Evaluation of Spiking vs. Artificial Networks in Reinforcement Learning of Control Tasks

Andrei Cozma
Department of Electrical Engineering and Computer Science
The University of Tennessee, Knoxville
Knoxville, TN
acozma@vols.utk.edu

***Abstract*—The current state of Spiking Neural Networks (SNNs) has shown great potential in outperforming traditional Artificial Neural Networks (ANNs) in many ways. Biologically-inspired neuron models have also shown to offer improved learning capabilities, and neuromorphic implementations promise improved energy efficiency, scalability, and performance. This paper seeks to evaluate the performance and behaviors of spiking networks as compared to the classic networks for the OpenAI Gym CartPole environment, which is a classic real-time control problem. The experimental approach contributes in two main areas through the use of a stochastic policy gradient reinforcement learning method, as well as a recurrent spiking network architecture. The first experiment compared the performance between the two in the CartPole environment, with the SNNs significantly outperforming the ANNs in terms of average reward, but at the cost of increased variance in the scores between different training trials. The second experiment explores the effect of hyper-parameter tuning on the performance of the spiking network, which outperforms the classic network to an even greater degree and brings some improvements to the variability. Lastly, this work provides insights into potential ways to further improve the learning and generalization capabilities of SNNs in reinforcement learning problems in the future.**

I. Introduction

Artificial Neural Networks (*ANNs*) have gained popularity in machine learning and artificial intelligence research over the last two decades, leading to an explosion of software packages for their training, evaluation, and deployment. Neuromorphic computing is still a relatively new field that designs computer systems modeled after the human brain's neurons and synapses. The inherent complexity and computational demands of biologically plausible neuron models, such as Spiking Neural Networks (*SNNs*), have previously been a limiting factor preventing neural network simulation from reaching its full potential.

Spiking networks have great potential to replace traditional learning techniques in computational neuroscience and biologically plausible machine learning. *SNNs* consist of interconnected neurons that can fire action potentials in an event-based fashion. Each neuron represents a single bit of data and fires when a particular condition is met. Because biologically-inspired neurons naturally integrate their inputs over time, they are thought to be more suitable for tasks requiring temporal data processing [1]. In other words, spiking neurons are very good at remembering a short history of their activation. Finally, their binary communication with other neurons is an advantageous feature for performing fast, and energy-efficient simulations on neuromorphic hardware devices [2].

This makes spiking networks an intriguing alternative for many machine learning applications, particularly those requiring real-time processing or temporal data modeling. There has been a lot more research recently in evaluating the behaviors and performance of *SNNs* to regular *ANNs*, especially so in classification tasks. Spiking networks also seem suitable for learning non-deterministic control tasks due to their event-based nature and ability to replicate genuine biological neurons. Furthermore, spiking neurons' temporal representations of information are physiologically realistic, making them an excellent candidate for learning control tasks.

However, evaluating the overall performance, efficiency, and behaviors of spiking neural networks compared to traditional artificial neural networks in various learning tasks such as reinforcement learning is still an active area of research. This paper seeks to explore some of these topics by comparing *SNNs* and traditional *ANNs* on the *OpenAI Gym - CartPole* environment in an attempt to answer how future agents constructed on the same basis as human brains may or may not perform as well as their optimized counterparts. The intended goal is to demonstrate that spiking networks are a feasible alternative to the traditional neural networks for real-time event-based reinforcement learning problems.

II. Related Work

A. Reinforcement Learning

Reinforcement Learning (RL) is a training approach that rewards positive actions while penalizing undesirable ones. In principle, an agent may read data from its surroundings, perform actions, and learn via trial and error while maximizing some measure of the agent's overall reward. At each stage, the agent performs an action, at which point the environment provides it with feedback in the form of a new observation and a reward. An RL algorithm is fundamentally concerned with creating decision sequences in this environment, formalized in literature as a partially observable Markov decision process [3]. General approaches such as *Policy Gradient*, *Q-learning*, and *Actor-Critic* methods have the potential to produce high performance when applied to fairly complex environments.

Identify applicable funding agency here. If none, delete this.

B. OpenAI Gym

The *OpenAI Gym* [4] is a Python library conceived as a result of the multitude of recent advances in Reinforcement learning and the necessity for a platform for exercising AI agents in simulated environments. It is designed to provide a common interface for the training and testing of reinforcement learning algorithms as it provides a variety of different environments in which the agent may be tested.

C. Spiking Networks in the Gym

While numerous research papers evaluate traditional neural networks in the OpenAI Gym environments, there are currently very few studies evaluating *SNNs* in OpenAI Gym environments. Previous work has shown an evaluation of spiking networks in deterministic environments with a Q-Learning approach [5]. For non-deterministic environments, another publication seeks to address questions regarding the strengths and limitations of Leaky-Integrate-and-Fire (LIF) spiking neuron ensembles using the Nengo library for application in OpenAI virtual environments such as *CartPole*, MountainCar, and LunarLander [6].

This previous work investigates several research questions, including how to represent arbitrary environmental signals from multiple senses and choose between equally viable actions in a given scenario, leading to an analysis of future possibilities to create a generic model that can learn and operate in various situations [6]. Their experiments on the *CartPole* environment used a multi-ensemble network composed of 500 neurons each, one for the stimulus (observations) and the other for the controls (actions). The authors conducted trials of several different learning methods for a transformation function, starting with general heuristics and eventually resorting to a Simulated Annealing approach to try to find a close approximation to state-action mapping [6]. While the authors did not provide results in terms of the accumulated reward over the time of the simulation, their work laid great insights and foundations on which the work presented in this paper builds.

D. Contributions

More specifically, one of the main contributions exemplified later in this paper is in terms of the learning method for state-action mapping. This approach is fundamentally different because it uses a stochastic policy optimization method. In policy optimization methods, the agent learns directly the policy function that maps state to action, and the output is a probability distribution over actions from a given state, often referred to as a *Partially Observable Markov Decision Process* [3].

The next significant contribution is in the network architecture. Instead of using a multi-ensemble network, the goal is to use a recurrent type of spiking network. As a result, the recurrent approach enables the network to adjust its spike sequences to match the distribution of sequences created by the encoded observations [7]. Previous work has also proposed the use of recurrent spiking networks to solve planning problems and provided a solid theory for doing so based on the framework of probabilistic inference [7]. More details about the implementation are discussed in the next section.

III. Experimental Setup

The following section describes the experimental setup used to compare the performance of spiking and artificial neural networks in the *OpenAI Gym - CartPole* environment. It starts by describing the libraries used to implement the overall experiment, including the implementation details for the learning method and the two different networks.

A. Library Choice

The choice of the library used in this experiment is an important part of the methodology that needs to be carefully considered. Many different libraries are currently available and in active research and development for simulating, training, and evaluating biologically-inspired neuron models and learning algorithms. While the authors used Nengo in the previous work discussed, the library is primarily designed to be used as a simulator, which proved to raise various challenges in previous work, which required various workarounds to be able to interface properly with the OpenAI Gym environments [6]. Several other libraries have been considered for a final choice, including *Norse* [8], *BindsNET* [9], and *The TENNlab Exploratory Neuromorphic Computing Framework* [10] [11].

The main decision factors for choosing a library is based on the ability to train on the GPU, available implementation for the recurrent cell with LIF neurons, and integration with other commonly used libraries. The Norse [8] library was finally chosen as the go-to for these experiments. Its API uses PyTorch to implement primitives for various bio-inspired neuron models, synapse dynamics, encoding, and decoding algorithms. Most importantly, it contains the desired implementation of a recurrent cell of LIF spiking neurons and several encoding methods to choose from. Further, the use of PyTorch as the back-end for Norse allows several advantages, including a familiar API, GPU-accelerated training, and seamless integration with PyTorch's ecosystem, including optimizers, schedulers, and other utilities.

B. Environment Details

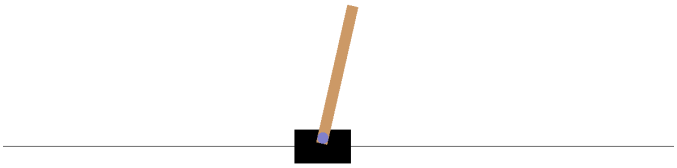

Fig. 1. OpenAI Gym - CartPole Environment

The *OpenAI Gym CartPole-v1* environment shown in Figure 1 was used for the course of the experiments. In the *CartPole* environment, a pole is attached to a cart that runs along a frictionless track. The goal is to keep the pendulum upright for as long as possible. Controlling the mechanism is as simple as delivering a force of *+1 or -1* to the cart. Every step the pole remains straight results in a reward of 1. An episode terminates when either the pole's angle to the cart is more than 15 degrees from vertical or the cart runs out of the bounds of the viewport [4].

The action space is a discrete value that indicates the direction of the force pushing the cart. The action 0 indicates that the cart is being pushed to the left, while a value of 1 indicates that the cart is being pushed to the right. The observation space is a four-dimensional vector consisting of the cart's position, velocity, angle, and angular velocity [4].

C. Learning Method

A custom module is implemented to accommodate the use of both *SNN* and *ANN* policy architectures in a unified approach for simulation, training, logging of metrics, saving model checkpoints, and performing hyper-parameter tuning. The Policy Gradient approach mentioned in the previous section is used as the learning technique for both the *ANN* and *SNN* implementations.

The implementation for the training loop runs the simulation for a specified number of episodes. In these experiments, the models are trained for 250 episodes on an *NVIDIA RTX 3070* GPU. For each episode, the simulation runs for 750 steps.

At each step, the action selection is implemented as follows:

1) The current state of the environment is passed through the policy specified as a configuration argument (either *SNN* or *ANN*) to retrieve a vector of the probabilities.
2) The probabilities are then used to construct a categorical distribution and sample the action to take.
3) Finally, the action is provided to the environment, and the reward is kept track of for later use when calculating the loss at the end of each episode.

At the end of each episode, the calculation for the loss is implemented as follows:

1) The saved rewards and the discount factor (*Gamma*) are used to calculate the discounted return for each time step.
2) The sequence of discounted returns is then normalized and the loss is calculated as a weighted sum of the returns.
3) Finally, one step of back-propagation is performed with the calculated loss using the Adam optimizer module from PyTorch.

D. Policy Implementations

The *ANN policy* is implemented as a standard feed-forward neural network with two hidden layers. In the implementation of the feed-forward function, each hidden layer is activated using the *ReLU* activation function. Dropout is also used to prevent any potential over-fitting. Finally, the output from the last layer is run through a *Softmax* activation function.

The *SNN policy* uses three primary components of the *Norse* library. The neuron model of choice is a Leaky-Integrate-and-Fire neuron since they provide a good balance between biological inspiration and complexity. The Norse *Constant Current LIF Encoder* [8] module encodes inputs as fixed (constant) voltage currents and mimic spikes that occur across several time-steps/iterations. The encoded inputs are then processed through a sequential integration loop based on the shape of the observation space.

Following the encoding of the inputs, they are routed through a *LIF Recurrent Cell* for each iteration to keep track of the state of the neurons and execute the integration. Specifically, the cell executes one Euler-integration step of a *Leaky Integrate-and-Fire (LIF)* neuron model with recurrence but no time [8]. To read the voltages, they are passed through a linearly weighted cell and then stored in a tensor, where the first dimension reflects the time steps of the integration loop. Finally, the output is computed by first obtaining the maximum voltages throughout the time-step axis and passing them through a *Softmax* activation function to get the probabilities for the policy gradient.

E. Experiments & Configurations

The first experiment aims to find a set of hyper-parameters that works decently well for both the *SNN* and the *ANN* simultaneously. The initial configurations used are based on manual tweaking of parameters such as the learning rate, the number of neurons, the dropout rate, as well as the discount rate (gamma). A total of 100 trials of training are executed for a maximum of 250 episodes, each using the same parameter configuration described below.

For the initial configurations of the *ANN* models, the parameters used are as follows:

- Learning Rate: 0.05
- Hidden Layer 1 Neurons: 128
- Hidden Layer 2 Neurons: 128
- Dropout Rate: 0.5

- Gamma: 0.97

For the initial configurations of the *SNN* models, the parameters used are as follows:

- Learning Rate: 0.05
- Hidden State Neurons: 128
- Dropout Rate: 0.5
- Gamma: 0.97

It is important to note that at this stage, the optimizer is run with an adaptive learning rate approach that gradually decreases the value when the running reward starts to plateau. The intuition behind this approach is that allowing the learning rate to decay would allow the model to continue learning in deeper valleys of the loss function rather than either getting stuck or completely diverging.

The second experiment intends to further optimize the potential of the *SNN* through hyper-parameter tuning. The aim is to investigate how sensitive the *SNN* is to parameter choices and to evaluate the effects these on the overall behavior of the network. In this experiment, the adaptive learning rate is turned off in order to investigate the effect of different fixed learning rates.

Each parameter is tuned individually in the order in which they are listed below, with ten training trials for each value for a maximum of 250 episodes. When varying one of the parameters, all of the others are kept fixed at a default value. The default value initially starts as the corresponding parameter values in the first experiment. When moving on to the next parameter, the best performing value for all the previous parameters is used as the default in order to converge to the best possible set. Finally, the best set of parameters is used to train a final set of trials in the same fashion as they are executed in the first experiment.

The ranges for the parameters are selected as follows:

1) LR: 0.015, 0.0125, 0.01, 0.0075, 0.005, 0.001, 0.0005, 0.0001
2) Hidden State Neurons: 8, 16, 32, 64
3) Dropout: 0.25, 0.5, 0.75
4) Gamma: 0.95, 0.97, 0.99

F. Data Collection and Graphs

Weights & Biases [12] is the platform of choice for the collection of data from these experiments. Each run is logged together with all the details needed for further analysis, such as the environment name, policy name, and hyper-parameters used. The total episode reward is logged, the policy loss, the calculated running average of reward, and the number of simulation steps for each episode.

Several types of graphs are generated and analyzed from the data collected to enable interpretation of the differences between the results. The first type of graph is a line graph depicting the running average of the total reward over the number of training episodes for the top 5 best models. This graph is helpful for observing if the model is indeed learning and if there are any drastic changes in reward averages.

The second type of graph is a line graph depicting both the mean and standard deviation of the rewards over the total number of training episodes. The mean and standard deviation for each episode is calculated from a pool of 20 trained models. The standard deviation graph is helpful to determine how much performance could differ between multiple trained instances of the policy with the same configuration.

The third type of graph is a box plot which shows the comparison between different hyper-parameter values. This graph is useful to determine which parameter values provide desirable results and behavior in terms of mean and standard deviation of the reward.

IV. Results

A. Initial Configuration Results

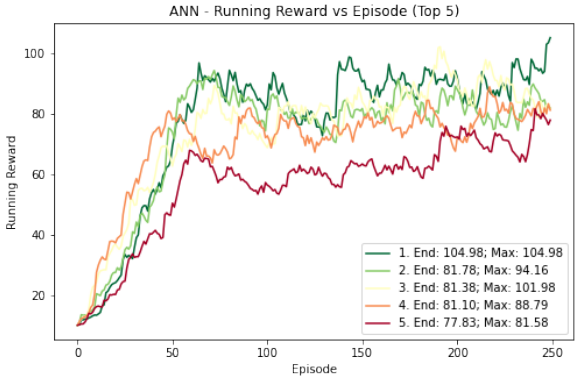

Fig. 2. *ANN* - Top 5 Runs for Initial Configuration

Figure 2 depicts the running average of the total reward for the Top 5 best training runs of the *ANN*. Model #1 is able to achieve a running average score of *104.98*. The results look fairly consistent across the top 5 best models, which indicates a relatively low standard deviation between different runs. Further, the performance started to plateau after around 75 episodes consistently for all 5 best models.

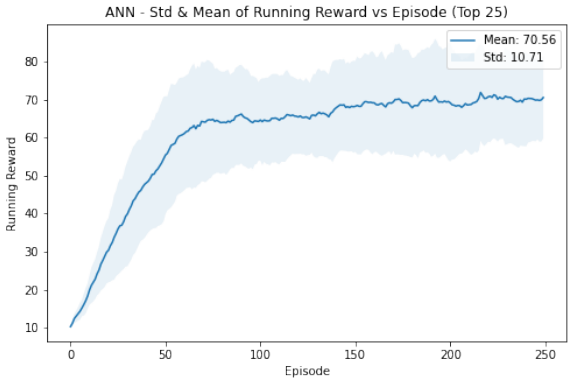

Fig. 3. *ANN* - Mean & Std. for Initial Configuration

Figure 3 further investigates the performance differences between the Top 25 training runs for the *ANN* in terms of

mean and standard deviation. The value for the mean at the end of the 250 episodes is *76.98*, and the standard deviation is 9.10. This confirms the previous assumption of a relatively low standard deviation.

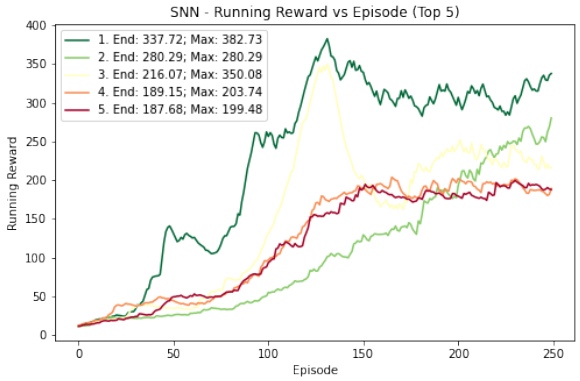

Fig. 4. *SNN* - Top 5 Runs for Initial Configuration

Figure 4 depicts the running average of the total reward for the Top 5 best training runs of the *SNN*. Model #1 is able to achieve a running average score of *382.73* at around 125 episodes, but the performance degraded slightly down to *337.72* after 250 episodes. It can be observed that the results seem a lot more varied between the 5 models. Interestingly, model #3 achieved a high score of *350.08* at around 125 episodes, but the performance degraded significantly afterward. Other models had a more steady increase in performance which indicates that the models could have achieved even better performance if they had been trained for longer.

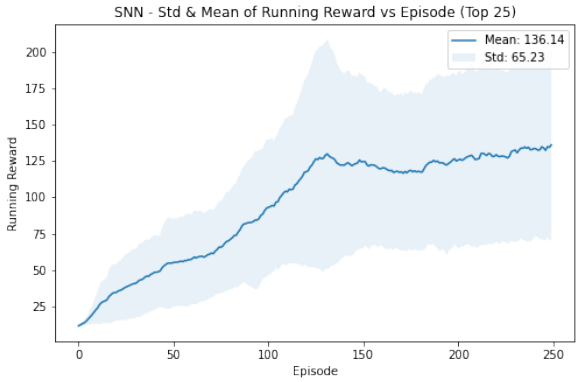

Fig. 5. *SNN* - Mean & Std. for Initial Configuration

Figure 5 further investigates the performance differences between the Top 25 training runs for the *SNN* in terms of mean and standard deviation. The value for the mean at the end of the 250 episodes is *136.14*, and the standard deviation is *65.23*. It can be seen that, on average, the *SNNs* offered more than double the performance as compared to the *ANNs*. However, the standard deviation between the spiking is more than 6 times higher than that of the traditional networks.

B. Hyper-parameter Tuning Results

The following are the results of the hyper-parameter tuning process for, which sought to push the boundaries of the *SNN* by further optimizing the learning rate, number of hidden state neurons, dropout rate, and gamma parameters:

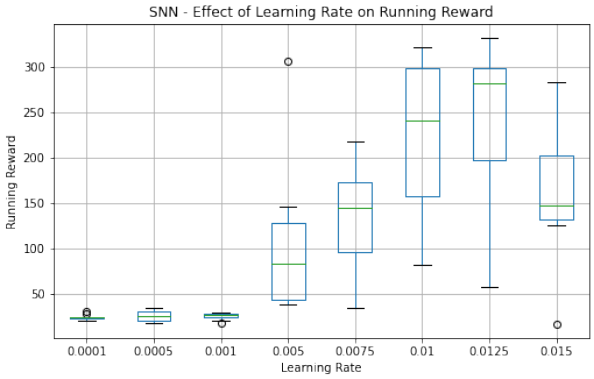

Fig. 6. *SNN* Tuning - Effect of Learning Rate

The learning rate was the first parameter chosen to be optimized in order to provide a solid and consistent foundation for optimizing the rest of the parameters. Each value is tested 10 times to evaluate the results of the differences between them. Figure 6 depicts the results as a box plot where the X-axis is the respective learning rate, one of: 0.015, 0.0125, 0.01, 0.0075, 0.005, 0.001, 0.0005, 0.0001. The Y-axis is the final Running Reward after 250 episodes of training. The results show that values below 0.005 performed extremely poorly, and the best overall value was *0.0125*. Values higher than this also seemed to degrade in performance.

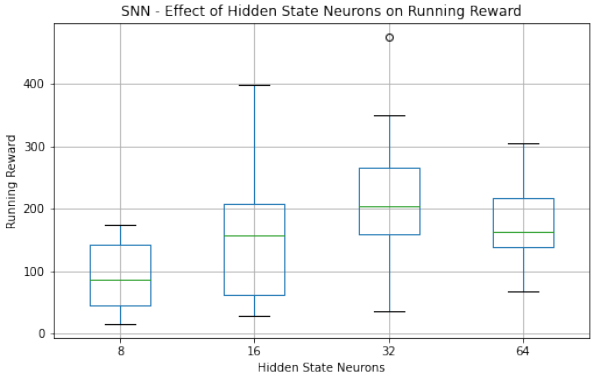

Fig. 7. *SNN* Tuning - Effect of Hidden State Size

Spiking networks are believed to favor a smaller number of neurons as compared to the higher number of *ANNs* generally used in *ANNs*. Since the first experiment used *128 neurons*, this experiment will use a range of smaller values to test this theory. Figure 7 depicts a box plot where the X-axis is the number of neurons in the recurrent hidden state: 8, 16,

32, 64. Each value is tested a total of 10 times to provide a solid comparison. The Y-axis denotes the final Running Reward after 250 episodes of training. Overall, a value of 32 performed the best, with moderate standard deviation between runs, which confirms the idea that spiking networks, due to their properties discussed earlier, would perform nicely with fewer neurons.

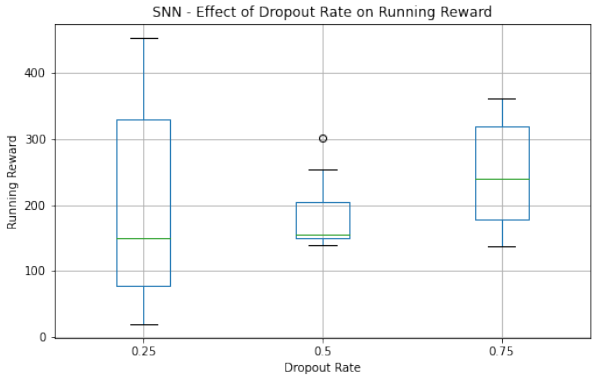

Fig. 8. *SNN* Tuning - Effect of Dropout Rate

Figure 8 depicts a box plot where the X-axis is the dropout rate with values 0.25, 0.5, 0.75. Similar to previous parameters, each value is tested a total of 10 times. The Y-axis is the final Running Reward after 250 episodes of training. The overall best value was *0.75*, which performed comparatively better than lower values in terms of the mean. The variance was decent, but overall much better than the low values of 0.25. Perhaps a value between 0.5 and 0.75 could be used to receive the desired balance between reprehensibility and performance.

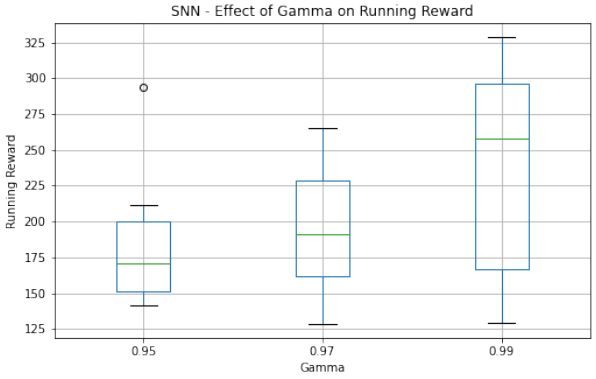

Fig. 9. *SNN* Tuning - Effect of Discount Factor (Gamma)

The discount factor (gamma) determines the emphasis that the agent puts on future rewards. Common values used for this parameter are in the range of 0.95 to *1*. Figure 9 depicts a box plot where the X-axis is the discount factor (*Gamma*) with values 0.95, 0.97, 0.99, and each value is tested 10 times. The Y-axis is the final Running Reward after 250 episodes of training. A smaller value represents a shorter horizon and, therefore, less emphasis on future rewards. While the best values are problem-specific, in this case, the highest performance was achieved with a gamma value of *0.99*.

C. Improved SNN Results

The best parameters showcased in the previous results were selected, and 25 tests were run for a total of 250 episodes. In summary, the parameters are as follows:

- LR: 0.0125
- Hidden State Neurons: 32
- Dropout: 0.75
- Gamma: 0.99

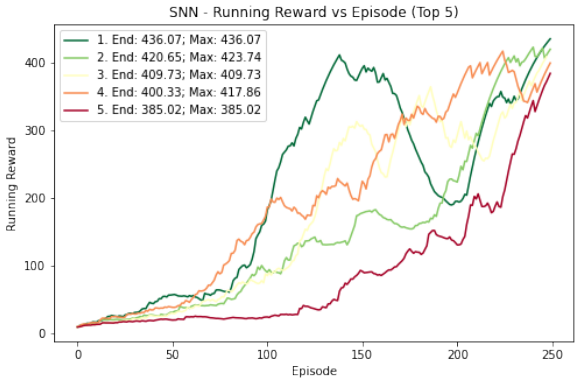

Fig. 10. *SNN* #2 - Top 5 Runs After Parameter Tuning

Figure 10 depicts the running average of the total reward for the Top 5 best training runs of the *SNN*. Model #1 is able to achieve a running average an impressive score of *436.07* at the end of the 250 episodes, which is an improvement of around 100 points. It can also be observed that the results seem a lot less varied and overall more consistent between the 5 models as compared to the initial configuration in the first experiment. Overall, all 5 top models ended their run with comparably similar scores, and it looks like they could have reached even higher scores if they were trained for more than 250 episodes.

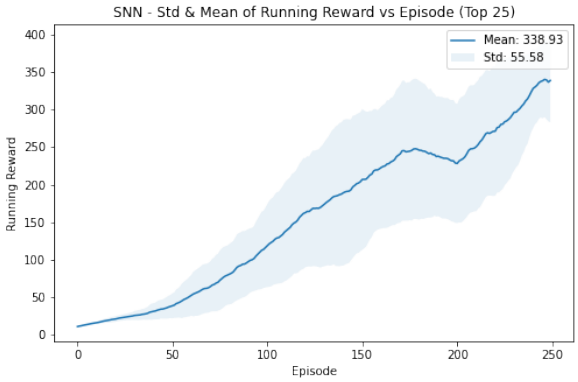

Fig. 11. *SNN* # 2 - Mean & Std. After Parameter Tuning

Figure 11 further investigates the improvements offered by optimized *SNN* configuration given by the mean and standard deviation across 25 runs of training. The value for the mean at the end of the 250 episodes is *338.93*, and the standard deviation is *55.58*. Comparatively, the mean was *136.14* and standard deviation of *65.23* prior to hyper-parameter tuning. Optimized parameters offered an impressive *200-point increase* in the mean reward, and around a *10-point decrease* in standard deviation. While the prior chart indicated a continually increasing deviation as the model is trained for more episodes, this time the standard deviation actually started to decrease after around 175 episodes.

V. Discussion, Conclusion, and Future Work

The spiking neural network architectures evaluated in this report provided impressive results and outperformed the traditional artificial neural networks in the *CartPole* environment. The mean reward value across many trained models is more than twice as high as that of the *ANNs*. While the *ANNs* had lower overall average scores, they proved to be more consistent, given the much better variance in scores among different tests.

While it does seem like some of the *SNN* trials could have been trained for longer than 250 episodes, others seemed to have reached their maximum potential earlier on. Because of this, many of the models started to over-fit relatively early in the training process, resulting in significantly higher variance in scores between different runs. This indicates that either the learning method employed, the properties of *SNNs*, or both, made it very sensitive to parameter choices and the random initialization of the weights. Hyper-parameter tuning performed in the second experiment seemed to play a huge role in tackling this issue, with the only caveat being significantly longer training time and extensive experimentation to alleviate this problem. This behavior has been consistent with results in previous work [5].

The evaluation results for the *ANN* policy show that the rate at which the performance improved saturated pretty early on. This is a strong indicator that the *ANNs* would likely benefit from not only a deeper network topology but also an increased number of neurons. Interestingly, while bigger and deeper networks often provide better results in traditional networks, this may not be true to the same extent in their biologically-plausible counterparts. Overall, the *SNNs* favored significantly fewer neurons than the traditional networks, which is highly advantageous, especially in neuromorphic hardware implementations. It is surprising that the very common thought of requiring a deeper network actually turned out to be the complete opposite.

Previous works discuss how the translation of numerical data into spikes suitable for use as input to a *SNN* often poses a significant barrier for most applications, and the conversion process is often non-trivial and highly specific to the application [13]. Early trials with different libraries, architectures, and environments confirmed the several challenges *SNNs* face in terms of the encoding and decoding of the observations and actions. This resulted in resorting to a generally simplistic encoding method due to limiting time constraints. Therefore, more work is needed to explore the effects of different encoding and decoding approaches and their effect on the capabilities of the *SNN*.

More complex encoding and decoding schemes can significantly affect application performance, and therefore, further improvements could be made in this regard. Further, The encoding strategy used can impact not just the effectiveness and accuracy of the system but also the rate at which data can be processed, and the system's energy efficiency, which is especially significant for neuromorphic implementations [13]. For a generalized approach that could handle multiple different environments, the representation problem would include more robust encoding and decoding methods, a multi-ensemble network architecture formed to manage the projected dimensionality, and regular online learning to deal with changing conditions [6].

Next, using a multi-ensemble architecture with a similar optimization approach could intuitively boost the capability of learning, and similarly with perhaps a recurrent implementation of the traditional network. Nonetheless, a multi-ensemble approach would be more realistic to how different parts of the human brain interact together in the decision-making process. However, this also raises further questions about how to best handle the optimization approach similarly to how it is employed in this project. One inciting possibility is the use of evolutionary algorithms as a biologically-plausible optimization method. In terms of improvement for the overall learning method, this could also be compounded with a *Q-Learning* or *Actor-Critic* reinforcement learning approach rather than the policy gradient approach in future work.

In conclusion, spiking networks are shown to possess highly efficient learning capabilities and have a great potential to improve upon the classic networks in many ways. This paper also discusses several insights into potential ways to further improve the learning and generalization capabilities of *SNNs* in reinforcement learning problems in future work. One of the still limiting factors, as discussed earlier, is the complexity of the biologically-realistic neuron models as compared to the highly generalized and optimized traditional neurons. Continued research efforts are needed to unlock the full potential of *SNNs* to eventually enable learning capabilities comparable to the human brain.

References

[1] A. Grüning and S. M. Bohte, "Spiking neural networks: Principles and challenges." in *ESANN*. Citeseer, 2014.

[2] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers in Neuroscience*, vol. 12, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.00774

[3] J. Zhang, A. Koppel, A. S. Bedi, C. Szepesvari, and M. Wang, "Variational policy gradient method for reinforcement learning with general utilities," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 4572–4583. [Online]. Available:

https://proceedings.neurips.cc/paper/2020/file/30ee748d38e21392de740e2f9dc686b6-Paper.pdf
[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016. [Online]. Available: https://arxiv.org/abs/1606.01540
[5] D. Chen, P. Peng, T. Huang, and Y. Tian, "Deep reinforcement learning with spiking q-learning," 2022. [Online]. Available: https://arxiv.org/abs/2201.09754
[6] C. Peters, T. C. Stewart, R. L. West, and B. Esfandiari, "Dynamic action selection in openai using spiking neural networks," in *FLAIRS Conference*, 2019.
[7] D. Rezende, D. Wierstra, and W. Gerstner, "Variational learning for recurrent spiking networks," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Eds., vol. 24. Curran Associates, Inc., 2011. [Online]. Available: https://proceedings.neurips.cc/paper/2011/file/069059b7ef840f0c74a814ec9237b6ec-Paper.pdf
[8] C. Pehle and J. E. Pedersen, "Norse - A deep learning library for spiking neural networks," Jan. 2021, documentation: https://norse.ai/docs/. [Online]. Available: https://doi.org/10.5281/zenodo.4422025
[9] H. Hazan, D. J. Saunders, H. Khan, D. Patel, D. T. Sanghavi, H. T. Siegelmann, and R. Kozma, "Bindsnet: A machine learning-oriented spiking neural networks library in python," *Frontiers in Neuroinformatics*, vol. 12, p. 89, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fninf.2018.00089
[10] J. S. Plank, C. D. Schuman, G. Bruer, M. E. Dean, and G. S. Rose, "The tennlab exploratory neuromorphic computing framework," *IEEE Letters of the Computer Society*, vol. 1, no. 2, pp. 17–20, 2018.
[11] S. R. Kulkarni, M. Parsa, J. P. Mitchell, and C. D. Schuman, "Benchmarking the performance of neuromorphic and spiking neural network simulators," *Neurocomputing*, vol. 447, pp. 145–160, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231221003969
[12] L. Biewald, "Experiment tracking with weights and biases," 2020, software available from wandb.com. [Online]. Available: https://www.wandb.com/
[13] C. D. Schuman, J. S. Plank, G. Bruer, and J. Anantharaj, "Non-traditional input encoding schemes for spiking neuromorphic systems," in *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019, pp. 1–10.