Neuroevolution of Spiking Neural Networks Using Compositional Pattern Producing Networks

Daniel Elbrecht
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
8de@ornl.gov

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

ABSTRACT

Spiking neural networks (SNNs) offer tremendous potential for the future of AI, including the ability to be implemented efficiently on neuromorphic systems. One of the challenges in building functioning SNNs is the training process, as standard error back-propagation cannot be easily applied. In this work, we extend an evolutionary approach for training SNNs by implementing an indirect encoding of individuals. Specifically, we evolve SNNs using Compositional Pattern Producing Networks, which are able to learn the connectivity patterns between neurons defined in a coordinate space. We validate the approach on multiple control and classification tasks.

KEYWORDS

spiking neural networks, genetic algorithms, evolutionary algorithms, indirect encoding, neuromorphic computing

ACM Reference Format:
Daniel Elbrecht and Catherine Schuman. 2020. Neuroevolution of Spiking Neural Networks Using Compositional Pattern Producing Networks. In *International Conference on Neuromorphic Systems 2020 (ICONS 2020), July 28–30, 2020, Oak Ridge, TN, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3407197.3407198

1 INTRODUCTION

Spiking neural networks (SNNs) are a class of neural networks where network communication is done via spikes, which are binary signal that propagate information through the network. SNNs incorporate biologically meaningful features such as spiking thresholds, neuron potential leakage, synaptic strength, and spike timing delay. A spectrum of biological fidelity exists within SNN models, with the most realistic models even including cellular ion gradients. Additionally, SNNs are designed to operate on neuromorphic hardware, which is characterized by parallel memory and computation, low energy inference, and other desirable features [15].

The same properties that make SNNs unique also create difficulties in training. Because the loss function for an SNN and the activation functions of individual neurons are not differentiable, back-propagation cannot be directly applied. Certain methodologies have been developed to apply a form of back-propagation to SNNs. Such methods typically impose additional constraints on SNNs to make them amenable to back-propagation, such as enforcing a feed-forward architecture, or eliminating the timing component from computations. Thus, they only operate on a subclass of potential SNN architectures. Evolutionary algorithms have emerged as a strategy for training or designing SNNs. In evolutionary approaches, neuron and synapse parameters as well as the network connectivity and structure are all potentially subject to optimization. Since evolutionary algorithms only require a definition for an individual and the ability to measure individual fitness, they are flexible for optimization in a wide variety of problem spaces.

In evolutionary algorithms, individuals are represented by encodings, which can either be direct or indirect. In a direct encoding, every attribute that defines an individual is separately specified, which can have an effect on the scalability of the evoluationary approach. Conversely, in an indirect encoding, sections of the encoding are reused, so that the similar patterns and properties will appear multiple times throughout the individual. Indirect encodings have the ability to scale, so that the encoding (genotype) is much smaller than the individual (phenotype). Additionally, indirect encodings should be able to exploit regularities in the problem space more easily than direct encodings. The hypothesis of this paper is that SNNs can be evolved through an indirect encoding approach. We demonstrate this by using an evolutionary algorithm with an indirect encoding to optimize SNNs for several control and classification tasks.

2 BACKGROUND AND RELATED WORK

2.1 Training Methods for SNNs

A common approach for training or learning in SNNs is Hebbian learning or synaptic plasticity mechanisms like spike timing dependent plasticity (STDP) [5]. STDP is based on the neuroscience axiom principle that "neurons that fire together, wire together." Though STDP is typically an unsupervised training approach, there have been efforts to extend it to be a supervised learning approach [9]. A key issue with utilizing approaches such as STDP for training or learning in SNNs is that it is often not clear what the structure of the network should be (i.e., number of neurons or layers, connectivity

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).

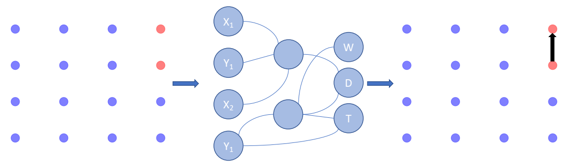


Figure 1: Generating a spiking neural network from a compositional pattern producing network. Coordinates for a pair of neurons (left, red) are passed in to the compositional pattern producing network (middle). The network output determines the weight and delay of the connection between the two neurons in the spiking neural network (right). This process is repeated for each pair of neurons to construct the full network.

between layers, etc.). This can make extending STDP-like training approaches to new applications difficult.

Multiple approaches exist which apply back-propagation to SNNs [2, 10, 16, 17]. Since the spike function is non-differentiable, these methods cannot employ traditional back-propagation, and modify the gradient update or the SNNs themselves in order to overcome this difficulty. These types of methods often enforce architectural constraints on the resultant SNNs, for example, by requiring a feed-forward SNN architecture. Therefore there exists many SNN topologies with desirable properties such as recurrence that cannot be optimized by these methods.

Evolutionary algorithms have also been developed for training SNNs [4, 7, 12, 14]. In an evolutionary approach, a population of SNNs is evaluated, and stochastic variation and selection processes are applied to generate the next population. Over successive iterations of variation and selection, this process will yield networks with increasing fitness, which can be widely defined as accuracy on a classification task, or maximum reward signal from an environment. One significant advantage of evolutionary approaches is their flexibility. When applied to SNNs, evolutionary optimization has the potential to act on any network topology, as well as to optimize any network parameter. For example, in the EONS algorithm [12], both network structure and parameter weights are optimized through the evolutionary algorithm.

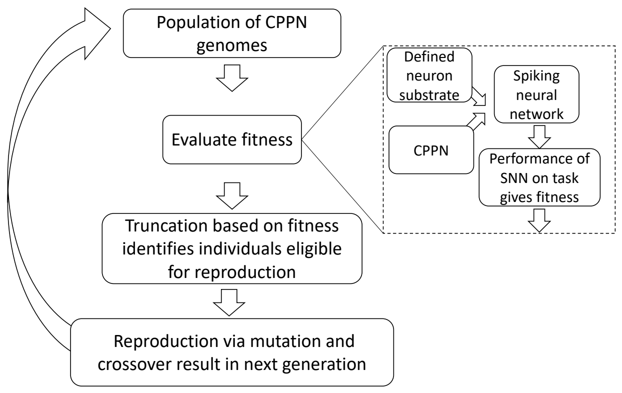

Population of CPPN genomes
Evaluate fitness
Truncation based on fitness identifies individuals eligible for reproduction
Reproduction via mutation and crossover result in next generation
Defined neuron substrate
CPPN
Spiking neural network
Performance of SNN on task gives fitness

Figure 2: Overview of proposed method

2.2 HyperNEAT

NeuroEvolution of Augmenting Topologies (NEAT) is a method for evolving traditional neural networks [20]. It uses a direct encoding, each of the network is individually represented in the genome. NEAT offers several advantages as an evolutionary algorithm: It allows for crossover between networks by tracking the origin of each parameter (node or connection) in the genome. This allows homologous genes to be "lined-up" and swapped during crossover. NEAT also uses speciation to protect innovation.

Compositional Pattern Producing Networks (CPPN) are a class of neural networks which aim to abstract the processes of natural evolution [18]. These networks are made up of neurons with different activation functions, including periodic functions like sine and symmetric functions like absolute value. These functions can be composed to produce patterns with properties observed in nature, such as bilateral and radial symmetry. In the HyperNEAT algorithm, CPPNs are extended to define neural networks [19][6]. The main insight was that while a CPPN with two inputs can define some characteristic at a single point, such as a pixel value for a pattern, a CPPN with four inputs can define a relationship between a pair of points. In HyperNEAT, CPPNs are used to define the weights of connections within a neural network. The fixed coordinated of neurons, known as a substrate, are then inputs into the CPPN. Then, to generate well performing neural networks, the CPPNs are modified through the evolutionary algorithm NEAT.

3 METHODOLOGY

Here we present a method for evolving SNNs using a generative encoding approach. First, we will describe the SNN model used in the study, then describe the algorithm for evolving the networks.

3.1 Spiking Neural Network Model

We utilize accumulate-and-fire neurons models and synapses with synaptic delays. To encode inputs values, inputs are converted into spike trains through a combination of binning and varying spike train length [13]. These spike trains are then applied to the input neurons in the SNN. Classification and control decisions are made by selecting the output neuron which has fired the most times in a given time frame. The SNNs in this model support complete recurrence, where any neuron can be connected to any other neuron, including inputs and outputs. The parameters of the SNNs to be optimized through the evolutionary algorithm are the connectivity patterns, synapse weights, delays, and neuron thresholds. These networks are compatible with the Caspian neuromorphic system [8], and thus can be directly deployed on that hardware without a mapping procedure.

3.2 Evolving SNNs using CPPNs

We use an evolutionary algorithm to evolve compositional pattern producing networks, which determine the connectivity patterns of the resulting SNNs. The format of our approach is as follows: an individual in our population is a CPPN, defined by its internal weights and activation functions. Any individual neuron in the CPPN can have any activation function from a defined set (sin, tanh, gauss, relu, identity). The activations and weights are represented in a linear genome, which defines one CPPN. To generate an SNN from the CPPN, we input into the CPPN every pair of coordinates for neurons, with the outputs defining the weight and synaptic delay of the resulting connection. SNN neuron firing thresholds are defined by a third output. The process of generating an SNN from a CPPN is shown in Figure1. In order to encourage sparsity in the network, outputs beneath a certain threshold result in no connection being formed. The positions of the SNN neurons, referred to as the substrate, are defined beforehand, and fixed for the duration of the evolutionary search. Once the connectivity for the SNN is defined, the network is constructed and evaluated on a given task, such as classification or control. Therefore, the fitness of a CPPN within our genetic algorithm is defined by the performance of the generated SNN on the specified task. Evolution of the CPPNs is done through the NEAT algorithm. Figure 2 demonstrates an overview of the full method.

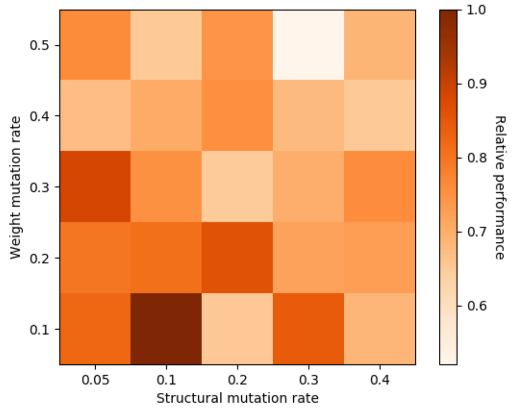


Figure 3: Effect of weight and structural mutation rates on algorithm performance. Higher performance is achieved using the low structural and weight mutation rates

4 RESULTS

To demonstrate applicability of the proposed approach, we apply it to multiple classification and control tasks. For control, we test the approach in the classic control environments available in the OpenAI gym[3]. For classification, we evaluated on the UCI wine and breast cancer datasets[1].

4.1 Effect of evolutionary hyper parameters on proposed method

We first explore the effect of several hyper parameters on the performance of the proposed method. We investigated the effect of the weight and structural mutation rates. The weight mutation rate is the independent probability of connection weights in the CPPN being mutated. The structural mutation rate is the probability of structural mutations; node addition, node removal, synapse addition, synapse removal. All other hyper parameters were held constant. We evaluated each set of hyper parameters 15 times on the cart-pole balancing task. The results of this experiment are shown in Figure 3. Best performance is achieved with a comparatively low weight mutation rate and low structural mutation rate. Indirect encodings can be more sensitive to mutation, since there is a layer of abstraction between genotype and phenotype. Consequently, a single structural mutation can impact a significant portion of the parameters of the expressed spiking neural network. This may explain why a low structural mutation rate produces better results.

	Feed Forward	Radial	p-value
Cart pole balancing	0.612	1.0	0.01
Mountain car	0.865	1.0	0.66
Acrobot	1.0	1.0	0.99
Cancer	1.0	1.0	0.99
Wine	0.95	1.0	0.82

Table 1: Relative performance of radial and feed forward neuron substrate configurations on multiple control and classification tasks

4.2 Effect of neuron substrate configurations

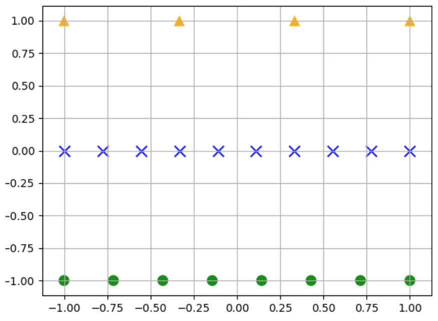
One advantage of using a CPPN encoding is that the neurons in the spiking neural network have defined locations. Because of this, relationships between input channels can be leveraged in the learned encoding. Therefore, it is expected that, for certain environments or problem spaces, a neuron configuration which exploits relationships in that space will outperform other neuron configurations. To test this, we examined the performance of multiple neuron configurations ten times across a variety of tasks. We compared the performance of a feed forward and radial neuron configuration, shown in Figure 4. The results of the comparison are shown in Table 1. P-values shown are derived from a two-tailed Welch's t-test, comparing 10 samples of each substrate configuration across multiple tasks. Across tasks, there is variation in the relative performance of each of the substrate configurations. While the radial substrate performs greater than or equal to the feed forward substrate across tasks, the degree varies significantly, indicating the task specific importance of the substrate.

4.3 Comparison with existing methods

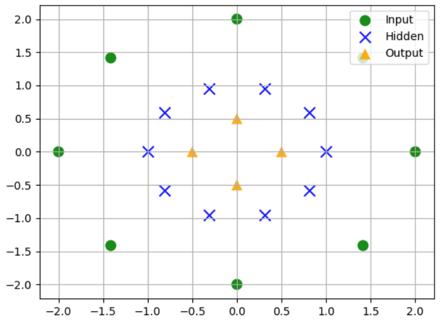
We compare the performance of the proposed method with EONS, another algorithm for evolving spiking neural networks. Performance of the two methods was compared on the cart-pole balancing task. Results of this comparison are shown in Figure 5. With population size of 500, both methods converge on the maximum fitness for the task, though EONS converges in fewer epochs.

5 DISCUSSION AND FUTURE WORK

In this work, we introduce an indirect encoding approach for generating spiking neural networks through an evolutionary algorithm. The purpose of the work was to demonstrate the feasibility and effectiveness of the method. The experiments shown demonstrate strengths and weaknesses of the approach and also identify paths for future research. One difficulty witnessed with the approach is the sensitivity to mutations the genetic encoding. Since the mutation of a single weight can impact the connectivity of a significant portion of the spiking neural network the hyper parameters must



(a) Feed forward neuron configuration



(b) Radial neuron configuration

Figure 4: Two possible neuron configurations, feed-forward and radial. Neuron configurations can be selected to suit a particular problem. The evolutionary algorithm learns the spatial connectivity pattern between the neurons in this space.

be carefully tuned to allow for an efficient search of the problem space.

One theoretical advantage of utilizing an indirect encoding is the ability to scale to large network sizes. In this work we only test networks with less than one hundred neurons. Therefore the large network scaling properties of using CPPNs to evolve spiking neural networks remain to be explored. Additionally, there are many other relationships between the unique characteristics of spiking neural networks and CPPNs that remain to be explored. For example, different strategies for encoding inputs may entail different neuron configurations for the substrate. There is also precedent in the literature for using Hyper-NEAT to encode a spatial pattern of STDP-like weight update rules [11]. Such a strategy could be applied in conjunction with the current approach to enable learning.

6 CONCLUSIONS

We demonstrate a indirect encoding approach for evolving spiking neural networks. This approach evolves compositional pattern producing networks which generate spiking neural networks that can perform across multiple classification and control tasks. Future work may exploit more relationships between HyperNEAT and SNNs and extend this method to more complex tasks.

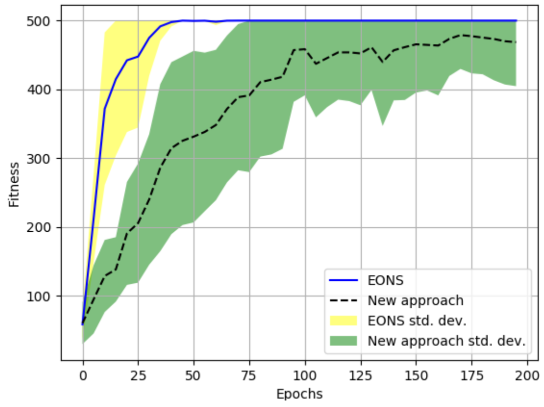


Figure 5: Comparison of EONS and proposed method on cart-pole balancing task. Lines show average max fitness value at each epoch over ten evolutionary runs for each algorithm. Shaded areas show the 1 standard deviation range for the max fitness values across multiple evolutionary runs at each epoch.

ACKNOWLEDGMENTS

This material is based 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.

REFERENCES

[1] Arthur Asuncion and David Newman. 2007. UCI machine learning repository.
[2] S.M. Bohte, J.N. Kok, and J.A. La Poutré. 2000. SpikeProp: backpropagation for networks of spiking neurons.. In *ESANN*. 419–424.
[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).
[4] S. Cawley, F. Morgan, B. McGinley, S. Pande, L. McDaid, S. Carrillo, and J. Harkin. 2011. Hardware spiking neural network prototyping and application. *Genetic Programming and Evolvable Machines* 12, 3 (2011), 257–280.
[5] Yang Dan and Mu-ming Poo. 2004. Spike timing-dependent plasticity of neural circuits. *Neuron* 44, 1 (2004), 23–30.
[6] David B D'Ambrosio, Jason Gauci, and Kenneth O Stanley. 2014. HyperNEAT: The first five years. In *Growing adaptive machines*. Springer, 159–185.
[7] N. Kasabov, V. Feigin, Z. Hou, Y. Chen, L. Liang, R. Krishnamurthi, M. Othman, and P. Parmar. 2014. Evolving spiking neural networks for personalised modelling, classification and prediction of spatio-temporal patterns with a case study on stroke. *Neurocomputing* 134 (2014), 269–279.
[8] J. Parker Mitchell, Catherine D. Schuman, Robert M. Patton, and Thomas E. Potok. 2020. Caspian: A Neuromorphic Development Platform. In *NICE: Neuro-Inspired Computational Elements (NICE)*. ACM, To appear.
[9] Milad Mozafari, Saeed Reza Kheradpisheh, Timothée Masquelier, Abbas Nowzari-Dalini, and Mohammad Ganjtabesh. 2018. First-spike-based visual categorization using reward-modulated STDP. *IEEE transactions on neural networks and learning systems* 29, 12 (2018), 6178–6190.
[10] Daniel Rasmussen. 2019. NengoDL: Combining deep learning and neuromorphic modelling methods. *Neuroinformatics* (2019), 1–18.
[11] Sebastian Risi and Kenneth O Stanley. 2010. Indirectly encoding neural plasticity as a pattern of local rules. In *International Conference on Simulation of Adaptive Behavior*. Springer, 533–543.
[12] Catherine D. schuman, J. Parker Mitchell, Robert M. Patton, Thomas E. Potok, and James S. Plank. 2020. Evolutionary Optimization for Neuromorphic Systems. In *NICE: Neuro-Inspired Computational Elements (NICE)*. ACM, To appear.
[13] Catherine D Schuman, James S Plank, Grant Bruer, and Jeremy Anantharaj. 2019. Non-Traditional Input Encoding Schemes for Spiking Neuromorphic Systems. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–10.
[14] Catherine D Schuman, James S Plank, Adam Disney, and John Reynolds. 2016. An evolutionary optimization framework for neural networks and neuromorphic architectures. In *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 145–154.

[15] Catherine D Schuman, Thomas E Potok, Robert M Patton, J Douglas Birdwell, Mark E Dean, Garrett S Rose, and James S Plank. 2017. A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint arXiv:1705.06963* (2017).

[16] William Severa, Craig M Vineyard, Ryan Dellana, Stephen J Verzi, and James B Aimone. 2018. Whetstone: A method for training deep artificial neural networks for binary communication. *arXiv preprint arXiv:1810.11521* (2018).

[17] Sumit Bam Shrestha and Garrick Orchard. 2018. Slayer: Spike layer error reassignment in time. In *Advances in Neural Information Processing Systems*. 1412–1421.

[18] Kenneth O Stanley. 2007. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines* 8, 2 (2007), 131–162.

[19] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. 2009. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life* 15, 2 (2009), 185–212.

[20] Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation* 10, 2 (2002), 99–127. https://doi.org/10.1162/106365602320169811