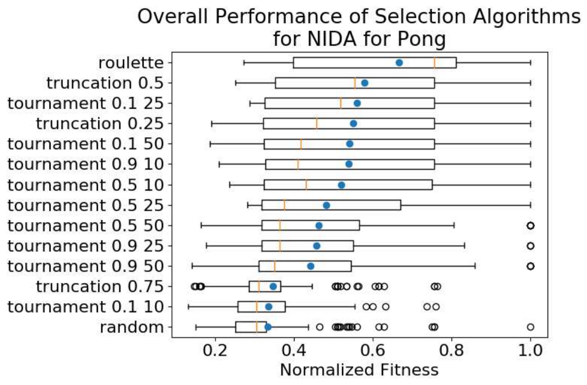
For both applications, in addition to the fitness value of f for each network, we also return to EONS the various f_i values. Initially, we however, we only utilize f as part of the training mechanism, though we track full fitness information.

B. Selection Algorithm Performance

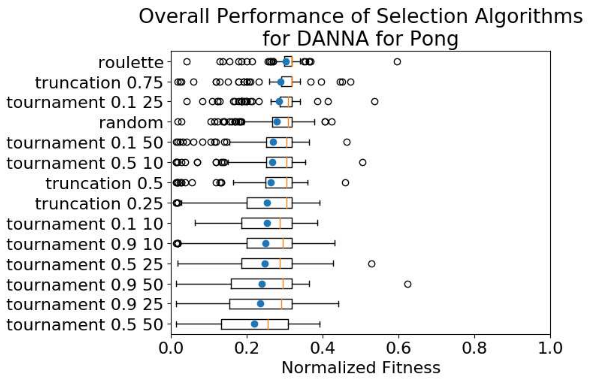
The first major question associated with selection and diversity for EONS is how different selection algorithms perform, both in terms of overall performance and in terms of diversity in the population. We examined four different types of selection algorithms: tournament, roulette, truncation, and random. Tournament selection has two parameters: n, the tournament size, and p, the likelihood that the best performing network in the population is chosen. In tournament selection, $0 < n \leq N$ networks are randomly chosen from the population, where N is the population size. The best performing network in the tournament is chosen with probability p, the second best network is chosen with probability $p(1-p)$, the third best network is chosen with probability $p(1-p)^2$, and so on. Roulette (or fitness proportionate) selection chooses networks to serve as parents based on their fitness functions; in particular, the higher the fitness value of a network with respect to other networks in the population, the more likely that network is to be chosen. Truncation selection has one parameter, a population fraction p, where $0 < p \leq 1$. In truncation selection, the top $p * N$ networks are chosen, and then a network from those $p * N$ networks is randomly chosen to serve as a parent. For random selection, a network is randomly selected from the population to serve as a parent. Random selection is included to provide a baseline of performance.

In all tests, we used a population size (N) of 100. In generating a child population from the previous generation's population, we include 10 random networks and the top 10 best networks from the previous population. We set the crossover and mutation rates to 0.9 each. Each run of EONS was allowed to train for 200 epochs or until a fitness value of 1 was reached. We include 10 random networks in each generation to help with diversity, but we also include the top 10 best networks to guarantee a non-decreasing fitness over the course of evolution. The inclusion of the top 10 best networks also allows us to maintain relatively high crossover and mutation rates in producing the remaining children. We performed 100 tests for each of the selection algorithms, and the same 100 random number generator seeds were used for all selection algorithms; thus, the difference in performance is entirely an effect of the different selection algorithms and not an artifact of different initial populations.

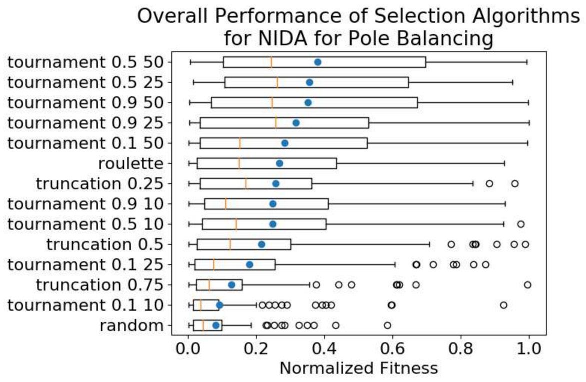
Figure 2 shows the final fitness performance for both NIDA and DANNA on Pong and pole balancing. We can note that for Pong, the top three selection algorithms for both NIDA and DANNA are roulette, truncation and then tournament, though the parameter for truncation differs for NIDA and DANNA. Truncation with $p = 0.75$ performed well for both applications for DANNA, but $p = 0.75$ for truncation had very poor performance for NIDA in both cases. For pole balancing, the different selection algorithms performed very differently for NIDA and DANNA. In all, it appears that the proper selection algorithm for any given application and model depends on both the chosen model and the application. There is not a one-size-fits-all approach that is guaranteed to perform well for



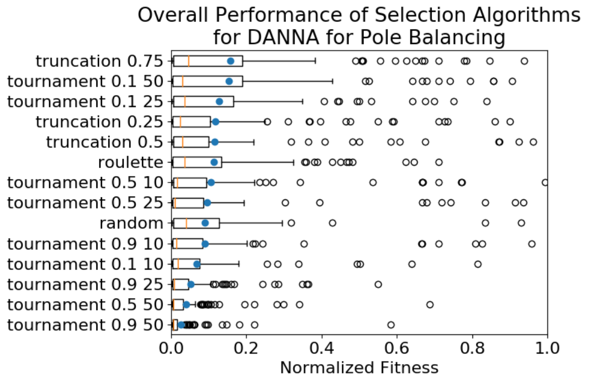
(a) Pong using NIDA



(b) Pong using DANNA



(c) Pole balancing using NIDA



(d) Pole balancing using DANNA

Fig. 2. Box plots showing final fitness value on Pong and pole balancing for NIDA and DANNA for different selection algorithms and parameters. The box plots are sorted, with the highest mean fitness value selection algorithm at the top and the lowest at the bottom. The mean fitness value is plotted as the solid blue circle.

all possible SRNNs and all possible applications. As such, if utilizing a standard selection approach, it is likely worthwhile to test a variety of selection approaches or implement an adaptive selection strategy.

It is also worth noting from Figure 2 that it is typically more difficult to build a DANNA network than it is to build a NIDA network for a given task. This is unsurprising, as there are fewer constraints on NIDA networks than there are on DANNA. It is also worth noting that Pong is harder for DANNA than pole balancing is, and in fact, out of all 1400 tests (across the 14 different selection approaches), none achieved maximum performance.

C. Population Diversity

To understand the effect of population diversity on performance, we must create a diversity metric. We consider the fitness "profile" of each network in the population to be an array containing its overall fitness value f and its various sub-goal fitness values f_i. One way to measure the diversity of a given population is to consider how many unique fitness profiles there are. Though two networks may not be identical in structure, if they have the same fitness profile then they are likely functionally similar if not functionally identical. To measure how well particular selection algorithms maintain diversity over time, we track the number of unique fitness profiles for each generation and then average those values over the entire evolution. Table I shows the relationship between the fitness values and diversity metric for the different combinations of SRNNs and applications, as well as Pearson's correlation coefficient (PCC) between the fitness value and the diversity metric for each of the test cases. In all cases except for NIDA on pole balancing, higher diversity correlated to higher fitness performance (positive correlations between the two). NIDA on pole balancing, however, had a significantly higher diversity than all of the other combinations and a negative correlation, indicating that too much diversity can also hurt performance. This is consistent with the idea of balancing exploration and exploitation that is present in many optimization algorithms, including evolutionary optimization. In general, however, if population fitness performance is stagnating, promoting diversity is more likely than not to help performance. Thus, selection algorithms that have some notion of how diverse the population is may be able to exploit that information to improve performance, which we explore in the next section.

TABLE I
RELATIONSHIP BETWEEN FITNESS AND DIVERSITY

	Pole Balancing			Pong		
	Mean Fitness	**Mean Diversity**	**PCC**	**Mean Fitness**	**Mean Diversity**	**PCC**
NIDA	0.329	53.77	-0.271	0.486	25.23	0.452
DANNA	0.122	29.83	0.293	0.261	18.03	0.301

D. Exploiting Diversity Information in Selection

Since the fitness profile information is collected over the course of training and an increase in diversity tends to lead to an increase in fitness with the baseline selection algorithms, taking into account the fitness profile during selection may improve performance. As such, we experimented with altering the selection algorithm to use fitness profile information as part of making a selection decision, which we call *diversity-aware selection*. Rather than looking at unique fitness profiles for each network in the population, we instead assigned a category number to each of the networks based on its fitness profile. In particular, if the application has G sub-goals for its fitness, there are a corresponding $G+1$ categories that each network may belong to. We assign a network category as follows:

$$\text{category}(\text{net}_i) = \text{argmax}_{i=1,\ldots,G} f_i \tag{5}$$

If there are multiple f_i values that achieve the maximum value, then the category is set to 0. As such, the category values are assigned such that the category number corresponds to the sub-goal for which it has the best performance, and the category is assigned to 0 if there is more than one sub-goal that has the "best" performance. We utilized these categories in our selection procedure in the following ways:

- We forced the set of best networks to include the network for each of the G categories that had the best f_i score for that category.
- Over the course of selection, we repeatedly iterated over the $i \in \{0, 1, \ldots, G+1\}$ categories, and for any given selection of two parents, we required the first parent to come from category i for that selection round and the second parent to come from a non-i category. The selection procedure followed the specified selection algorithm (i.e., roulette, tournament, etc.), but it was repeated until parents from the appropriate categories were chosen (or until a fixed number of selections were attempted).

Because there is not one clear selection algorithm that performs best, we compare the best result for each seed value across all selection algorithms. Figure 3 shows how many of the 100 tests for each application and SRNN model combination performed better, worse, or the same as the basic selection algorithm (the results shown in Figure 2) and Figure 4 shows more detail about the differences in performance between the original selection implementation and the diversity-aware selection implementation. These figures show that, on average, the diversity-aware selection approach produces better results for all of the different application and SRNN combinations, though the improvement is more significant for DANNA than NIDA.

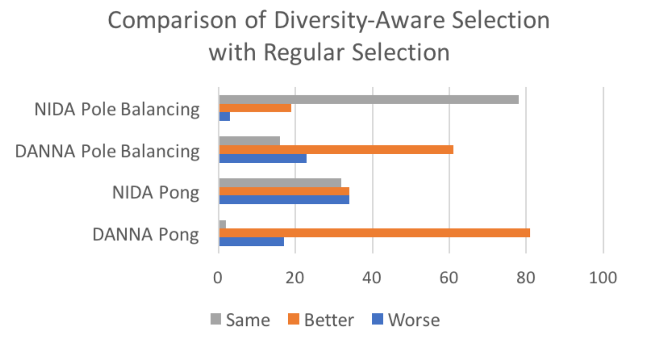


Fig. 3. Bar chart showing the number of instances where the best performance of the diversity-aware selection is better, worse, or the same as the normal selection approach. The comparison was made between the best final fitness value across all of the 14 selection algorithms for each both normal and diversity-aware selection for each of the 100 tests cases.

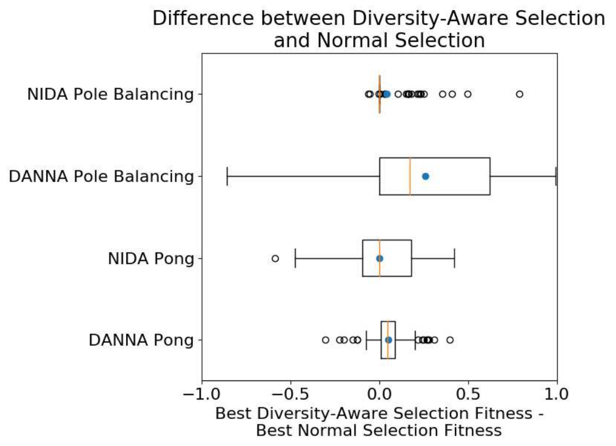


Fig. 4. Box plot showing the difference in the final fitness value between the diversity-aware selection and normal selection across the 100 tests cases. The best final fitness value was found across all 14 selection approaches for each of the 100 tests cases. The mean difference is plotted as a solid blue circle.

VI. CONCLUSION

Evolutionary approaches for training spiking recurrent neural networks (SRNNs) have many advantages, but are not widely used because they may not converge in a timely manner. In this work, we explore how selection algorithms perform when training different SRNN models for different applications, and we extend the selection algorithm approach to improve the training performance. In particular, we compare the performance of different selection algorithms both in terms of best achieved fitness in training. We found that there is not one clear selection algorithm to use, as different selection algorithms produce the best behavior for different SRNN models and applications. We also demonstrated the link between diversity in the population and training performance across all of the selection algorithms; namely, higher diversity tends to lead to better performance, but too much diversity in the population can also lead to poorer performance. Based on these results, we augmented the selection approach to utilize a richer fitness definition, in particular, a fitness profile rather than a single fitness value. We demonstrate that utilization of this additional information in the selection procedure leads to increases in performance of the evolutionary optimization approach when training over the same number of generations or epochs.

There is much future work to explore with respect to selection algorithm improvements for SRNNs and neuromorphic networks. We plan to explore adaptive selection approaches so that the evolutionary optimization can switch between different selection approaches if training performance begins to plateau. We also intend to explore additional diversity-aware fitness approaches, for example, by developing additional measures of diversity. Overall, we are encouraged by the preliminary results presented in this work, and we believe that continuing to exploit this information will help in producing evolutionary optimization approaches for training SRNNS effectively and in a timely manner in the future.

Acknowledgment

Research sponsored 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. This research was supported in part by an Air Force Research Laboratory Information Directorate grant (FA8750-16-1-0065).

References

[1] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[2] J. Cabessa and H. T. Siegelmann, "The super-turing computational power of plastic recurrent neural networks," *International journal of neural systems*, vol. 24, no. 08, p. 1450029, 2014.

[3] 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.

[4] S. M. Bohte, J. N. Kok, and H. La Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1, pp. 17–37, 2002.

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

[6] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in computational neuroscience*, vol. 9, 2015.

[7] G. Srinivasan, S. Roy, V. Raghunathan, and K. Roy, "Spike timing dependent plasticity based enhanced self-learning for efficient pattern recognition in spiking neural networks," in *Neural Networks (IJCNN), 2017 International Joint Conference on*. IEEE, 2017, pp. 1847–1854.

[8] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[9] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 47–62, 2008.

[10] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[11] F. Gomez, J. Schmidhuber, and R. Miikkulainen, "Accelerated neural evolution through cooperatively coevolved synapses," *The Journal of Machine Learning Research*, vol. 9, pp. 937–965, 2008.

[12] D. Yeung, J.-C. Li, W. Ng, and P. Chan, "Mlpnn training via a multiobjective optimization of training error and stochastic sensitivity," *Neural Networks and Learning Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[13] N. T. Siebel and G. Sommer, "Evolutionary reinforcement learning of artificial neural networks," *International Journal of Hybrid Intelligent Systems*, vol. 4, no. 3, p. 171, 2007.

[14] P. P. Palmes, T. Hayasaka, and S. Usui, "Mutation-based genetic neural network," *Neural Networks, IEEE Transactions on*, vol. 16, no. 3, pp. 587–600, 2005.

[15] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton, "Optimizing deep learning hyper-parameters through an evolutionary algorithm," in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*. ACM, 2015, p. 4.

[16] H. Burgsteiner, M. Kröll, A. Leopold, and G. Steinbauer, "Movement prediction from real-world images using a liquid state machine," *Applied Intelligence*, vol. 26, no. 2, pp. 99–109, 2007.

[17] Y. Zhang, P. Li, Y. Jin, and Y. Choe, "A digital liquid state machine with biologically inspired learning and its application to speech recognition," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 11, pp. 2635–2649, 2015.

[18] Y. Jin, R. Wen, and B. Sendhoff, "Evolutionary multi-objective optimization of spiking neural networks," in *Artificial Neural Networks–ICANN 2007*. Springer, 2007, pp. 370–379.

[19] R. Batllori, C. B. Laramee, W. Land, and J. D. Schaffer, "Evolving spiking neural networks for robot control," *Procedia Computer Science*, vol. 6, pp. 329–334, 2011.

[20] N. Kasabov, V. Feigin, Z.-G. Hou, Y. Chen, L. Liang, R. Krishnamurthi, M. Othman, and P. Parmar, "Evolving spiking neural networks for personalised modelling, classification and prediction of spatio-temporal patterns with a case study on stroke," *Neurocomputing*, vol. 134, pp. 269–279, 2014.

[21] C. D. Schuman, J. S. Plank, A. Disney, and J. Reynolds, "An evolutionary optimization framework for neural networks and neuromorphic architectures," in *International Joint Conference on Neural Networks*, Vancouver, July 2016.

[22] J. Schemmel, K. Meier, and F. Schürmann, "A vlsi implementation of an analog neural network suited for genetic algorithms," in *International Conference on Evolvable Systems*. Springer, 2001, pp. 50–61.

[23] K. D. Carlson, N. Dutt, J. M. Nageswaran, and J. L. Krichmar, "Design space exploration and parameter tuning for neuromorphic applications," in *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. IEEE Press, 2013, p. 20.

[24] G. Howard, E. Gale, L. Bull, B. de Lacy Costello, and A. Adamatzky, "Towards evolving spiking networks with memristive synapses," in *Artificial Life (ALIFE), 2011 IEEE Symposium on*. IEEE, 2011, pp. 14–21.

[25] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of genetic algorithms*. Elsevier, 1991, vol. 1, pp. 69–93.

[26] R. Sivaraj and T. Ravichandran, "A review of selection methods in genetic algorithm," *International journal of engineering science and technology*, vol. 3, no. 5, pp. 3792–3797, 2011.

[27] C. D. Schuman, "Neuroscience-inspired dynamic architectures," Ph.D. dissertation, University of Tennessee, May 2015.

[28] M. E. Dean, C. D. Schuman, and J. D. Birdwell, "Dynamic adaptive neural network array," in *13th International Conference on Unconventional Computation and Natural Computation (UCNC)*. London, ON: Springer, July 2014, pp. 129–141.

[29] C. D. Schuman, J. D. Birdwell, and M. E. Dean, "Spatiotemporal classification using neuroscience-inspired dynamic architectures," *Procedia Computer Science*, vol. 41, pp. 89–97, 2014.

[30] J. P. Mitchell, G. Bruer, M. E. Dean, J. S. Plank, G. S. Rose, and C. D. Schuman, "NeoN: Neuromorphic control for autonomous robotic navigation," in *IEEE 5th International Symposium on Robotics and Intelligent Sensors*, Ottawa, Canada, October 2017, pp. 136–142.

[31] J. S. Plank, G. S. Rose, M. E. Dean, C. D. Schuman, and N. C. Cady, "A unified hardware/software co-design framework for neuromorphic computing devices and applications," in *IEEE International Conference on Rebooting Computing (ICRC 2017)*, Washington, DC, November 2017.

[32] C. D. Schuman, A. Disney, S. P. Singh, G. Bruer, J. P. Mitchell, A. Klibisz, and J. S. Plank, "Parallel evolutionary optimization for neuromorphic network training," in *Machine Learning in HPC Environments, Supercomputing 2016*, Salt Lake City, November 2016.