# Playing Atari using Spiking Neural Networks

COMPSCI 696: Independent study report
Devdhar Patel
Faculty advisor: Dr. Robert Kozma
University of Massachusetts Amherst

## Introduction

In recent years, deep learning has achieved outstanding results in various machine learning tasks [12, 17, 19]. Most of these results have been achieved by static neurons that produce analog output as opposed to biological neurons which are dynamic and produce discrete output. However, training such deep neural networks requires repeated redundant updates of many neurons which requires time and energy.

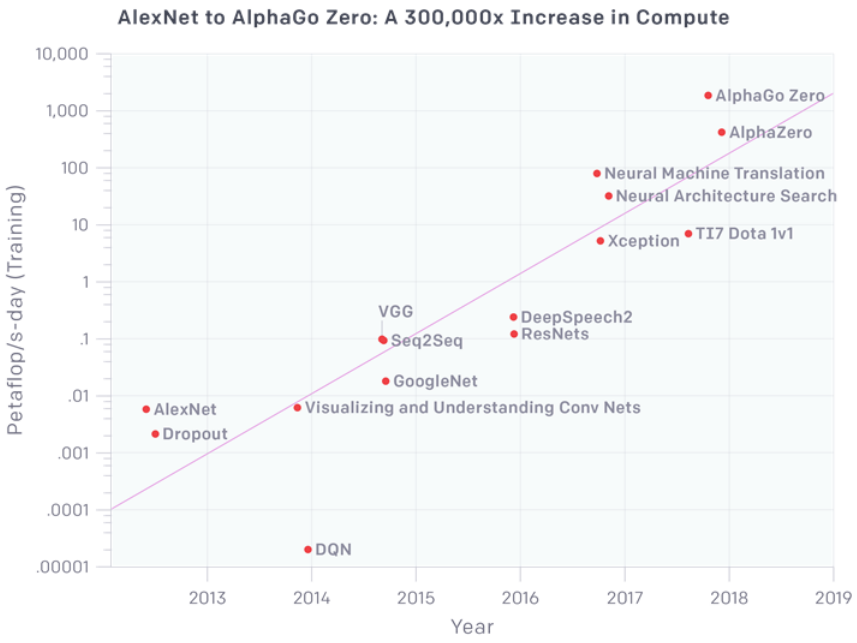



Figure 1: There has been exponential growth in the training runs for artificial intelligence algorithms. The algorithms topping the compute requirements (AlphaGo Zero and AlphaZero) are reinforcement learning algorithms. This graph was published in a recent article by OpenAI [6].

Spiking neurons [26, 14] are models on biological neurons and spiking neural

networks (SNNs) are considered to be the third generation of neural networks [13]. Spiking neurons, when implemented on spike-based neuromorphic hardware, are more energy efficient[5, 15] than static neurons and SNNs can learn from event based, data driven updates. These characteristics of spiking neurons make them a better choice than static neurons for real time applications.

In recent years, there have been many breakthroughs in reinforcement learning tasks [17, 22, 23, 16], however, many of reinforcement learning algorithms are hard to train and require long training periods. We believe that spiking neurons have the potential to learn reinforcement learning tasks faster and perform better with less training.

To that end, we test the performance of SNN on the game of Atari 2600 Breakout. Spiking neural networks are hard to train using gradient descent due to the non-differentiable nature of the spike. Therefore, we train an artificial neural network (ANN) with similar network architecture using the DQN algorithm [17] and transfer the learned weights to the SNN. We show that SNNs are capable of outperforming ANNs with similar weights and network architecture.

# Related Work

There has been a lot of recent work on the ANN-to-SNN conversion. Perze-Carrasco et al. (2013) [18] first introduced the idea of converting CNN to spiking neurons with the aim of processing inputs from event-based sensors. Cao et al. (2015)[4] suggested that frequency of spikes of the spiking neuron is closely related to the activations of a rectified linear unit (ReLU) and reported good performance on computer vision benchmarks. Diehl et al. (2015)[8] proposed a method of weight normalization that rescales the weights of the SNN to reduce the errors due to excessive or too little firining of neurons. They also showed near loss-less conversion of ANNs for the MNIST classification task. Rueckauer et al. (2016 and 2017)[20, 21] demonstrated spiking equivalents of a variety of common operations used in deep convolutional networks like max-pooling, SoftMax, batch-normalization and inception modules. This allowed them to convert popular CNN architectures like VGG-16, Inception-V3, BinaryNet, etc. They achieved near loss-less conversion of these networks. There has been no previous work on conversion of Deep Q-networks into spiking neural networks to our knowledge.

# Background

## Deep Q-network

The Deep Q-network (DQN) [17] is a reinforcement learning algorithm that has surpassed human performance on many Atari 2600 games. It uses a neural network that can learn policies from only the pixels of the screen and the game

score.

The network approximates the optimal action-value function (equation 1) and the weights of the network are updated using stochastic gradient descent.

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t = s, a_t = a, \pi] \tag{1}$$

Here, $r_t$ is the reward gained by performing the action $a_t$ at the state $s_t$. After taking the action $a_t$, the state of the environment changes to $s_{t+1}$. The action-value function $Q^*$ is the maximum sum of rewards $r_t$ gained at each time step $t$ achievable by a behaviour policy $\pi = P(a/s)$ after taking action $a$ at state $s$. The reward is discounted by a factor of $\gamma$ at each time step to incentivize the agent to pick rewards earlier.

The action-value function is a Bellman equation and can be written as:

$$Q(s_t, a_t) = r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \tag{2}$$

Therefore, for every transition (state, action, reward, next state), we can calculate the difference between the estimates of current state $s_t$ and the next state $s_{t+1}$ (loss) as:

$$L = r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{3}$$

We can use this loss to update the network weights using stochastic gradient descent.

## Atari 2600 Breakout

We choose the Atari 2600 game Breakout as the reinforcement learning environment. The DQN algorithm performs very well on Breakout gaining reward 12 times higher than the human performance. Therefore, we use the breakout game as a benchmark to test if spiking neuron can be used with the same learning condition as the ANN counterpart.

The game screen consists 6 layers of bricks on the upper half of the screen. A ball travels across the screen bouncing of the walls of the screen. When the ball hits a brick, the ball bounces off the brick and the brick is destroyed. If the ball falls off the bottom of the screen, the player loses a life. To prevent this from happening, the player has to control a paddle that can move left and right and is located at the bottom of the screen. The ball can bounce off the paddle. Every time the ball hits a brick, the score of the game is increased. More points are gained for destroying the upper layers of bricks than the lower layers. The objective of the game is to destroy all the bricks. The player has 5 lives at the start of the game. The game is over if the player has no lives left. The agent has 4 possible actions to choose from at each step:


Figure 2: Atari 2600 Breakout

1. No action: Do nothing.
2. Fire: This action has to be selected at the start of each game and after losing a life to start the game.
3. Right: Move the paddle right.
4. Left: Move the paddle left.

We use the OpenAI gym environment [3] to simulate the game environment. Specifically, we use the 'BreakoutDeterministic-v4' environment that shows every $4^{th}$ frame so that the agent sees and picks and action every 4 frame. The last selected action is repeated on the skipped frames. This is done in order to speed up the training process.

# Methods

## Weights transfer

Training of deep spiking networks typically does not use spike-based learning rules. But instead, a conventional ANN is trained using backpropogation and then converted to SNN. Much of the recent work on deep spiking networks has been focused image classification [4, 7]. We use this method of training in the reinforcement learning domain. We train an ANN to play Breakout using a DQN and transfer the trained weights to a SNN with similar network architecture.

## Network architecture

Typically, the network used to train on Atari games using the DQN algorithm consists of multiple convolutional layers followed by fully connected layers [17]. However, to reduce the complexity of the network and reduce the amount of parameters, we choose a fully connected network with one hidden layer.

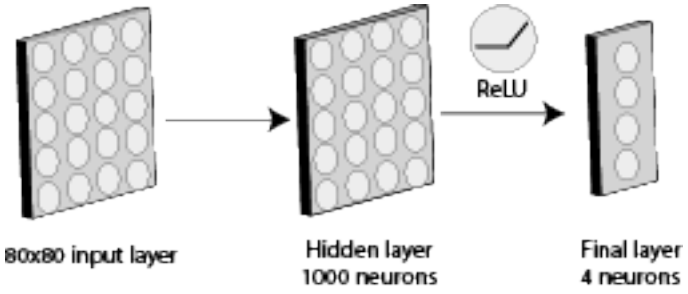



Figure 3: Artificial neural network architecture

Figure 3 shows the network architecture of the ANN. The ANN consists of 80x80 pixel input followed by a fully connected hidden layer with 1000 ReLU neurons. The output layer is a fully connected layer with 4 neurons that give the estimate of the optimal action-value of each of the 4 possible actions. The table 1 lists the hyperparameters and their values for training the ANN with the DQN algorithm.

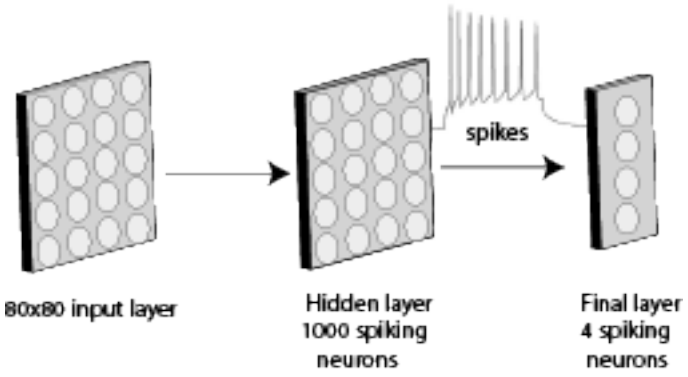



Figure 4: Spiking neural network architecture

Similarly, the SNN consists of 80x80 input neurons followed by a fully connected hidden layer of 1000 leaky integrate and fire (LIF) neurons [1] or LIF neurons with adaptive thresholds [11]. The final layer is a fully connected layer with 4 LIF neurons. Table 2 lists the hyperparameters and their values for the spiking neurons. The SNN runs for 500ms for each input during which time the

spikes for each neuron in the output layer is summed. While this value may be a lot higher than the actual estimate of the optimal action-value, we can still use it to test the network since we only need to pick the action with the highest value and the magnitude of the value does not impact this. We use the BindsNET package [9] to simulate the spiking neural networks.

| **Hyperparameter** | **Value** | **Description** |
|---|---|---|
| Training Length | 30000 episodes | Number of games the agent trains over. |
| Mini-batch size | 32 | Number of transitions sampled from the replay memory for which each stochastic gradient descent update is computed. |
| Replay memory size | 200000 | Number of most recent transitions saved in the replay memory. |
| Replay memory init size | 50000 | Initial size of the replay memory filled by using random actions. |
| Agent history length | 4 | The number of most recent frames that are given to the agent as the input. |
| Target network update frequency | 10000 | Number of updates on the network in between each target network update. |
| Discount factor | 0.99 | The discount factor $\gamma$ used for estimating the optimal action-value. |
| Frame skip | 3 | Number of frames skipped in between each state. |
| Action repeat | 4 | Number of times each selected action is repeated. |
| Update frequency | 4 | Number of actions selected by the agent in between each stochastic gradient descent update. |
| Update rule | RMSProp | The rule for each weight update[25]. |
| Learning rate | 0.00025 | The learning rate for RMSProp. |

. . . continued

| **Hyperparameter** | **Value** | **Description** |
|---|---|---|
| Gradient momentum | 0.95 | The gradient momentum for RMSProp. |
| Squared gradient momentum | 0.95 | The squared gradient momentum for RMSProp. |
| Min squared gradient | 0.01 | Constant added to the squared gradient in the denominator of the RMSProp. |
| Initial exploration | 1 | Initial value of $\epsilon$ in $\epsilon-$greedy exploration. |
| Final exploration | 0.1 | Final value of $\epsilon$ in $\epsilon-$greedy exploration. |
| Final exploration step | 200000 | Number of update steps over which the initial value of $\epsilon$ is linearly annealed to its final value. |
| Reward | 1 | Any reward gained is scaled to 1. There are no negative rewards. |

Table 1: List of hyperparameters for the DQN algorithm

| **Leaky Integrate and fire neuron** | | |
|---|---|---|
| **Hyperparameter** | **Value** | **Description** |
| Refractory Period | 0 | Number of milliseconds that the neuron is in refractory period after a spike. |
| Threshold voltage | -52 | The membrane potential at and above which a neuron spikes. |
| Resting voltage | -65 | The resting membrane potential of the neuron. |

. . . continued

| **Hyperparameter** | **Value** | **Description** |
|---|---|---|
| Voltage decay | 0.01 | Amount of voltage decay at each time step. |
| **Leaky Integrate and fire neuron with adaptive threshold** | | |
| **Hyperparameter** | **Value** | **Description** |
| Theta plus | 0.05 | Amount of threshold increased after each spike. |
| Theta decay | $1x10^{-7}$ | Time constant of adaptive threshold decay. |

Table 2: List of hyperparameters for the spiking neurons

# Experiments and results

## Binary input

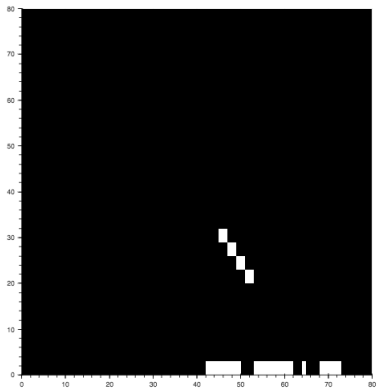


Figure 5: Example binary input

The first part of our experiments use binary pixel inputs. Each state consists of an 80x80 image of binary pixels. The frames from the Gym environment are preprocessed to create the state. The image preprocessing procedure is described in the next section.

### Preprocessing

Each frame from the gym environment is cropped to remove the text above the screen displaying the score and the number of lives left. The image is then re-sized to a 80x80 image and converted to a binary image. The previous frame is then subtracted from the current frame while clamping all the negative value to 0. We then add the most recent 4 difference frames to create a state. Thus, a state is a 80x80 binary image containing the movement information of the last 4 states. Figure 5 shows an example of the input state for the network. The image shows the movement of the ball and the paddle, however, it is not possible to detect the direction of the movement from the image.

### Artificial neural network performance

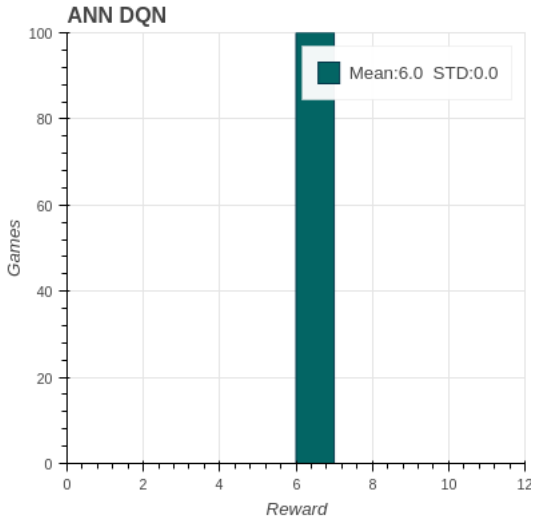



Figure 6: Performance of ANN on binary input

We then trained the ANN using the DQN algorithm with the hyperparameters described in table 1. The trained network was then tested for 100 games using greedy policy and no exploration. Figure 6 shows the reward achieved by the ANN for the 100 games. The ANN consistently achieves a reward of 6. (Note: All rewards achieved during testing are scaled rewards. Therefore, reward is equal to the number of bricks destroyed during each game.)

### Spiking neural network performance

To convert the ANN to SNN, we scaled the weights for each layer to produce spikes. Since the analog values of ANN are too low for the SNN, we scaled the weights of the first layer by 10 times and the weights of the second layer by 100

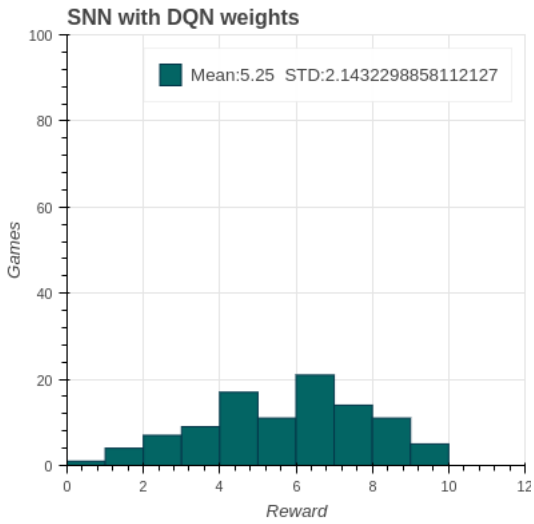



Figure 7: Performance of SNN on binary input. The first layer of weights are scaled by 10x and the second layer of weights are scaled by 100x. The hidden layer uses LIF neurons with adaptive threshold.

times. The Figure 7 shows the performance of a spiking neural network with a hidden layer of neurons with adaptive threshold.

### Time-to-first-spike coding

The subject of rate-based coding versus spike-based coding is an area of active research [2, 24]. For each game with a total reward in between 3-15, the agent needs to process 150-500 frames or states. Each state requires simulating 500 time-steps. While on a neuromorphic hardware this would be fast and energy efficient, simulating the network is time consuming. Therefore, we tested the performance of the network on spike-based coding. Instead of running the network for 500 time-steps, we pick the action that spikes first. In most cases, the neurons in the final layer spike together initially, therefore, we pick the action that wins the tie first. Figure **??** shows the performance of the spike-based coding. The network has similar performance but is 3 times faster than rate-based coding.

### Stochastic LIF neurons

One of the reasons for the decrease in performance after transferring the weights from ANN to SNN could be due to the difference in the non-linear nature of the neurons. The ANN has rectified linear units (ReLU) that behaves linearly for positive value and is zero for the negative values. Spiking neurons, however, communicate by spikes which are binary in nature. Therefore, spiking neurons

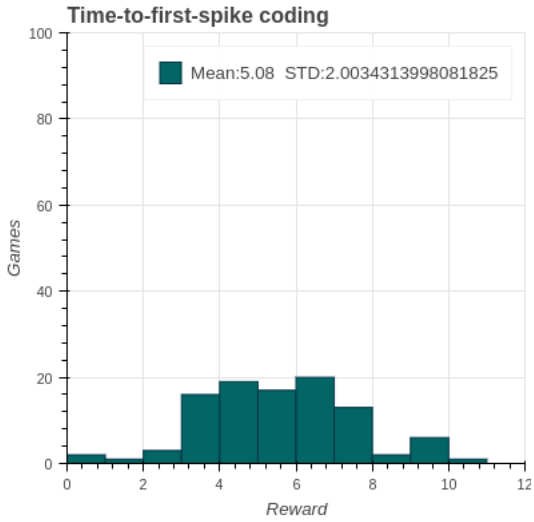



Figure 8: Performance of SNN using the time-to-first-spike coding. The first layer of weights are scaled by 10x and the second layer of weights are scaled by 100x. The hidden layer uses LIF neurons with adaptive threshold.

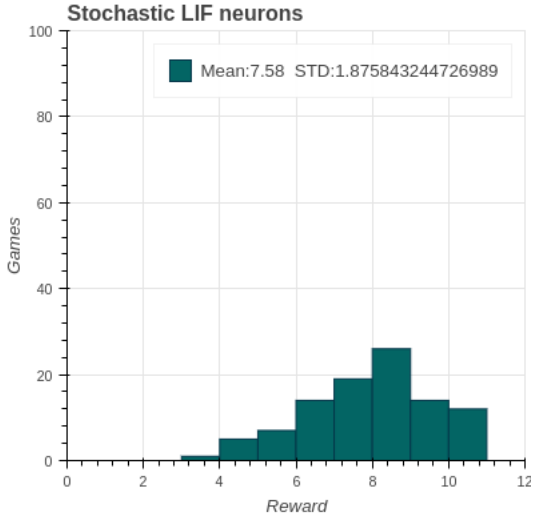



Figure 9: Performance of SNN using stochastic LIF neurons. The first layer of weights are scaled by 10x and the second layer of weights are scaled by 100x. The hidden layer uses LIF neurons with adaptive threshold.

do not give any output of their membrane potential is below threshold. To fix this problem, we use a stochastic LIF neuron [26] that has a chance to spike

at each step with probability directly proportional to its membrane potential. Therefore, the probability to spike for the neuron increases linearly with its membrane potential. The use of stochastic LIF neurons improves the performance of the network significantly and the network performs better on average than the ANN network as shown in figure 9.

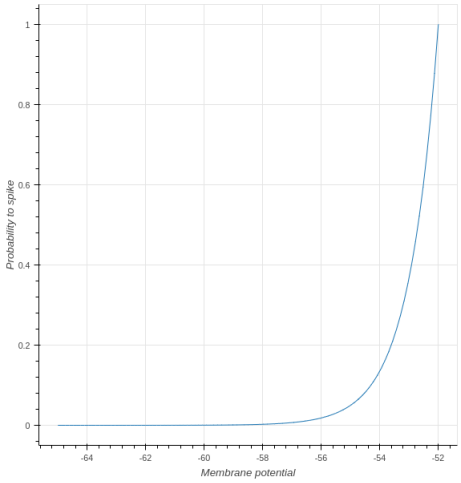



Figure 10: Membrane potential vs. spike probability for stochastic LIF neuron with exponential escape noise function.

The probability to spike below the threshold is also called escape noise in spiking neuron models [26]. Instead of being a linear function, the escape noise $\sigma$ can also be modeled as a bounded exponential function:

$$\sigma(V) = \begin{cases} \frac{\delta t}{\tau_\sigma} \exp\left(\beta_\sigma (V - \theta)\right), & \text{if less than 1} \\ 1, & \text{otherwise} \end{cases} \tag{4}$$

Where $V$ is the membrane potential, $\theta$ is the threshold voltage, $\delta t$ is the duration of the time step and $\tau_\sigma$ and $\beta_\sigma$ are constant positive parameters. For our experiment, we chose $\tau_\sigma$ and $\beta_\sigma$ to be 1, which produces the exponential curve in figure 10. Figure 11 shows the performance of exponential stochastic LIF neurons which performs better than ANN on average, however, not as well as the linear stochastic LIF. Further parameter tuning could provide better results. We leave that to future work.

## Grayscale input

Our initial experiments uses binary difference images since they are easy to convert into spikes, however, the binary input does not contain the information of about the direction of the ball movement which could confuse the agent. To

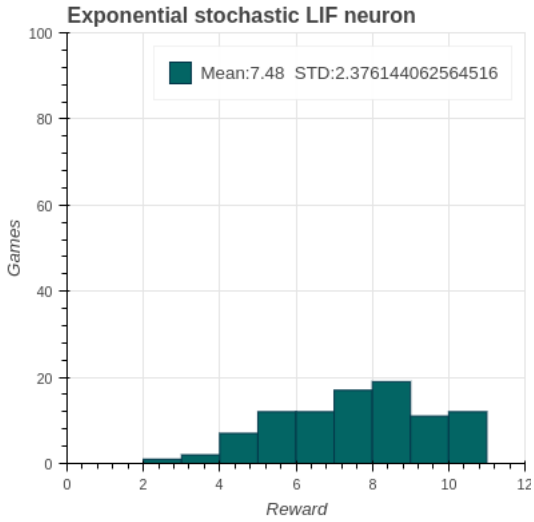



Figure 11: Performance of SNN using stochastic LIF neurons with exponential escape noise function. The first layer of weights are scaled by 10x and the second layer of weights are scaled by 100x. The hidden layer uses LIF neurons with adaptive threshold.

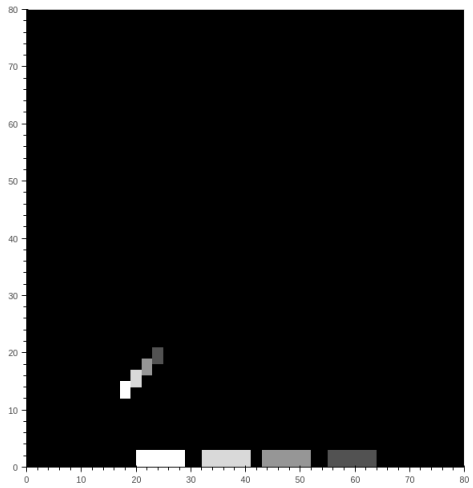


Figure 12: Example grayscale input

solve this problem, we weighted each frame according to time and added them to create the state. The most recent frame has the highest weight and the least recent frame has the least weights. At time $t$ the state is made up of sum of the

most recent 4 frames as follows:

$$S_t = F_t * 1 + F_{t-1} * 0.75 + F_{t-2} * 0.5 + F_{t-3} * 0.25 \tag{5}$$

Where $S_t$ and $F_t$ are the state and the frame at time $t$ respectively. An example of the grayscale input is shown in figure 12.

**Performance**

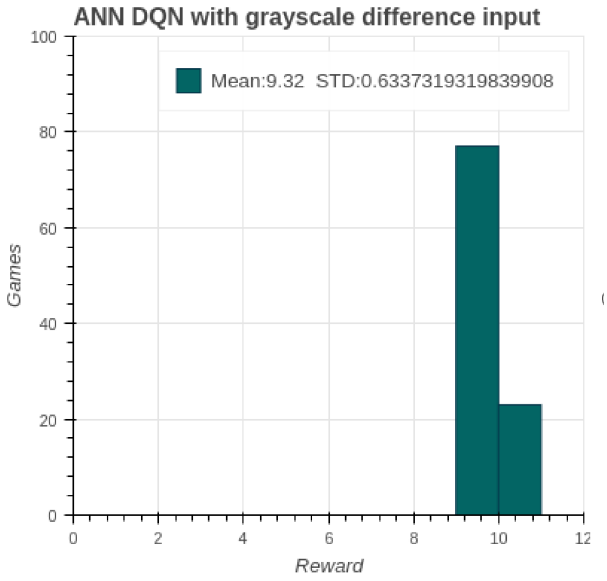



Figure 13: Performance of ANN on grayscale input.

After training, the ANN performance improves to an average reward of 9.32 as shown in figure 13. When converting grayscale input to spikes, we considered two different options:

1. The first method uses the intensity of each pixel as the probability to spike. At each time step, the input neuron spikes with the probability equal to the value of its corresponding pixel.

2. The second method uses the actual grayscale image as the input. Therefore, instead of a binary spike, we value of the spike is equal to the intensity of each pixel. We can also achieve the same result with binary spikes by multiplying the spikes with weights equal to the intensity of the corresponding pixel.

The first method of input does not maintain the relationship between the intensity values of the image and therefore the network does not receive the

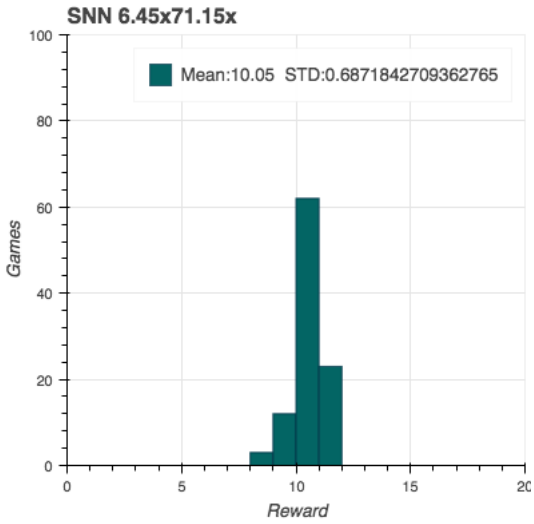



Figure 14: Performance of SNN on grayscale input. The first layer of weights are scaled by 5x and the second layer of weights are the same. The neurons used for this experiment are LIF neurons.

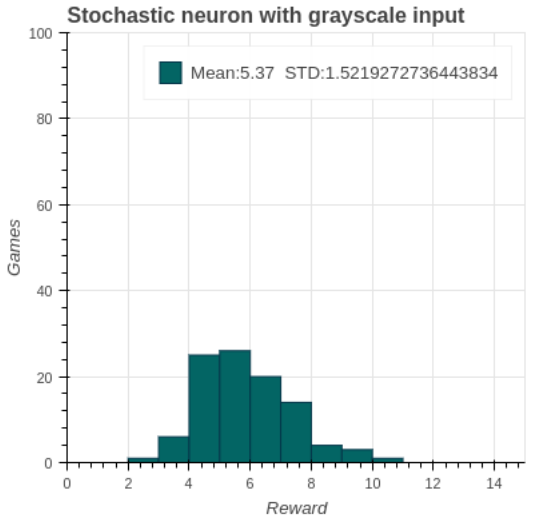


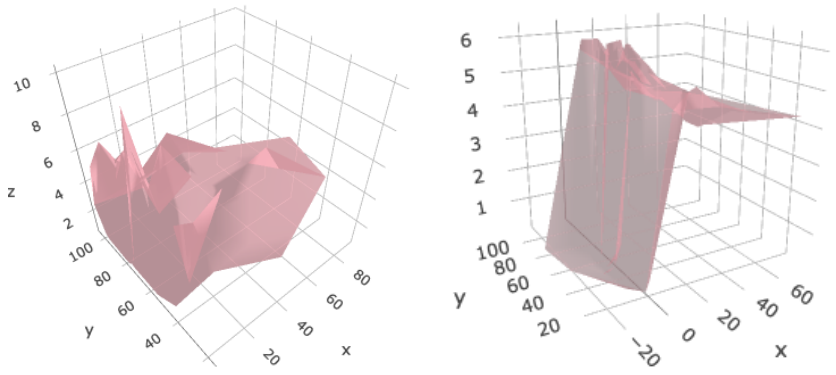

Figure 15: Performance of SNN using stochastic LIF neurons on grayscale input. The first layer of weights are scaled by 180x and the second layer of weights are also scaled by 180x. The neurons in the second layer have adaptive threshold.

weighted time data. Due to this reason, the first method does not perform very well.

The SNN using the second method gets input similar to the ANN and therefore, is able to perform better. Figure 14 shows the performance of the best network for the second method input. Figure 15 shows the performance of the best stochastic network for the second method of input.

**Parameter search**

Finding a spiking neural network that performs well requires manual parameter search. We tested many different parameters such as the type of spiking neuron (stochastic LIF vs. LIF, adaptive threshold vs static threshold), scale of the weights transferred from ANN and the encoding method of the input.
We performed intensive parameter search for the scale of the weights for the two layers of the network for grayscale input network with LIF neurons and stochastic LIF neurons. We used particle swarm optimization(ref) algorithm to perform the parameter search.




(a) SNN (b) stochastic SNN

Figure 16: Surface plot of the particle swarm optimization performed on the weight scales of the two layers of the spiking neural networks plotted against the expected reward. The x-axis represents the scale of the first layer, the y-axis represents the scale of the second layer and the z-axis represents the average reward over 100 episodes.

Figure 16 shows the surface plot of the parameter search performed over these two networks. Note that the $\beta_\sigma$ parameter of the stochastic LIF neurons was not tuned. To achieve better performance on the stochastic LIF network, the $\beta_\sigma$ needs to be tuned.

## Robustness

Recent work have shown that deep Q-networks are vulnerable to white-box and black-box adverserial attacks [10]. The policies learned by the DQN algorithm also generalize poorly for the states of the game that the agent has not trained on. To test the robustness of the SNN against the ANN, we test the performance of each network when the three horizontal bars of pixels of the input are occluded. The performance is tested for every possible position of the bar on the screen. This is a challenging task, however, it tests the robustness and generalization of the policies represented by both the networks.

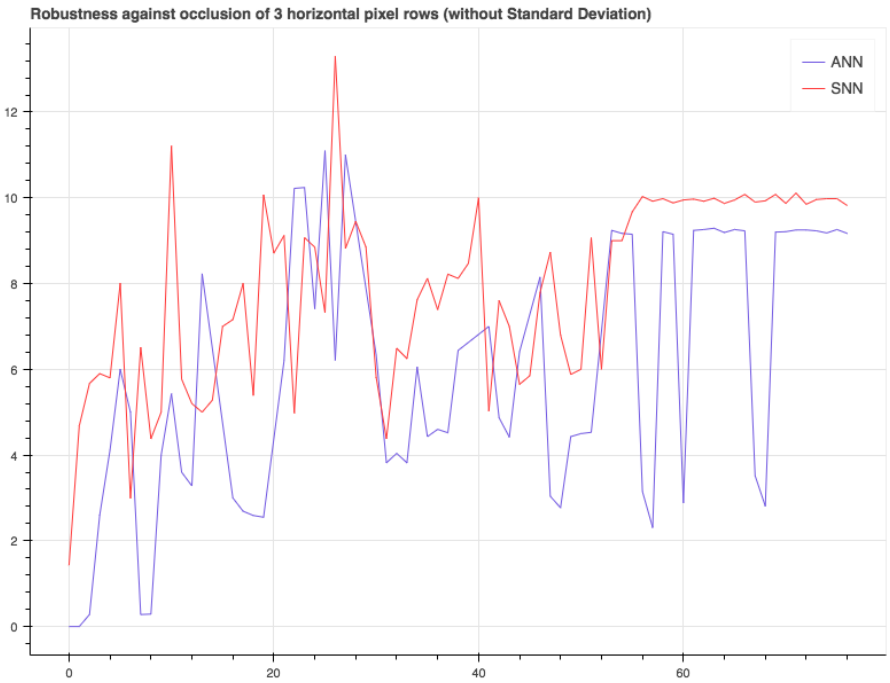



Figure 17: Performance of ANN and SNN for the robustness test. The x-axis represents the position of the bottom most occluded pixel of the horizontal occlusion bar. The y-axis represents the average performance. The plot shows the result of 77 experiments, one for each possible position of the horizontal occlusion bar. Each experiment was run for 100 episodes using greedy policy.

Our experiments on robustness show that SNNs are much more robust compared to ANN even though they share the same weights. The artificial neural network trained using the DQN algorithm is sensitive to changes at a few places in the input. When these areas are occluded, the ANN performs poorly. Surprisingly, occluding these areas does not affect the performance of the SNN.

Figure 17 shows the performance of the ANN and SNN for the robustness task. The lower areas of the screen, represented by lower values on x-axis, are more sensitive and result in poor performance by both networks. The higher areas of the screen contain some sensitive areas that when occluded result in poor performance of the ANN but does not affect the SNN. We also see that the SNN performs better than the ANN in most of the experiments.

# Conclusion

| **Method of input** | **ANN** | **SNN** | **stochastic SNN** |
|---|---|---|---|
| Binary | $6.0 \pm 0$ | $5.25 \pm 2.14$ | $7.58 \pm 1.88$ |
| Grayscale | $9.32 \pm 0.6$ | $10.05 \pm 0.6$ | $5.37 \pm 1.52$ |

Table 3: The best performance achieved by both the methods of input

Table 3 shows the summary of results of the experiments. In this study, we have shown that:

1. Spiking neural networks can be used to represent policies for reinforcement learning tasks like playing Atari games.

2. Spiking neural networks can be trained by transfer of weights from artificial neural networks.

3. Spiking neural networks can outperform the artificial neural network from which the weights have been transferred on reinforcement learning tasks like playing Atari games.

4. Spiking neural networks are more robust to attacks and perturbations in the input. They are also more generalized and perform better on states that they have not encountered before.

Spiking neural networks trained in such a way when coupled with energy efficient neuromorphic hardware have a great advantage over artificial neural networks in terms of energy consumption and performance.

We did not focus on improving the performance of the network by increasing the network size and adding convolutional layers. We believe that it is possible to train larger networks that can outperform human on Atari games. We leave that to future work.

# References


[1] Richard B. Stein. A theoretical analysis of neuronal variability. 5:173–94, 04 1965.

[2] Romain Brette. Philosophy of the spike: Rate-based vs. spike-based theories of the brain. 9, 11 2015.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[4] Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113(1):54–66, May 2015.

[5] J. M. Cruz-Albrecht, M. W. Yung, and N. Srinivasa. Energy-efficient neuron, synapse and stdp integrated circuits. *IEEE Transactions on Biomedical Circuits and Systems*, 6(3):246–256, June 2012.

[6] Danny Hernandez Dario Amodai. Ai and compute. Technical report, OpenAI, 5 2018.

[7] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. Liu, and M. Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2015.

[8] Peter U. Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2015.

[9] H. Hazan, D. J. Saunders, H. Khan, D. T. Sanghavi, H. T. Siegelmann, and R. Kozma. BindsNET: A machine learning-oriented spiking neural networks library in Python. *ArXiv e-prints*, June 2018.

[10] Sandy H. Huang, Nicolas Papernot, Ian J. Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *CoRR*, abs/1702.02284, 2017.

[11] Ryota Kobayashi, Yasuhiro Tsubo, and Shigeru Shinomoto. Made-to-order spiking neuron model equipped with a multi-timescale adaptive threshold. *Frontiers in Computational Neuroscience*, 3:9, 2009.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[13] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659 – 1671, 1997.

[14] Maass, W. Lower Bounds for the Computational Power of Networks of Spiking Neurons. *Neural Computation*, 8(1):1–40, January 1996.

[15] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha. A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45nm. In *2011 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–4, Sept 2011.

[16] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[17] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[18] J. A. Perez-Carrasco, Bo Zhao, C. Serrano, B. Acha, T. Serrano-Gotarredona, Shouchun Chen, and B. Linares-Barranco. Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing–application to feedforward convnets. *IEEE Transactions on Pattern Analysis  Machine Intelligence*, 35(11):2706–2719, Nov. 2013.

[19] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[20] Bodo Rueckauer and Tobi Delbruck. Evaluation of event-based algorithms for optical flow with ground-truth from inertial measurement sensor. *Frontiers in Neuroscience*, 10:176, 2016.

[21] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, and Michael Pfeiffer. Theory and tools for the conversion of analog to spiking convolutional neural networks, 2016.

[22] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[23] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. 550:354–359, 10 2017.

[24] Simon Thorpe and Jacques Gautrais. *Rank Order Coding*, pages 113–118. Springer US, Boston, MA, 1998.

[25] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning, 4, 2012.*

[26] W. M. Kistler W. Gerstner. *Spiking Neuron Models. Single Neurons, Populations, Plasticity.* Cambridge University Press, 2002.