Intelligent Reservoir Generation for Liquid State Machines using Evolutionary Optimization

John J. M. Reynolds, James S. Plank
Department of Electrical Engineering and Computer Science
University of Tennessee
Knoxville, Tennessee 37996
Email: [jreyno40, jplank]@vols.utk.edu

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

***Abstract*—Neuromorphic Computing is a burgeoning field of research. Many groups are exploring hardware architectures and theoretical ideas about spiking recurrent neural networks. The overarching goal is to exploit the low power promise of these neuromorphic systems. However, it is difficult to train spiking recurrent neural networks (SRNNs) to perform tasks and make efficient use of neuromorphic hardware. Reservoir Computing is an attractive methodology because it requires no tuning of weights for the reservoir itself. Yet, to find optimal reservoirs, manual tuning of hyperparameters such as hidden neurons, synaptic density, and natural structure is still required. Because of this, researchers often have to generate and evaluate many networks, which can result in non-trivial amounts of computation. This paper employs the reservoir computing technique (specifically liquid state machines) and genetic algorithms in order to develop useful networks that can be deployed on neuromorphic hardware. We build on past work in reservoir computing and genetic algorithms to demonstrate the power of combining these two techniques and the advantage it can provide over manually tuning reservoirs for use on classification tasks. We discuss the complexities of determining whether or not to use the genetic algorithms approach for liquid state machine generation.**

***Index Terms*—spiking neural networks, reservoir computing, genetic algorithms, machine learning, neuromorphic computing.**

I. Introduction

Spiking recurrent neural networks (SRNN) are an increasingly popular model of computation. This is in part due to the rise of commercial spiking neuromorphic systems such as IBM's TrueNorth and Intel's Loihi [1] [2] [3]. Because of their theoretical capabilities [4], they are able to tackle complex tasks such as spatiotemporal data processing [5]. A key question associated with SRNNs is that of training, both for general SRNNs and for those developed with intent to deploy on real neuromorphic systems. Specifically, determining the network's topology (number of neurons and synapses and their connectivity) as well as the parameters of the network (weights, thresholds, etc.) is not a straightforward task. To achieve the full computational power of SRNNs and thus of neuromorphic systems, it is imperative that we develop effective algorithms for building SRNNs tailored to real tasks.

In this work, we explore two optimization techniques for randomly creating reservoirs in a liquid state machine implementation of reservoir computing. The first technique is a grid search on two hyperparameters – hidden neurons and synaptic density – where random reservoirs are created on each instance of the grid search. The second technique employs a genetic algorithm called *Evolutionary Optimization of Neuromorphic Systems (EONS)*, where smaller random populations of reservoirs are created and then evolved to optimize the reservoirs. We compare the two approaches on a classification problem implemented by a spiking neuromorphic system called DANNA2 [6]. We also explore the impact of standard metrics for reservoirs on each of the reservoirs generated by both techniques.

II. Background and Related Work

Spiking neural networks gained prominence in neural network literature in the late 1990's. One of the most important distinguishing factors between spiking neural networks and previous generations is the non-trivialization of time in the computation of the network. In particular, the timing of events in a spiking neural network are integral to the way that computation occurs [4]. Since then, they have become increasingly popular, partially due to the rise of commercial spiking neuromorphic hardware [1]. Spiking recurrent neural networks (SRNNs) are those that allow for recurrent connections. Typically, these networks are not organized in a layered structure that is typical with other neural network types.

As noted in the introduction, a key issue associated with spiking neural networks is how to train them. Gradient-descent or back-propagation-based spiking neural network training algorithms have been proposed [7], but they typically rely on a pre-determined network topology and do not fully leverage the computational capabilities of the networks. One of the most common training approaches for spiking neural networks in general (including spiking recurrent neural networks) is spike-timing dependent plasticity or STDP [8]. Again, in this case, elements of the network like synaptic delays and topology may not be defined by the training algorithm, leaving them for the user to define, often in an ad hoc manner.

Defining network topology in a systematic way has been done for both traditional neural networks and spiking neural networks using algorithms in the field of neuroevolution [9]. Neuroevolution methods have been used to determine all aspects of neural networks, including determining network topology and parameters simultaneously [10] and determining network topology while utilizing a more traditional weight training algorithm [11]. Neuroevolution methods have also been used specifically for evolving spiking neural networks [12], [13] and spiking neural networks for neuromorphic implementation [14], [15].

Reservoir computing is another widely used approach for utilizing recurrent neural networks [16]. As previously mentioned, it is particularly attractive because no tuning of reservoir synaptic weights is required. However, the technique still typically requires the tuning of hyperparameters, which we discuss later in this work. The two main reservoir computing approaches are echo state networks and liquid state machines. Echo state networks [17] use non-spiking recurrent neural networks for the reservoir. Liquid state machines [18] use spiking recurrent neural networks. Reservoir computing and liquid state machines have successfully utilized spiking recurrent neural networks for a variety of applications, such as robotics controls, object tracking, motion prediction, pattern classification, signal processing, and time-series prediction/-classification [19], [20], [21], [22].

Using memristive devices and neuromorphic architectures for reservoir computing is also an active area of research [23], [24], [25], [26]. The ability for spiking architectures to provide interesting complex computation without training the full network leads to energy efficient hardware applications. Further, reservoir computing can be implemented with other sufficiently dynamical physical systems. For instance, research groups have demonstrated implementations of reservoir computing using actual liquid [27], optoelectronics [28], and quantum systems [29].

III. Spiking Recurrent Neural Network Model and Training Methods

A. Spiking Recurrent Neural Network Model

For this work, we use the DANNA2 neuromorphic model [6]. DANNA2 offers a software simulation and a digital hardware architecture for spiking recurrent neural networks (SRNN). Each network is a collection of neurons arranged within a two-dimensional coordinate system. Neurons follow a simple accumulate and fire model in which each neuron accumulates charge over time until a specified threshold is reached leading to an output spike. After a neuron fires, it enters a configurable refractory period in which it may not fire again until refractory is complete. Further, neurons have a configurable leak parameter. Spikes propagate from one neuron to another through synapses. Each synapse transfers a spike from one neuron to another with a configurable weight value and temporal delay. The weight parameter for a given synapse can be dynamically modified during operation using Spike Timing Dependent Plasticity (STDP). Weights can be both negative and positive, thus allowing for both excitatory and inhibitory connections. In this work, STDP is not utilized.

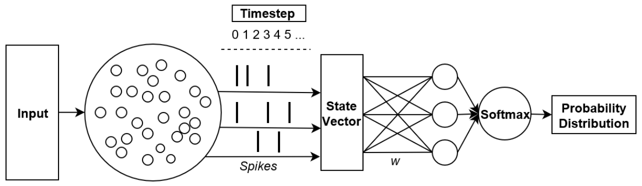


Fig. 1: Diagram of reservoir computing. Input is passed into the reservoir (liquid in this case), then after a predetermined amount of time, the reservoir is sampled and the sample is passed into the readout layer as a feature vector.

B. Training: Reservoir Computing

A Liquid State Machine (LSM), shown in Figure 1 is a computational model incorporating the time-varying behavior of recurrent spiking neural networks as a filter for information. The LSM model includes three parts; an input layer, the liquid, and a readout layer. In order to act successfully as the liquid or reservoir, a spiking neural network must have two important properties: input separability and fading memory. Input separability simply means that the liquid filters the data in such a way that, provided different inputs, different states are reached. Without this property, a readout mechanism will be unable to differentiate between the different inputs. For example, measures such as Euclidean distance can be used to track rates of separability across output vectors [4]. Fading memory is the requirement that a single input will not recursively propagate ad infinitum. Several measures of reservoir stability have been explored, such as the spectral radius and the Jacobian of the weight matrix [30]. These features give numerical measures of reservoir stability with respect to fading memory, and they have been used in studies of genetic algorithms for reservoir generation [31].

If a network has input separability and fading memory, it can be proven to be a universal function approximator via the Stone-Weierstrass theorem [20]. In theory, if a spiking neural network has these two characteristics, then it can act as the liquid for a LSM. Thus, the only training step for the spiking neural network component is ensuring that the network is sufficiently complex as to have both properties required of the liquid. This can make randomly generated liquids sufficient. However, the likelihood of a randomly generated network working well for a given problem depends on prior information, such as a relative size for the randomly generated liquid. It also depends on the problem to be solved. Hyperparameters like the number of hidden neurons, the synaptic density, the actual topology (connections), leak values, and any other tunable network parameters can have an impact on reservoir success. In this respect, researchers often have to generate and evaluate many networks before arriving at a successful or optimal reservoir, even when a valid set of hyperparameters are selected. As we discuss in this paper, genetic algorithms can help to generate successful topologies without needing to have intuition about these properties in advance.

After the potential liquid has been generated, it is stimulated to generate states for post-processing. The readout layer is trained to classify information from the extracted reservoir state vectors. In particular, the readout layer is typically trained with linear regression, though the definition of a Liquid State Machine is general enough to allow for different kinds of readout layers such as support vector machines, perceptrons, or multivariate logistic regression (softmax).

Here, we use a multinomial logistic regression (a softmax classifier) for the readout layer. The weight matrix is trained with gradient descent and the backpropagation algorithm. Input is fed into the liquid, which transforms its state throughout the process of spiking events. It is this altered state (the outputs of the liquid) that is captured for processing by the readout layer. After a predetermined number of simulation cycles, the state of the liquid is taken and passed to the softmax classifier. The state representation of the liquid is contained within a vector. The state vector Z that we utilize in this work consists of the number of spikes an output produces throughout simulation, and contains elements equal to the number of outputs of the network. A state corresponding to a particular input may be reused; it does not have to be re-simulated for every training step. Mathematically, the representation is described as follows:

$$Z_i = \sum_{j}^{N} Output_{i,j} \tag{1}$$

N is equivalent to the number of simulated timesteps. The vector $Output_{i,j}$ contains a 0 if the output neuron i did not fire on timestep j, and a 1 if it did fire. Z_i represents the state for output i. The state vector is computed after simulation and passed to the readout layer. Instead of a snapshot of the entire state of the liquid at a particular time, the state vector tracks the dynamical behavior of the network through output spikes. Note that this is merely one possible state representation used for the ease of computation. Other possibilities exist and can improve classification performance through the use of more extensive information from the spike trains, such as low-pass exponential filtering of the generated spike signals.

The representation described above is passed through the readout layer to determine the final output. During the training phase, the readout layer then computes the error gradient of the softmax and squared error cost function. Lastly, backpropagation is used to update the weights.

C. Training: Evolutionary Optimization of Neuromorphic Systems

For our genetic algorithm, we use Evolutionary Optimization of Neuromorphic Systems (EONS), which has been employed previously to train SRNNs intended for use on neuromorphic systems [14], [32]. What makes EONS interesting is that it optimizes not only the numeric hyperparameters of the SRNN's, but also their structure. Networks in the EONS populations are represented as graphs in which the nodes and edges have optimizable parameters. Crossover and mutation operations leverage this graph-based representation. Each application must implement a fitness function that takes a network as input and returns a single numerical score, in which higher fitness scores correspond to better performing networks. The EONS framework determines both the topology of the network (number of neurons and synapses and connectivity pattern) and the parameters of the network (weights, thresholds, etc.) Note that having to define a fitness function which determines what output comprises high success rates is different than that of a readout layer in reservoir computing.

D. EONS Applied to LSMs

We explore two methods for training liquid state machines: grid search and EONS. To use EONS, we must provide a value of fitness for each network in the population. The value of fitness is computed in multiple steps. First, the network is stimulated by applying spikes over time. After the predetermined number of time-steps have passed, the state vectors are collected in the format described in the preceding section. Then, the readout layer is trained. The fitness achieved by the readout layer is returned as the fitness value. Several other fitness values for genetic algorithms training liquid state machines have been explored, such as using metrics describing the network separability or spectral radius [33]. As we demonstrate in our results, relying on those properties may not be the best approach to finding successful reservoirs.

IV. Experiment and Results

The classification accuracies of reservoir computing on many well known datasets have been demonstrated in previous work by other researchers [34]. Here we focus on the Ionosphere dataset as a case study [35], as our goal is to demonstrate the usefulness of EONS when compared to grid search. First, we will describe the dataset. Then, we will delve into the experiment we performed. Finally, we will examine the results obtained with each of the individual methods.

A. Ionosphere

The Ionosphere dataset is a collection of information generated by a phased array of 16 high-frequency antennas. The antennas send a multipulse pattern targetting free electrons in the ionosphere. A receiver is activated in-between pulses. The result is 17 pulse numbers each composed of 2 complex attributes representing the electromagnetic signal, resulting in a total of 34 continuous valued features. Each of the 34 samples is converted into a range of 10 pulses on an individual input neuron, with an interval of 1 time step between pulses. Thus, 34 inputs, each spiking a number of pulses proportional to the input value. The phase shift of returns is measured to determine the target velocity. In specific, an autocorrelation function is computed on the returned signal, which can be used to determine the Doppler velocity of the target. If a target is moving with constant velocity, the autocorrelation function will show a phase shift proportional to the pulse number. A sample is determined to be a "good" radar return if structure was detected (and the phase shift is proportional), and determined to be "bad" otherwise. Reservoir computing results

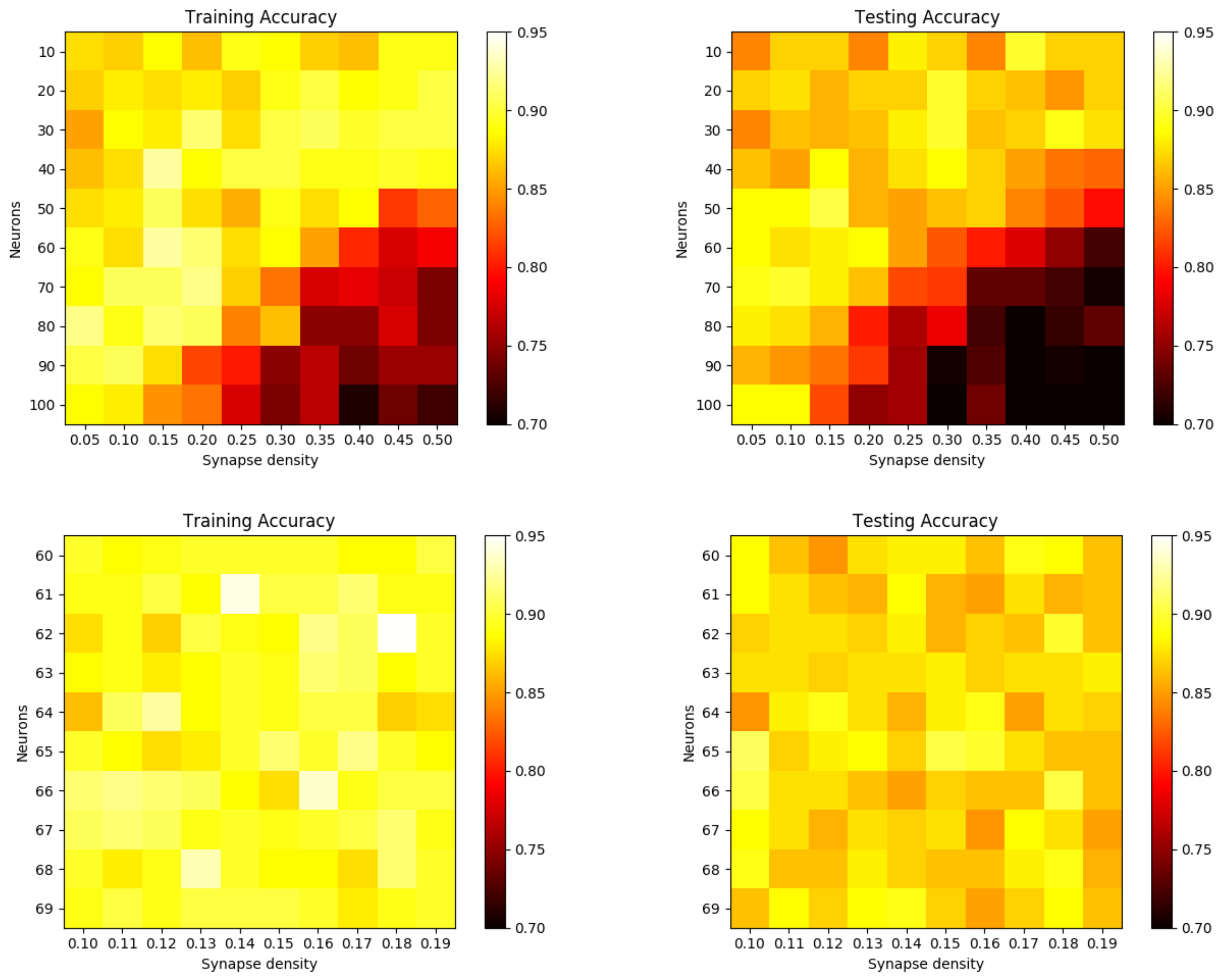


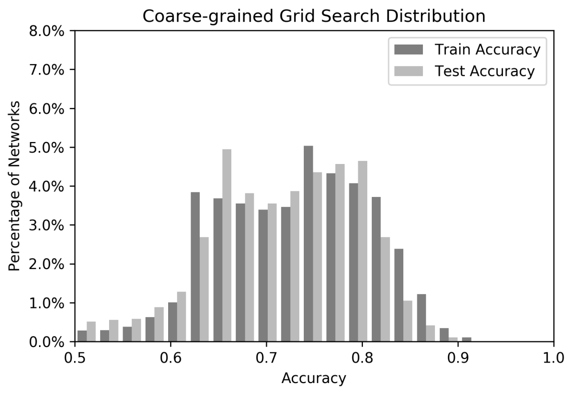
Fig. 2: The heat maps above were generated from a two-level grid search on randomly created reservoirs trained on the ionosphere dataset. Each block in the heat map represents the best accuracy from a population of 100. The number of hidden neurons and the synaptic density are varied on the y and x axes respectively. Note that the y-axis refers to number of hidden neurons, and that the synaptic density refers to hidden + 20 outputs, so $(hidden + 20)^2 * density$ for all possible connections. The bottom heat maps are of a second level search across the 60 neurons block centered on the best accuracy from the top level grid search: 0.15 synaptic density. This demonstrates that the hyperparameter configuration is an integral part of reservoir success.

from randomly generated reservoirs are previously reported as 92%, but an accuracy of 96% is typical of traditional machine learning algorithms, with a high of 98% [34].

B. Experiment Design

For our experiment we evaluated 60,000 networks using both grid search and EONS, which resulted in a total of 120,000 network evaluations. The number of tests was kept the same to demonstrate the results of each method with the exact same number of network simulations. There are 34 inputs to the reservoir, 20 outputs from the reservoir to the readout, and a softmax layer with two outputs for the readout. Each method was evaluated on the ionosphere dataset with a 50/50 train/test split. The readout was trained on each individual network for 1,000 epochs with a learning rate of 0.001 (determined experimentally). DANNA2 synaptic weights range from -256 to 255, and delays range from 0 to 255 [6]. Values are uniformly distributed across this range for generation.

We recorded the training and testing accuracy of each of the 120,000 evaluated networks as well as the metrics relevant to reservoir computing discussed in previous sections. Specifically, we look at the spectral radius of networks, which is related to the fading memory property. We also recorded the silhouette coefficient, which is a commonly used metric in statistics ranging from -1.0 to 1.0. The silhouette coefficient represents the validity of clusters generated by clustering methods. A coefficient value of -1.0 means that all points are incorrectly clustered, and a value 1.0 represents fully valid clusters [36]. In the context of reservoir computing, it is a single value that can be viewed as a representation of the ability of a liquid to separate information, and it is computed



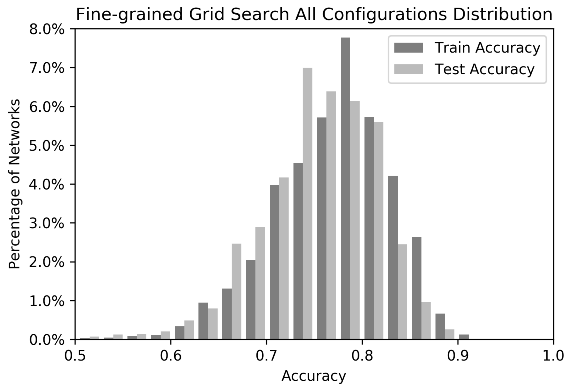
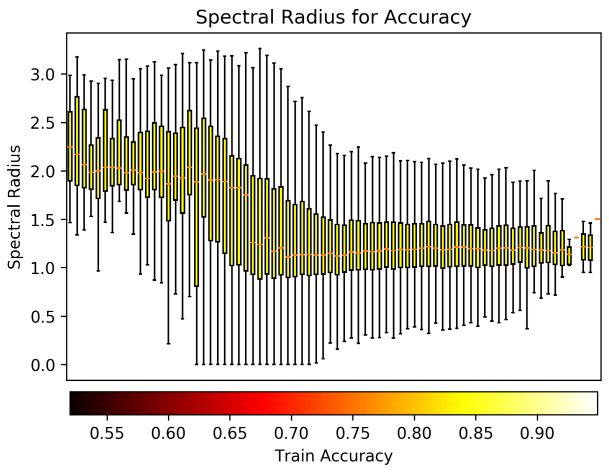


Fig. 3: The histograms depicted above represent the accuracy distribution across configurations utilized in the grid search. It is clear that a valid choice of hyperparameters still does not ensure successful reservoirs. Further, even within a fine-grained search across a good choice of hyperparameters, many liquid evaluations may be required. The accuracy distribution is higher on average in the fine-grained approach, but the highest performing reservoirs are still a small portion of the overall distribution.



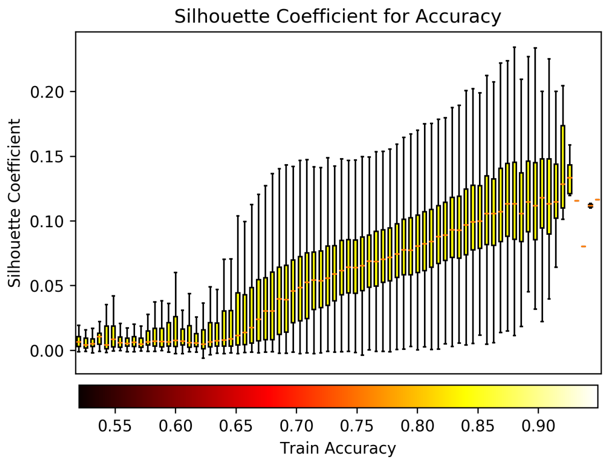


Fig. 4: These graphs are generated from all 60k of the grid search tests. The average silhouette coefficient increases with accuracy, but still has a wide range of values. The spectral radius also narrows to a certain range with higher accuracy, but spectral radius values in that range exist across a majority of the spectrum of accuracy, so having a value in that range does not guarantee success, nor prevent it.

from the state vectors generated by sampling a liquid after stimulation.

C. Grid Search Results

Knowing where to start with the hyperparameters for liquid state machine generation can be unclear. There are guidelines and recommendations for reservoir computing [37], but often researchers have to generate many random networks to find a satisfactory solution. To demonstrate this, we performed a grid search varying the number of hidden neurons and the synaptic density (relative to the number of hidden neurons). The number of hidden neurons was varied from 10 to 100 in increments of 10, and the synaptic density was varied from 5% to 50% in increments of 5%. For each of these configurations, 100 random networks were generated. After the coarse-grained grid search, the five best hyperparameter settings for number of hidden neurons and synaptic density were chosen, and a fine-grained grid search was performed in which we fine-tuned the search to vary across the best performing configurations in increments of 1 neuron and 1% synaptic density (also with 100 networks per configuration). In total, the coarse-grained grid search was 10,000 tests, and each fine-grained grid search was 10,000 tests, for a total of 60,000 tests.

The relevant results are shown in the heat maps in Figure 2. The top two heat maps are of the coarse-grained grid search training and testing accuracy. It is clear that certain configurations outperform others, with the highest synaptic densities resulting in far lower levels of accuracy. This is a result of the network activity becoming too chaotic to convey information. The success depends upon the combination of neurons & synaptic density rather than on one of the two parameters. In other words, this forms a joint probability distribution. To further demonstrate that hyperparameter configuration is important, we display results from the fine-grained search on the best training accuracy spot: 60 neurons centered on a synaptic density of 0.15. The bottom heat maps are of this search, and show that high accuracy is attained across the spectrum of hyperparameters when an appropriate coarse-grained configuration is found. However, it is important to note that these heat maps are demonstrating the best accuracy achieved out of 100 networks for each configuration. Even after finding an appropriate selection of hyperparameters, the distribution of reservoir success can vary wildly. This is shown in Figure 3, and serves as an example of the need to evaluate multiple reservoirs before finding a successful one. An exhaustive search is required in order to know the true distribution of successful reservoirs.

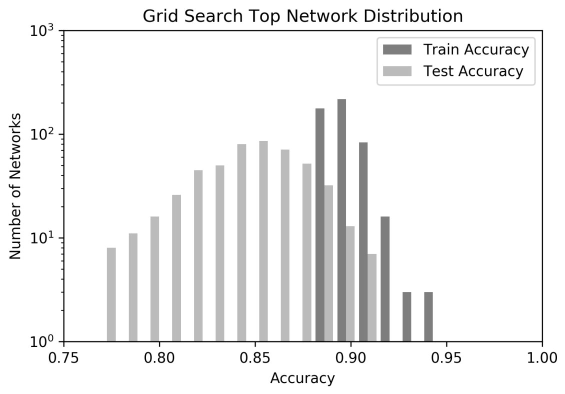
As mentioned earlier in this paper, metrics associated with reservoirs are often evaluated in order to determine the validity and success of said reservoir. While there certainly can be a correlation between certain metrics and accuracy, having a "good" value for such a metric does not guarantee success. On the other hand, having high accuracy does not mean one will have good reservoir values. This is demonstrated in the box plots in Figure 4. We can see that the average silhouette coefficient increases with accuracy, but still has a wide range of values. Further, the highest accuracy networks seemingly do not follow the trend, and thus appear to be outliers. Notice the lack of range on some of the x points, as they appear to be single instances of these high accuracy networks rather than repeatable scores. They are included because this data was all generated by the experiments performed as a part of this work, and the outliers demonstrate the necessity of hyperparameter tuning to achieve high levels of reservoir performance.

The spectral radius also narrows to a certain range with higher accuracy, but spectral radius values in that range exist across a majority of the spectrum of accuracy, so having a value in that range does not guarantee success [37]. Because the spectral radius alone is an inadequate measure of reservoir capability, and separability metrics all rely on gathering state vectors from reservoir simulation, it is essentially a requirement that the liquids be simulated for evaluating success accurately. Therefore, the number of computations can easily become the same as that of performing an intelligent generation of liquids, as is done with EONS.

It is important to note that the rate of success from random generation can essentially go to zero if networks are not complex enough or too complex, even when the spectral radius falls into the recommended region from literature (which is less than 1.0, but not too low to not generate activity). The size of the network required depends entirely upon the problem attempting to be solved, and thus to truly map the distribution of rates of success from random generation for a particular problem, one would need to perform an exhaustive parameter sweep across the number of neurons, number of synapses, number of outputs being sampled, limiting delays on synapses, leak on/off, synaptic plasticity on/off, and any other adjustable network parameters. This is clearly comparable in complexity to utilizing a structured run of EONS.

D. EONS LSMs

We set the population of each EONS run to be 50 networks. We performed 100 runs, each lasting 12 epochs, and each of which matched a different configuration from the coarse-grained grid search performed for random generation. In other words, one EONS run population was initialized randomly with 10 hidden neurons and 5% density, another was 10 hidden neurons and 10% density. That way, for each grid configuration, there was a matching configuration for a run of EONS. These parameters were selected to match the grid search, as 50x100x12 = 60,000, the same number of networks as the grid search.



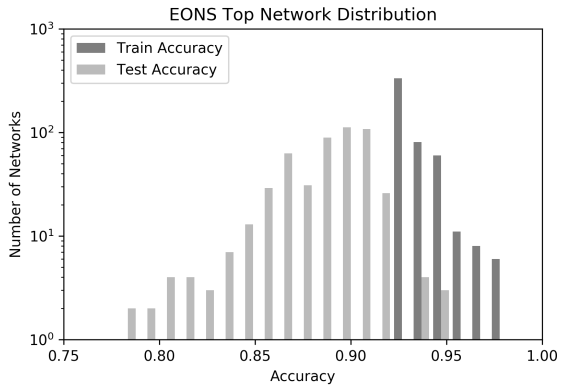


Fig. 5: Top 500 performing networks from each of the methods. The y-axis is presented on a log scale. The plots demonstrate that the EONS results are skewed towards higher performance. Slight over-fitting can be remedied through manual tweaking of parameters, and becomes less apparent when reviewing more than the top 500 performers.

In order to perform a direct comparison of the success between the grid search and the EONS runs, we sorted all 60,000 tests for each method by training accuracy. Then, the top 500 liquids were taken for each method (based on training accuracy), and the distribution of accuracy was plotted. Figure 5 demonstrates that although both methods are capable of reaching expected levels of accuracy, the structured search performed by EONS skews the distribution towards higher performing networks. Further, looking at Figure 6, we can see that EONS converges to the expected accuracy regardless of the beginning population initialization configuration. In other words, even though each EONS run began with a different number of hidden neurons and synaptic density combination, they all converge. This fits our thesis, which is that simply allowing an intelligently structured search from EONS to generate reservoirs arrives at successful hyperparameters

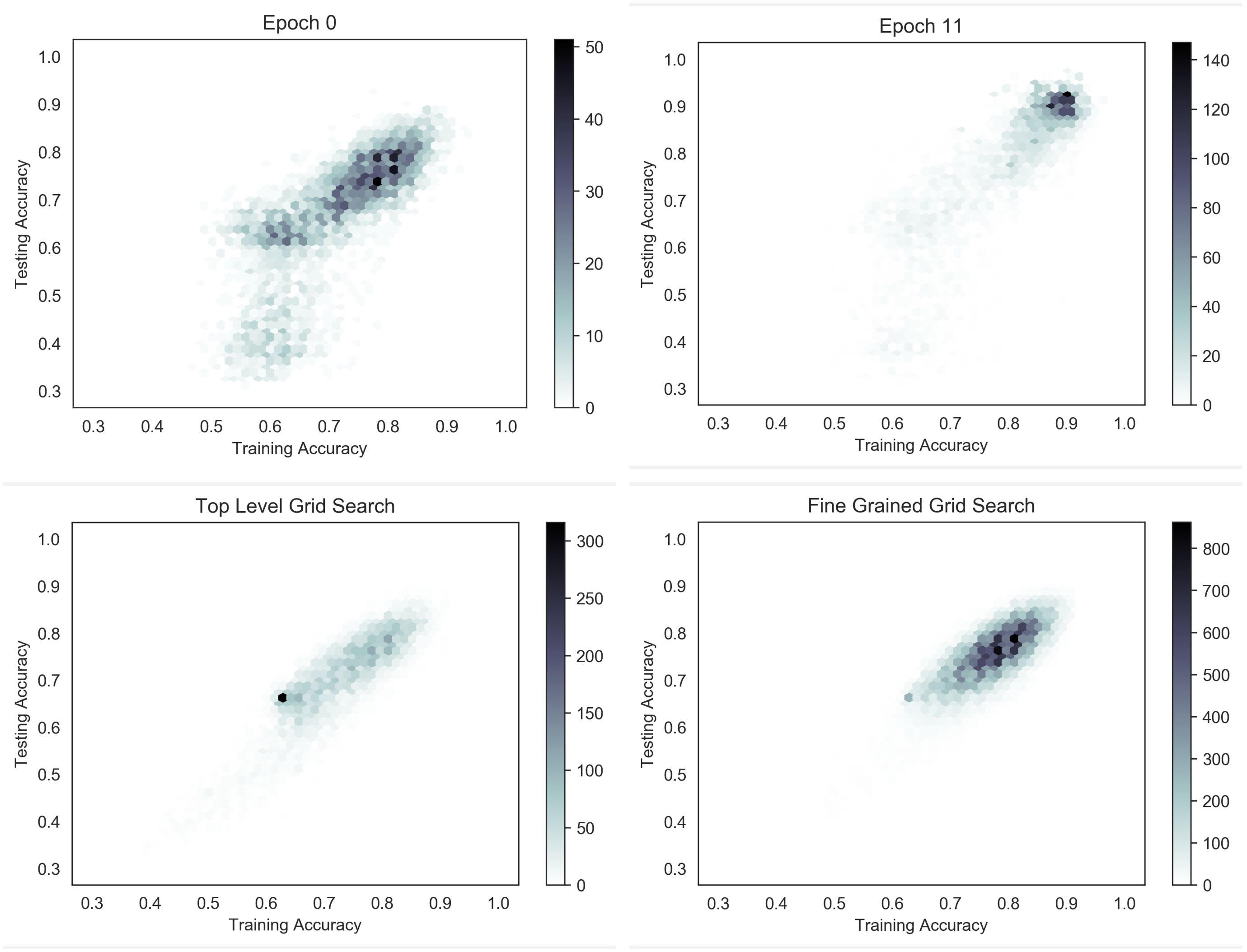


Fig. 6: The top two figures are hex-bin plots of the EONS tests performed. The top-left plot is epoch 0, and the top-right plot is epoch 11. Each plot is of all 50 networks from all 100 runs for a total of 5k. Each hexagon represents a bin, and all of the samples falling into that region are integrated into that bin. The color gradient represents the number of samples in a bin. EONS converges to the expected accuracy regardless of the starting configuration, which is an inherently different result than the two plots at the bottom; the left being the coarse-grained grid search, and the right being of all 5 fine-grained grid searches.

regardless of the starting condition. As demonstrated, this conclusion does not hold for random generation.

V. Discussion

The number of neurons in a system has an effect on the complexity that a neural network can represent. For reservoir computing, knowing exactly how many neurons are necessary to adequately separate one's problem set does not have an answer. Choosing to use too many can result in needlessly large networks, while choosing too few can easily result in networks that are incapable of separating the information. In the context of neuromorphic computing, where low power is a key promise, using an unnecessarily large size will use more power than necessary.

The properties of simulation and evaluation of SRNNs are dependent on many different factors. For instance, higher levels of connectivity typically generate more events, and are thus slower to simulate. Further, the number of neurons being sampled for output can contribute to the overall success of reservoir systems. In order for random generation of LSMs to succeed, one must be randomly generating networks with sufficient complexity. Certainly, one may perform a search over the many different parameters, but this can end up taking just as much time as EONS and perform comparably in terms of accuracy. As demonstrated in this paper, generating satisfactory reservoirs for a given problem can depend on prior knowledge, which often leads to researchers generating many random networks before arriving at one that works. In many scenarios, we would recommend that systems like EONS be put into place, as they can more efficiently search through that same problem space.

VI. Conclusion

In this paper, we have explored reservoir generation for a spiking recurrent neural network implementation of liquid state machines on a classification problem. We explored two well-known hyperparameter optimization techniques – a two-level grid search, and a genetic algorithm called EONS, which includes network structure of the reservoir in its optimization. In our tests, the EONS optimization discovered better networks on the whole, and also converged more reliably. We used our experiment to evaluate the metrics of spectral radius and silhoutte coeficient on reservoir effectiveness. While the two metrics showed converging trends for networks that train better, when used in isolation, they are not sufficient to be used to construct good reservoirs.

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] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, pp. 668–673, 2014.

[3] M. Davies, N. Srinivasa, T. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, January 2018.

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

[5] A. Polepalli, N. Soures, and D. Kudithipudi, "Reconfigurable digital design of a liquid state machine for spatio-temporal data," in *Proceedings of the 3rd ACM International Conference on Nanoscale Computing and Communication*. ACM, 2016, p. 15.

[6] J. P. Mitchell, M. E. Dean, G. Bruer, J. S. Plank, and G. S. Rose, "DANNA 2: Dynamic adaptive neural network arrays," in *International Conference on Neuromorphic Computing Systems*. Knoxville, TN: ACM, July 2018.

[7] 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.

[8] T. Masquelier and S. J. Thorpe, "Unsupervised learning of visual features through spike timing dependent plasticity," *PLoS computational biology*, vol. 3, no. 2, p. e31, 2007.

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

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

[11] E. Alba and J. F. Chicano, "Training neural networks with ga hybrid algorithms," in *Genetic and Evolutionary Computation Conference*. Springer, 2004, pp. 852–863.

[12] 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.

[13] N. Pavlidis, O. Tasoulis, V. P. Plagianakos, G. Nikiforidis, and M. Vrahatis, "Spiking neural network training using evolutionary algorithms," in *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, vol. 4. IEEE, 2005, pp. 2190–2194.

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

[15] 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.

[16] M. Lukoševičius, H. Jaeger, and B. Schrauwen, "Reservoir computing trends," *KI - Künstliche Intelligenz*, vol. 26, no. 4, pp. 365–371, Nov 2012. [Online]. Available: https://doi.org/10.1007/s13218-012-0204-5

[17] H. Jaeger, "The echo state approach to analysing and training recurrent neural networks-with an erratum note," *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, no. 34, p. 13, 2001.

[18] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[19] B. Schrauwen, D. Verstraeten, and J. V. Campenhout, "An overview of reservoir computing: theory, applications and implementations," in *Proceedings of the 15th European Symposium on Artificial Neural Networks*, 2007, pp. 471–482.

[20] W. Maass and H. Markram, "On the computational power of circuits of spiking neurons," *Journal of computer and system sciences*, vol. 69, no. 4, pp. 593–616, 2004.

[21] E. Goodman and D. Ventura, "Spatiotemporal pattern recognition via liquid state machines," in *Neural Networks, 2006. IJCNN'06. International Joint Conference on*. IEEE, 2006, pp. 3848–3853.

[22] Z. Yanduo and W. Kun, "The application of liquid state machines in robot path planning," *Journal of Computers*, vol. 4, 11 2009.

[23] M. S. Kulkarni and C. Teuscher, "Memristor-based reservoir computing," in *Nanoscale Architectures (NANOARCH), 2012 IEEE/ACM International Symposium on*. IEEE, 2012, pp. 226–232.

[24] M. Lukoševičius, H. Jaeger, and B. Schrauwen, "Reservoir computing trends," *KI-Künstliche Intelligenz*, vol. 26, no. 4, pp. 365–371, 2012.

[25] N. Soures, L. Hays, and D. Kudithipudi, "Robustness of a memristor based liquid state machine," in *Neural Networks (IJCNN), 2017 International Joint Conference on*. IEEE, 2017, pp. 2414–2420.

[26] D. Kudithipudi, Q. Saleh, C. Merkel, J. Thesing, and B. Wysocki, "Design and analysis of a neuromemristive reservoir computing architecture for biosignal processing," *Frontiers in neuroscience*, vol. 9, p. 502, 2016.

[27] C. Fernando and S. Sojakka, "Pattern recognition in a bucket," in *European Conference on Artificial Life*. Springer, 2003, pp. 588–597.

[28] Y. Paquot, F. Duport, A. Smerieri, J. Dambre, B. Schrauwen, M. Haelterman, and S. Massar, "Optoelectronic reservoir computing," *Scientific reports*, vol. 2, 2012.

[29] K. Fujii and K. Nakajima, "Harnessing disordered quantum dynamics for machine learning," *arXiv preprint arXiv:1602.08159*, 2016.

[30] D. Verstraeten and B. Schrauwen, "On the quantification of dynamics in reservoir computing," in *International Conference on Artificial Neural Networks*. Springer, 2009, pp. 985–994.

[31] A. A. Ferreira and T. B. Ludermir, "Genetic algorithm for reservoir computing optimization," in *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*. IEEE, 2009, pp. 811–815.

[32] J. S. Plank, C. D. Schuman, G. Bruer, M. E. Dean, and G. S. Rose, "The TENNLab exploratory neuromorphic computing framework," Preprint at doi.ieeecomputersociety.org/10.1109/LOCS.2018.2885976, 2018.

[33] E. Hourdakis and P. Trahanias, "Improving the classification performance of liquid state machines based on the separation property," in *Engineering Applications of Neural Networks*, L. Iliadis and C. Jayne, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 52–62.

[34] L. A. Alexandre, M. J. Embrechts, and J. Linton, "Benchmarking reservoir computing on time-independent classification tasks," in *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*. IEEE, 2009, pp. 89–93.

[35] V. G. Sigillito, S. P. Wing, L. V. Hutton, and K. B. B. and, "Classification of radar returns from the ionosphere using neural networks," *Johns Hopkins APL Tech. Dig*, vol. vol. 10, pp. 262–266, 1989, in.

[36] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53 – 65, 1987. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0377042787901257

[37] M. Lukosevicius, "A practical guide to applying echo state networks," in *Neural Networks: Tricks of the Trade*, 2012.