Neural Networks 120 (2019) 108–115

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Improved robustness of reinforcement learning policies upon conversion to spiking neuronal network platforms applied to Atari Breakout game

Devdhar Patel [a,*], Hananel Hazan [a], Daniel J. Saunders [a], Hava T. Siegelmann [a], Robert Kozma [a,b,*]

[a] Biologically Inspired Neural and Dynamical Systems Laboratory (BINDS) College of Computer and Information Sciences, 140 Governors Drive, University of Massachusetts Amherst, Amherst, MA 01003, USA
[b] Center for Large-Scale Integrated Optimization and Networks (CLION) Department of Mathematical Sciences, 373 Dunn Hall, University of Memphis, Memphis, TN 38152, USA

ARTICLE INFO

Article history:
Available online 25 August 2019

Keywords:
Spiking neural networks
Reinforcement learning
Deep learning
Robustness
Atari

ABSTRACT

Deep Reinforcement Learning (RL) demonstrates excellent performance on tasks that can be solved by trained policy. It plays a dominant role among cutting-edge machine learning approaches using multi-layer Neural networks (NNs). At the same time, Deep RL suffers from high sensitivity to noisy, incomplete, and misleading input data. Following biological intuition, we involve Spiking Neural Networks (SNNs) to address some deficiencies of deep RL solutions. Previous studies in image classification domain demonstrated that standard NNs (with ReLU nonlinearity) trained using supervised learning can be converted to SNNs with negligible deterioration in performance. In this paper, we extend those conversion results to the domain of Q-Learning NNs trained using RL. We provide a proof of principle of the conversion of standard NN to SNN. In addition, we show that the SNN has improved robustness to occlusion in the input image. Finally, we introduce results with converting full-scale Deep Q-network to SNN, paving the way for future research to robust Deep RL applications.

1. Introduction

Among the giants of neural networks and brain science, Stephen Grossberg has a truly unique and overarching legacy in the research fields encompassing the physiology and mathematical modeling of neural processing and brain functions. From the enormous amount of his influential research achievements produced for over 60 years, here we emphasize his results on laminar cortical models of spiking neurons and laminar computing. The Synchronous Matching Adaptive Resonance Theory (SMART) model is a groundbreaking study employing synchronous oscillations in spiking neurons to describe attentive learning in thalamocortical circuits (Versace & Grossberg, 2008). The laminar computing principle later has been extended to modeling visual cortical processing and the formation of visual percepts (Cao & Grossberg, 2012; Leveille, Versace, & Grossberg, 2010). Grossberg's laminar computing approach has important impact on not only computational models, but also on hardware developments, and it provides a blueprint for advanced chip designs to implement various machine learning tasks, such as IBM's TrueNorth (Pedroni et al., 2016), Intel's Loihi (Davies et al., 2018), and SpiNNaker at Manchester, UK (Furber, Galluppi, Temple, & Plana, 2014). These chips employ spiking neural networks, which provide the energy efficiency required for the future dynamical development of *sustainable AI* and brain-inspired technologies.

Recent advancements in AI and Machine Learning (ML) have astonishing results surpassing human performance in various testbeds, including ATARI games (Hasselt, Guez, & Silver, 2016; Mnih et al., 2015; Wang et al., 2016). These successes have lead to enormous interest in AI and neural networks, including Deep Reinforcement Learning (RL). However, rigorous analysis showed that deep RL is susceptible to random and malicious perturbations in the inputs, related to adversarial AI (Huang, Papernot, Goodfellow, Duan, & Abbeel, 2017). A consequence of the applied gradient descent algorithm is that the trained agent learns to focus on a few sensitive areas. However, the performance of the RL agent deteriorates when these areas are occluded or perturbed. Moreover, there is evidence that the policies learned by the networks in deep RL algorithms do not generalize well; the performance of the agent deteriorates when it encounters

* Corresponding authors.
E-mail addresses: devdharpatel@cs.umass.edu (D. Patel), rkozma@cs.umass.edu (R. Kozma).

https://doi.org/10.1016/j.neunet.2019.08.009
0893-6080/

a state that it has not seen before, even if it is similar to other experienced states (Witty et al., 2018).

Biological systems tend to be very noisy by nature (Richardson & Gerstner, 2006; Stein, Gossen, & Jones, 2005), still they operate well even under harsh conditions that affect their input and internal state. Spiking Neural Networks (SNNs) are considered to be closer to biological neurons due to their event-based nature; they are often termed the third generation of neural networks (Maass, 1996). A spike is the quantification of the internal and external processes involving the neuron. The individual neurons operating with spikes serve as microscopic bottlenecks, which have the ability to sustain low intermittent noise and do not transmit sub-threshold noise to their neighbors. Moreover, populations of spiking neurons in a network can mitigate the impact of noise even further due to their collective effect and their architectural connectivity (Hazan & Manevitz, 2012). Following biological intuition, we involve SNNs to enhance the benefits of deep RL solutions.

An important potential advantage of SNNs is their energy efficiency. Due to the binary, event-based nature, SNNs can support energy utilization that is more efficient than the one provided by traditional neural networks, especially when implemented on neuromorphic hardware (Martí, Rigotti, Seok, & Fusi, 2016). Among the various hardware solutions, memristor technology demonstrated great promise for neuromorphic computing (Bichler, Roclin, Gamrat, & Querlioz, 2013; Kozma, Pino, & Pazienza, 2012; Srinivasa & Cruz-Albert, 2012). In recent years, we witnessed the proliferation of neuromorphic hardware platforms, such as IBM TrueNorth and Intel Loihi (Benjamin et al., 2014; Davies et al., 2018; Pedroni et al., 2016).

It is difficult to train spiking neurons using backpropagation due to the non-differentiable nature of the spike dynamics (Pfeiffer & Pfeil, 2018). One important direction of recent work with SNNs has focused on adjusting backpropagation to the event-based nature of spiking neuron activation (Huh & Sejnowski, 2018; Wu, Deng, Li, Zhu, & Shi, 2018; Zenke & Ganguli, 2018). An alternative approach aimed at biologically inspired local learning rules, e.g., spike-timing-dependent plasticity (STDP) to train the network (Bengio, Fischer, Mesnard, Zhang, & Wu, 2015; Davies et al., 2018; Diehl & Cook, 2015; Ferre, Mamalet, & Thorpe, 2018; Gilra & Gerstner, 2018; Hazan et al., 2018).

In this work, we explore a different approach, in which no learning takes place in the SNN at all, rather the weights obtained by training a ReLU NN are converted to the SNN having the same structure. This idea may sound either trivial or crazy, still there is ample evidence that it indeed works. The idea of converting Convolutional NNs (CNNs) to SNNs with the aim of processing inputs from event-based sensors was first introduced in Perez-Carrasco et al. (2013) Cao, Chen, and Khosla (2015) observed that the activations of ReLU neurons can be mapped to the frequency of spikes produced by the spiking neurons and reported good performance on computer vision benchmarks.

Diehl et al. (2015) proposed a method of weight normalization that re-scales the weights of the SNN to reduce the errors due to excessive spiking or due to sparse spiking of the neurons. They also demonstrated a near lossless conversion of ReLU NNs to SNNs for the MNIST classification task. Rueckauer and colleagues identified spiking equivalents of a variety of common operations used in deep convolutional networks like max-pooling, softmax, batch-normalization, and inception modules (Rueckauer, Lungu, Hu, & Pfeiffer, 2016; Rueckauer, Lungu, Hu, Pfeiffer, & Liu, 2017). This allowed them to convert popular CNN architectures like VGG-16 (Simonyan & Zisserman, 2014), Inception-V3 (Szegedy, Vanhoucke, Ioffe, Shlens, & Wojna, 2016) and BinaryNet (Courbariaux, Hubara, Soudry, El-Yaniv, & Bengio, 2016) to SNN. They achieved near lossless conversion in these networks. All these efforts aimed at classification tasks, while according to our knowledge, there has not been previous work on the conversion of Deep Q-networks to SNNs.

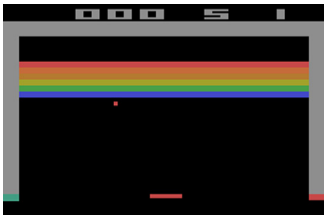


Fig. 1. Screenshot of Atari 2600 Breakout game. The ball bounces between the wall (lines of colored bricks) and the paddle (red bar at the bottom). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The combination of RL and spiking neurons is a natural choice, since animals learn to perform certain tasks using semi-supervised and reinforcement learning. Moreover, there is evidence that biological neurons learn using evaluative feedback from neurotransmitters such as dopamine (Wang et al., 2018), e.g., in the postulated dopamine reward prediction-error signal (Schultz, 2016). However, since spiking neurons are fundamentally different from ReLU artificial neurons, it is not clear if SNNs can address machine learning in RL domain. This raises the questions: Do SNNs have the capability to represent the same functions as ReLU NNs? To be more specific, can SNNs represent complex policies that can successfully play Atari games?

We answer these questions by demonstrating that ReLU NNs trained using RL algorithms can be converted to SNN without deterioration of the performance on the RL task when playing Atari Breakout game. Furthermore, we show that the converted SNN is more robust to input perturbations than the original NN. Finally, we demonstrate that full-sized Deep Q-Network (DQN) (Mnih et al., 2015) can be converted to SNN and maintain its better than human performance, paving the way for future research in robustness and RL with SNNs. Results presented in this paper have been produced using the open source BindsNET spiking neural networks library, available on Github https://github.com/Hananel-Hazan/bindsnet.

2. Background

2.1. Arcade learning environment

The Arcade learning environment (ALE) (Bellemare, Naddaf, Veness, & Bowling, 2013) is a platform that enables researchers to test their algorithms on over 50 Atari 2600 games. The agent sees the environment through image frames of the game, interacts with the environment with 18 possible actions, and receives feedback in the form of the change in the game score. The games were designed for humans and thus are free from experimenter bias. The games span many different genres that require the agent/algorithm to generalize over various tasks, difficulty levels, and timescales. ALE thus has become a popular test-bed for reinforcement learning (Mnih et al., 2015).

Breakout: We demonstrate our results on the game of Breakout; Fig. 1 shows a frame of the game. Breakout is similar to the popular game Pong. The player controls a paddle at the bottom of the screen; see red bar at the bottom of Fig. 1. There are rows of colored bricks on the upper part of the screen. A ball bounces in between the bricks and the player controlled paddle. There are 4 possible actions: move left, move right, do not move, and fire. If the ball hits a brick, the brick breaks and the score of the game is increased. However, if the ball falls below the paddle, the player loses a life. The game starts with five lives, and the player/agent is supposed to break all the bricks before they run out of lives.

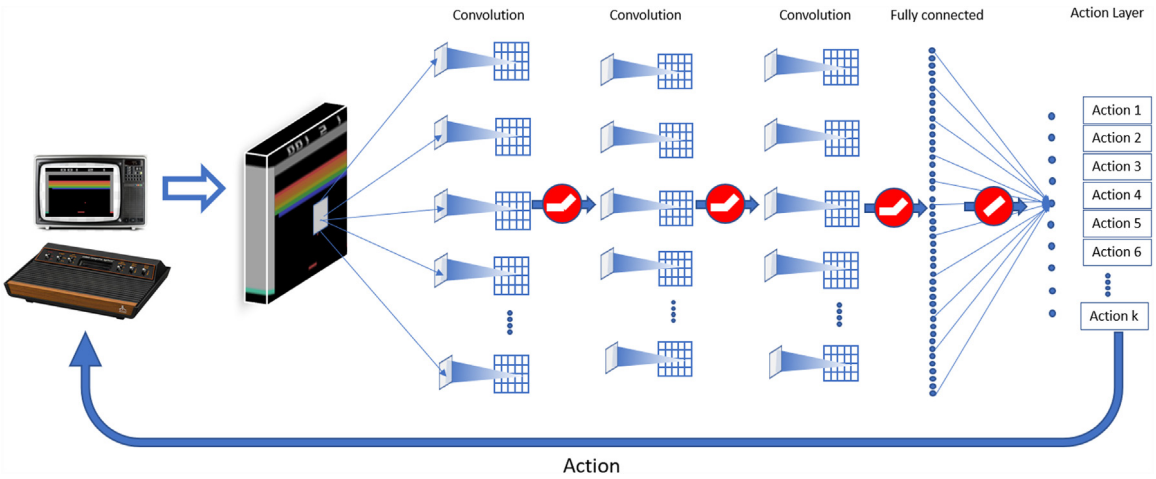


Fig. 2. Architecture of Deep Q-networks; following Mnih et al. (2015); ReLU nonlinear units are emphasized by red circles.

2.2. Deep Q-networks

Reinforcement learning algorithms train a policy π to maximize the expected cumulative reward received over time. Formally, this process is modeled as a Markov decision process (MDP). Given a state-space $\mathcal{S}$ and an action-space $\mathcal{A}$, the agent starts in an initial state $s_0 \in \mathcal{S}_0$ from a set of possible start states $S_0 \in \mathcal{S}$. At each time-step t, starting from $t = 0$, the agent takes an action a_t to transition from s_t to s_{t+1}. The probability of transitioning from state s to state s' by taking action a is given by the transition function $P(s, a, s')$. The reward function $R(s, a)$ defines the expected reward received by the agent after taking action a on state s.

A policy π is defined as the conditional distribution of actions given the state $\pi(s, a) = Pr(A_t = a|S_t = s)$. The Q-value or action-value of a state–action pair for a given policy, $q^\pi(s, a)$, is the expected return following policy π after taking the action a from state s.

$$q^\pi(s, a) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k} | S_t = s, A_t = a, \pi] \tag{1}$$

where γ is the discount factor. The action-value function follows a Bellman equation that can be written as:

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

Many widely used reinforcement learning algorithms first approximate the Q-value and then select the policy that maximizes the Q-value at each step to maximize returns (Sutton & Barto, 2018). Deep Q-networks (DQN) (Mnih et al., 2015) are one such algorithm that uses deep artificial neural networks to approximate the Q-value. The neural network can learn policies from the pixels of the screen and the game score. It has been shown that DQN surpass human performance on many of the Atari games.

2.3. Spiking neurons

SNNs may use any of the various neuron models (Gerstner, 2002; Tuckwell, 1988). Here, we introduce four different types of spiking neurons. We use the following notations to describe the dynamics of the neurons: $v(t)$ is the time-dependent membrane potential voltage; v_{rest} is the resting membrane potential; v_{thresh} is the firing threshold of the neuron; τ is the time constant of the neuron dynamics.

1. *Integrate-and-fire (IF) neuron:* The IF neuron is the simplest form of spiking neuron models. The neuron simply integrates input until the membrane potential $v(t)$ exceeds the voltage threshold $v_{threshold}$ and a spike is generated. Once the spike is generated, the membrane potential is reset to v_{reset}.

$$\tau \frac{dv(t)}{dt} = \sum_{i=1}^{n} W_i * Input_i. \tag{3}$$

2. *Subtractive Integrate-and-fire (SubIF) neuron:* The SubLIF neuron behaves similar to the IF neuron with one small change, when the membrane potential voltage exceeds threshold value, the neuron emits a spike and resets its membrane voltage to $v_{reset} + (v(t) - v_{threshold})$ (Cassidy et al., 2013; Diehl et al., 2016; Rueckauer et al., 2017). By adding the overshoot voltage the neuron "remembers" the excessive voltage from the last spike and will be more prone to be excited with the next incoming inputs. This reduces the information lost when spiking in SNN is converted from ReLU NN.
3. *Leaky integrate-and-fire (LIF) neuron:* The LIF neuron behaves similarly to the IF neuron. However, for every time-step that its membrane potential is above the resting potential, the neuron leaks a constant amount of current:

$$\tau \frac{dv(t)}{dt} = -(v_t - v_{rest}) + \sum_{i=1}^{n} W_i * Input_i. \tag{4}$$

4. *Stochastic leaky integrate-and-fire neuron:* The stochastic LIF neuron is based on the LIF neuron. However, the neuron may spike if its membrane potential is below the threshold with probability proportional to its membrane potential (escape noise). The escape noise (σ) is described here:

$$\sigma = \begin{cases} 1/\tau_\sigma \exp(\beta_\sigma(v_t - v_{threshold})) & \text{if less than 1} \\ 1 & \text{otherwise,} \end{cases} \tag{5}$$

where τ_σ and β_σ are positive constant parameters. For the spiking models listed above, the neuron enters a refractory period after a spike, during which they are unable to spike or integrate input. In this paper, for simplicity, we ignore the refractory period in the conversion from artificial neurons, and we set both τ_σ and β_σ to 1. For a complete list of the parameters used for the neurons, see *Supplementary Materials*, Table 2.

Note that unlike traditional artificial NNs, SNNs need to be simulated for a period of time to produce spike trains and interpret the resulting activity. The simulation is done in discrete

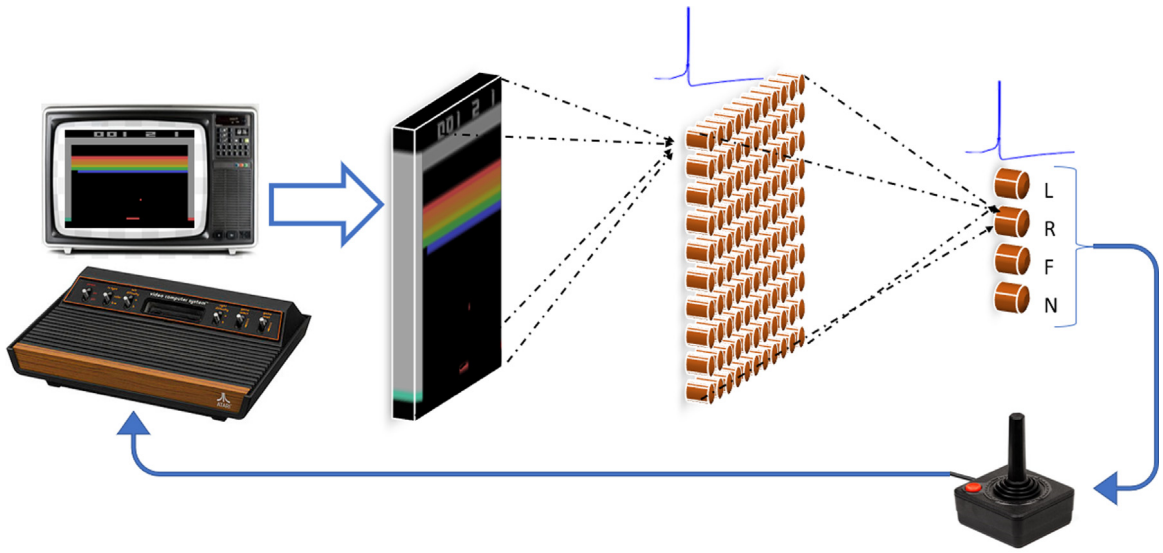


Fig. 3. Network architecture: The input to the network consists of an 80 × 80 image produced by preprocessing the frames of the game. The hidden layer consists of 1000 neurons, the output layer has 4 nodes corresponding to the number of possible actions.

time steps. To avoid confusion between the time step of the RL environment and the time step of the SNN, we denote the time step of the RL environment by t and the time step of the SNN by nt.

3. Methods

3.1. Inputs

3.1.1. Binary input

First we consider binary pixel inputs, following Mnih et al. (2015). Each state consists of an 80 × 80 image of binary pixels. The frames from the AI Gym environment are pre-processed to create the state for further analysis. Each frame from the AI 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 an 80 × 80 image and converted to a binary image. The previous frame is then subtracted from the current frame while clamping all the negative values to 0. We then add the most recent four such difference frames to create a state for the RL environment. Thus, a state is an 80 × 80 binary image containing the movement information of the last four states.

3.1.2. Grayscale input

The binary input described above does not contain information about the direction of the ball movement, which we believe can confuse the agent. To alleviate 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 the sum of the most recent 4 frames as follows:

$$S_t = F_t * 1 + F_{t-1} * 0.75 + F_{t-2} * 0.5 + F_{t-3} * 0.25. \quad (6)$$

Here S_t and F_t are the state and the frame at time t, respectively.

3.2. Network architecture

The networks used in the DQN algorithm for Atari games consist of multiple convolutional layers followed by fully connected layers (Mnih et al., 2015). In this work, we started by testing our methods on a shallow ReLU NN with one hidden layer, in order to demonstrate the feasibility of using weight transfer in RL problems. Then we moved on to full-sized Deep Q-network with the same architecture as Mnih et al. (2015).

Fig. 3 shows the network architecture of the shallow SNN. The network architecture of the SNN is the same as the ReLU NN, except that the ReLU nonlinearities of the neurons are replaced by spiking neurons. The network consists of 80 × 80 input layer, followed by a fully connected hidden layer with 1000 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 in the Breakout game.

Table 1
Best performance achieved for different inputs and networks. Each value represents an average of 100 episodes.

Input	ReLU NN	SNN with LIF	Stochastic SNN w/ LIF
0.05 epsilon greedy			
Binary	5.77 ± 3.07	6.21 ± 1.74	7.12 ± 2.47
Grayscale	6.55 ± 1.53	7.28 ± 1.79	7.5 ± 2.16
Greedy			
Binary	6.0 ± 0	5.25 ± 2.13	7.58 ± 1.87
Grayscale	9.32 ± 0.63	10.05 ± 0.68	8.0 ± 2.37

3.3. Training by reinforcement Q-learning approach

We trained the networks using the DQN algorithm (Mnih et al., 2015). We trained the smaller networks using a replay memory size of 200,000 and initial replay memory size of 50,000. We trained the network over 30,000 episodes. Each episode refers to one game of breakout with five lives. The episode ends when the agent/player runs out of lives. The rest of the hyper-parameters we used are same as in Mnih et al.'s (2015) work. For a complete list of the hyper-parameters, see *Supplementary Materials*, Table 1.

3.4. Conversion of trained ReLU NN to SNN

The ReLU NN, which has been trained using the DQN algorithm, is converted to SNN. For the converted SNN, the firing frequency of the spiking neurons in the output layer is proportional to the Q-value of the corresponding action. ReLU NN can be converted to SNN by replacing the ReLU neurons with spiking neurons. However, the result of this straightforward conversion

may produce a very sparse spiking activity in the network. This is due to the fact that the spiking neurons have a constant positive threshold while ReLU neurons activate at any value above zero. To address this sparsity issue, the SNN is simulated for a large number of time steps(nt) for a given input to generate sufficient level of spiking activity, allowing robust estimation of the Q values. In our experiments, we simulate the SNN for nt = 500 time-steps, and we repeat this simulation for each input pattern. In order to expedite the process, we also increase the spiking activity by scaling up the weights. Generally, the weights of deeper layers need to be scaled more than weights of the layers close to the inputs because the network activity becomes sparser in deeper layers.

Due to the fundamental difference between a spiking neuron and a ReLU neuron, the frequency of the spiking neurons cannot accurately represent the output of equivalent ReLU neurons. This is due to the fact that the spiking neurons can only output discrete spikes, while the ReLU neurons have continuous outputs. The conversion of continuous activity (ReLU) to discrete/spiking activity lies in the very heart of this method. The output frequency of the spiking neuron is limited by the choice of membrane potential threshold and simulation time-steps (nt). However, we can reduce the error of spiking neurons by scaling the weights of each layer of network and improve the performance of the SNN. We treat the scaling of the weights at each layer as independent parameters to be optimized to achieve high the performance of the network in the Atari game testbed. All the weights of the same layer are scaled by the same factor thus preserving the learned filters. The optimal weight scaling parameters can be searched by various methods. Rueckauer et al. (2017) showed a useful approach of scaling by normalizing the weights. Their approach was based on scaling the weights in a way that the output error of the majority of the spiking neurons is minimized.

There are various alternatives to Rueckauer's method (Rueckauer et al., 2017) to optimize the transfer from ReLU NN to spiking NNs. For example, Sharmin et al. (2019) provide impressive results in the context of adversarial AI. In our present study, we explored several ways to search for optimal scaling parameters, including particle swarm optimization (PSO) (Clerc, 2012), and simple exhaustive grid search. Among the studied optimization methods, PSO has produced the best performance, and we briefly summarize it here. PSO uses particles to evaluate the fitness of various positions inside the search space. The particles inform each other on their previous best positions. Each particle has a velocity attached to it, and the velocity and the position of the particles are updated at each iteration using a set of rules, which allow efficient exploration of the search space. In our approach, the nth dimension of the particle position determines the value, by which the nth layer of the SNN is scaled. The fitness of a particular position is determined by evaluating how well a specific network performs on the game, based on the scaling the coordinates in that position.

PSO-based optimization of the network demonstrates much improved performance on the Atari game w.r.t. Rueckauer et al.'s (2017) method, which reduces the error between the output of spiking neurons and ReLU neurons. In short, PSO acts as a training algorithm for the SNN. PSO is better suited for the networks trained using the DQN algorithm because unlike image classification tasks, which use the cross-entropy loss, the output values of Q-networks do not differ by large values thus making it harder to differentiate when they are discretized in the SNN.

4. Results

4.1. Performance in breakout games using shallow NNs

Testing SNN based agents in the ALE is a computationally demanding task. We simulate spiking neurons using the PyTorch based open source library BindsNET (Hazan et al., 2018). BindsNET has the advantage compared to some alternative spiking NN packages of allowing users to leverage GPUs to simulate the SNN and speed up testing.

We used PSO algorithm to determine the scale of each of the two layers; thus the dimension D of the search space is 2. The swarm size S is set using the formula:

$$S = 10 + [2\sqrt{D}],$$

where $[u]$ is the integral part of the real number u. For our experiment, the swarm size is $S = 13$. The fitness of each particle was given by the average reward over 100 episodes. The stochastic LIF network has a smoother surface of performance over the parameter space than the LIF network. This suggests that the stochastic LIF network is more robust to change in the scaling of its weights. The escape noise of the stochastic LIF neuron can be tuned to improve the performance further however we leave that to future work.

Results of the experiments with shallow NNs are summarized in Table 1. The displayed performance values are obtained by running 100 episodes using two different input encoding (Binary and Grayscale), and applying two policies (greedy and 0.05 epsilon-greedy). We tested the ReLU NN, SNN using LIF neurons, and SNNs using stochastic LIF neurons.

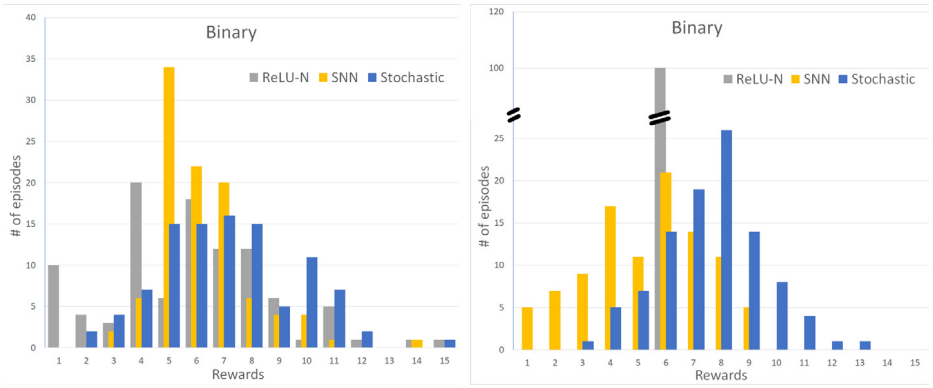
Table 1 summarizes the performance of the ReLU NN against the performance of SNN and the stochastic SNN for binary and grayscale inputs. Data in Table 1 with *Binary input* demonstrate that SNNs are capable of representing policies developed through RL, and they can outperform the ReLU NNs they originate from. We can see that the stochastic SNN performs better on average than the ReLU NN it has been converted from. The optimal parameters for the binary input spiking neural networks were found using grid search.

Table 1 shows that the performance of networks with *Grayscale input* is higher then the *Binary input* for both networks (SNN and ReLU NN). Note that the shown reward values significantly exceed the values by random choice, which is 1.27 ± 1.45 in these experiments. Note that the ANN has no probabilistic components, and by starting the games with the same initialization over 100 episodes, it reproduces the same outcome every time (zero standard deviation). We also see that the standard deviation of the rewards gained by the SNN is lower and the behavior is less random than for the binary input.

Fig. 4 provides further details on the performance of the various classification methods, using the histograms of the reward distributions. The distributions in figure were determined using 0.05 epsilon greedy policy with binary and grayscale inputs.

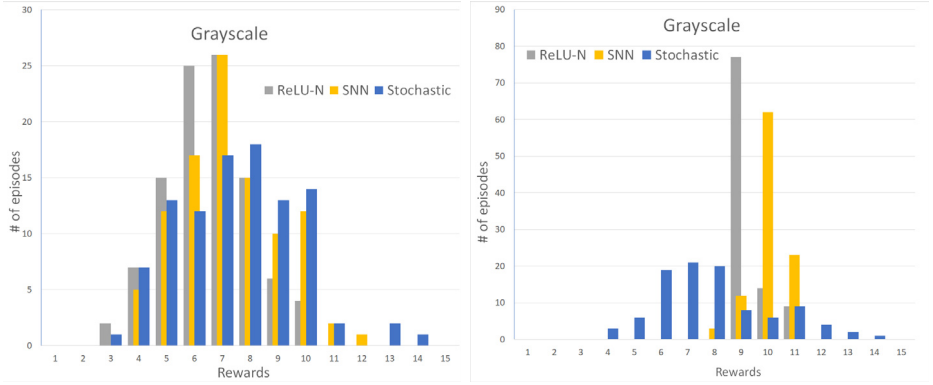
4.2. Robustness of the SNN performance

Deep Q-networks are vulnerable to white-box and black-box adversarial attacks (Huang et al., 2017). Witty et al. (2018) showed that the policies learned by the DQN algorithm generalize poorly to the states that the agent has not seen during training. To evaluate the robustness of the SNN, we test the performance of the shallow Relu-N and SNN networks with grayscale input when a 3-pixel thick horizontal bar spanning the entire width of the input is occluded. The thickness of the occlusion bar corresponds to the thickness of the paddle on the screen after preprocessing. We tested the performance for every position of the bar, by



(a) 0.05 epsilon greedy binary input

(b) Greedy binary input



(c) 0.05 epsilon greedy grayscale input

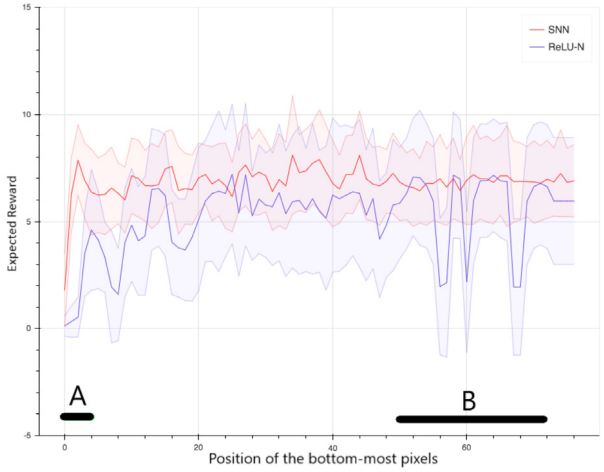
(d) Greedy grayscale input

Fig. 4. Performance of the networks for Binary and Grayscale inputs; each plot shows the reward distribution over 100 episodes using 0.05 epsilon greedy policy.

moving it from the lowest position at the bottom of the screen, step-by-step until it reaches the top. The position of the occlusion bar does not change during each episode. This is a challenging task, since the bar may completely or partially occlude the ball or the paddle.

Fig. 5 shows the performance of the ReLU NN and SNN for the robustness task. The *x*-axis represents the vertical position of the lowest occluded pixels. As we move from left to right on the plot, the occlusion bar moves from bottom to the top of the screen; this represents in total 77 experiments for the 77 positions of the occlusion bar. Each experiment was run for 100 episodes using 0.05 epsilon greedy policy.

The SNN is more robust to occlusions than ReLU NN, as it is seen in Fig. 5, as the reward of the SNN (red) is typically higher that the reward of ReLU NN (blue). Moreover, the ReLU NN is very sensitive to occlusions and perturbations at a few places in the input; namely at the bottom near the paddle (shows as region *A*), and at the medium positions where the brick wall is located (region *B*). When these areas are occluded, the ReLU-N performs poorly. Occlusions in these areas result in drastic decrease in the performance of the ReLU NN. Occluding some of the positions in area *B*, results in a sharp drop in performance for ReLU NN. This can be explained by the nature of the gradient descent updates. Since the score changes when the ball hits the bricks and the backpropagation loss calculated using the TD-error is highest when the score changes, the filters of the network learn to discriminate these areas. Thus, when these areas are occluded, the performance drops. Occlusions near areas *A* and *B* have much less negative impact on the performance of the SNN. Once the paddle is visible, we see that the SNN has no significant loss in performance. Over the other sensitive occlusion area *B* near the wall, where ReLU NN has significant drop in performance, SNN performance is sustained without deterioration. For detailed list of results for positions of the occlusion see *Supplementary Materials*, Table 3. We are intensively working on the interpretation of this robustness result and its generalization to a range of task domains.



(a) Pixel-wise robustness ReLU-N vs SNN

Fig. 5. Performance of ReLU-N and SNN for the robustness test. The x-axis represents the position of the bottom most occluded pixels of the 3-pixel thick horizontal occlusion bar. The y-axis represents the average reward. The standard distribution for the reward distribution is shown using the shaded region. The two critical areas are marked by the black bars **A** and **B** at the bottom of the plot. **A** shows the area near the paddle, while **B** marks the region of the screen occupied by the brick wall.

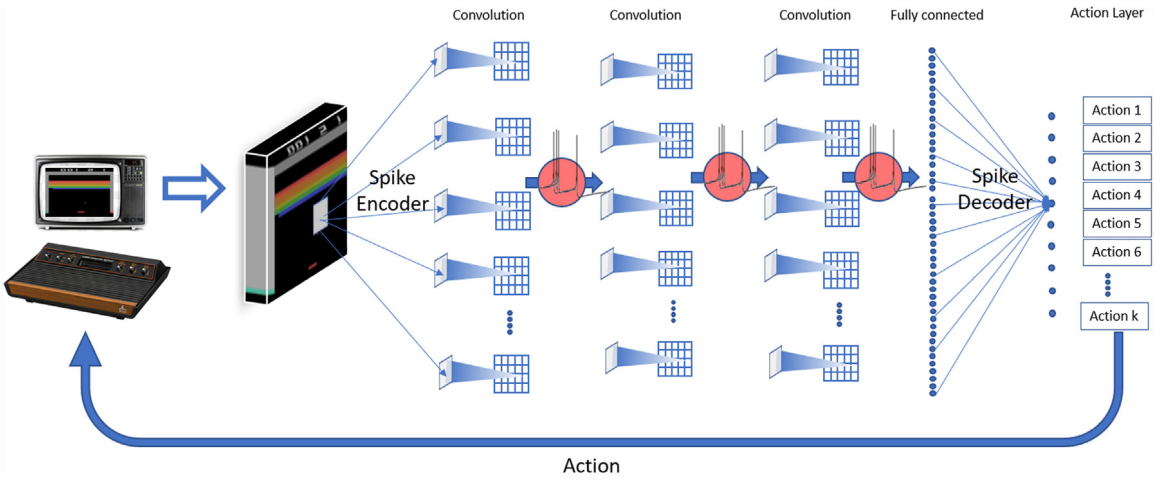
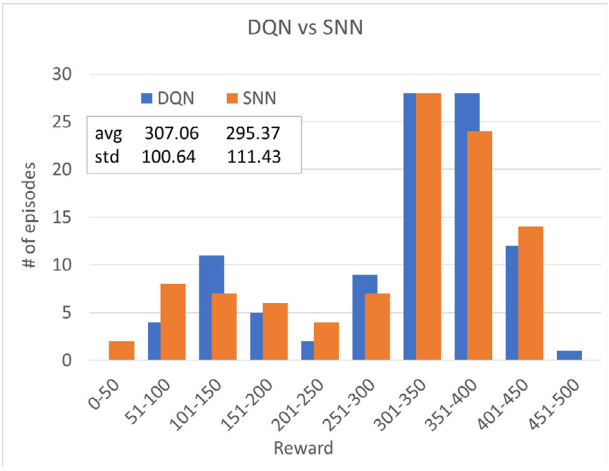


Fig. 6. Network architecture following Mnih et al. (2015), after converting ReLU nonlinearity to spiking network.



(a) DQN vs SNN

Fig. 7. Performance of Deep Q-network vs. Deep Spiking Network. Each plot shows the reward distribution over 100 episodes using 0.05 epsilon greedy policy.

5. Performance of SNN obtained by weight transfer from Deep Q-network

To test our approach for state-of-the-art, large-scale networks, we trained the Deep Q-Network (Mnih et al., 2015) and converted the weights to SNN. We used the OpenAI baseline implementation of DQN to train the network (Dhariwal et al., 2017). Fig. 6 shows the SNN converted from the Deep Q-Network in Fig. 2. Since converting the DQN to SNN requires a search for a large number of parameters, we used the established parameter normalization method (Rueckauer et al., 2017). This approach shows reasonable performance, although it can be clearly improved using a systematic parameter optimization method, like PSO. In the Deep Q-SNN, we used the subtractive-IF neuron.

Fig. 7 displays the distribution of the rewards in the DQN and Deep SNN. These results show that the Deep Q-Network can be converted to Spiking Q-network without significant loss in performance. At the present stage of our studies, we did not conduct robustness test for the full-scale trained networks. We leave a systematic robustness study and comparison as the objective of future research.

6. Conclusions

In this paper, we demonstrate that shallow and deep ReLU NNs trained on the game breakout can be converted to SNNs without degradation of performance. Moreover, SNN seems to display more robustness to occlusion attack. We hypothesize that robustness may be due to the binary nature of spiking neurons, ignoring small perturbations in the data unlike high-precision traditional neural networks. Moreover, the properly optimized conversion method from ReLU to spiking nonlinearity also contributes to the robustness of the results. Moreover, in some cases, SNNs perform better than ReLU NN on previously unseen states.

These results, combined with additional benefits of SNNs, such as energy efficiency on neuromorphic hardware, show that SNNs may be useful to supplement the power of reinforcement learning in DQN tasks, when resources are limited and the input data are noisy and potentially misleading.

Acknowledgments

The work of D.P., H.H., D.J.S., and R.K. has been supported in part by Defense Advanced Research Project Agency, USA Grant, DARPA/MTO HR0011-16-l-0006. Partial support has been provided to R.K. by National Science Foundation, USA Grant NSF-CRCNS-DMS-13-11165. H.T.S. contribution to this work took place prior to assuming her position at DARPA. The information contained in this work does not necessarily reflect the position or the policy of the Government.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.neunet.2019.08.009.

References

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: an evaluation platform for general agents. *Journal of Artificial Intelligence Research*, *47*, 253–279.

Bengio, Y., Fischer, A., Mesnard, T., Zhang, S., & Wu, Y. (2015). From stdp towards biologically plausible deep learning. In *Deep learning workshop, international conference on machine learning*.

Benjamin, B., et al. (2014). Neurogrid: a mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, *102*, 699–716.

Birchler, O., Roclin, D., Gamrat, C., & Querlioz, D. (2013). Design exploration methodology for memristor-based spiking neuromorphic architectures with the xnet event-driven simulator. In *Proceedings of the 2013 IEEE/ACM international symposium on nanoscale architectures* (pp. 7–12). IEEE Press.

Cao, Y., Chen, Y., & Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, *113*(1), 54–66.

Cao, Y., & Grossberg, S. (2012). Stereopsis and 3d surface perception by spiking neurons in laminar cortical circuits: a method for converting neural rate models into spiking models. *Neural Networks*, *26*, 75–98.

Cassidy, A. S., Merolla, P., Arthur, J. V., Esser, S. K., Jackson, B., Alvarez-Icaza, R., et al. Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores. In *The 2013 international joint conference on neural networks* (pp. 1–10).

Clerc, M. (2012). Standard particle swarm optimisation. 15 pages. [Online] Available https://hal.archives-ouvertes.fr/hal-00764996.

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., & Bengio, Y. (2016). Binarized neural networks: training deep neural networks with weights and activations constrained to+ 1 or-1. arXiv preprint arXiv:1602.02830.

Davies, M., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro*, *38*(1), 82–99.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., et al. (2017). Openai baselines. https://github.com/openai/baselines.

Diehl, P., & Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, *9*, 99.

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S., & Pfeiffer, M. (2015). Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 international joint conference on neural networks* (pp. 1–8).

Diehl, P. U., Pedroni, B. U., Cassidy, A., Merolla, P., Neftci, E., & Zarrella, G. (2016). Truehappiness: Neuromorphic emotion recognition on truenorth. In *2016 international joint conference on neural networks* (pp. 4278–4285).

Ferre, P., Mamalet, F., & Thorpe, S. J. (2018). Unsupervised feature learning with winner-takes-all based stdp. *Frontiers in Computational Neuroscience*, *12*, 24.

Furber, S., Galluppi, F., Temple, S., & Plana, L. (2014). The spinnaker project. *Proceedings of the IEEE*, *102*, 652–665.

Gerstner, W. M. K. W. (2002). *Spiking neuron models. single neurons, populations, plasticity*. Cambridge University Press.

Gilra, A., & Gerstner, W. (2018). Non-linear motor control by local learning in spiking neural networks. In J. Dy, & A. Krause (Eds.), *Proceedings of the 35th international conference on machine learning, ser. proceedings of machine learning research: Vol. 80* (pp. 1773–1782). Stockholmsmässan, Stockholm Sweden: PMLR.

Hasselt, H. v., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the thirtieth AAAI conference on artificial intelligence, ser. AAAI'16* (pp. 2094–2100). AAAI Press.

Hazan, H., & Manevitz, L. M. (2012). Topological constraints and robustness in liquid state machines. *Expert Systems with Applications*, *39*(2), 1597–1606.

Hazan, H., Saunders, D. J., Khan, H., Sanghavi, D. T., Siegelmann, H. T., & Kozma, R. (2018). Bindsnet: a machine learning-oriented spiking neural networks library in python. *Frontiers in Neuroinformatics*, *12*, 89.

Huang, S. H., Papernot, N., Goodfellow, I. J., Duan, Y., & Abbeel, P. (2017). Adversarial attacks on neural network policies. *CoRR*, *abs/170202284*.

Huh, D., & Sejnowski, T. J. (2018). Gradient descent for spiking neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems: Vol. 31* (pp. 1440–1450). Curran Associates, Inc.

Kozma, R., Pino, R. E., & Pazienza, G. E. (2012). *Advances in neuromorphic memristor science and applications*. Springer Science and Business Media.

Leveille, J., Versace, M., & Grossberg, S. (2010). Running as fast as it can: how spiking dynamics form object groupings in the laminar circuits of visual cortex. *Journal of Computational Neuroscience*, *28*, 323–346.

Maass, W. (1996). Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, *10*, 1659–1671.

Martí, D., Rigotti, M., Seok, M., & Fusi, S. (2016). Energy-efficient neuromorphic classifiers. *Neural Computation*, *28*, 2011–2044.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

Pedroni, B., Das, S., Arthur, J., Merolla, P., Jackson, B., Modha, D., et al. (2016). Mapping generative models onto a network of digital spiking neurons. *IEEE Transactions on Biomedical Circuits and Systems*, *10*, 837–854.

Perez-Carrasco, J. A., Zhao, B., Serrano, C., Acha, B., Serrano-Gotarredona, T., Chen, S., & Linares-Barranco, B. (2013). 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 and Machine Intelligence*, *35*(11), 2706–2719.

Pfeiffer, M., & Pfeil, T. (2018). Deep learning with spiking neurons: opportunities and challenges. *Frontiers in Neuroscience*, *12*, 774.

Richardson, M. J. E., & Gerstner, W. (2006). Statistics of subthreshold neuronal voltage fluctuations due to conductance-based synaptic shot noise. *Chaos. An Interdisciplinary Journal of Nonlinear Science*, *16*(2), 026106.

Rueckauer, B., Lungu, I.-A., Hu, Y., & Pfeiffer, M. (2016). Theory and tools for the conversion of analog to spiking convolutional neural networks. arXiv preprint arXiv:1612.04052.

Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., & Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, *11*, 682.

Schultz, W. (2016). Dopamine reward prediction-error signalling: a two-component response. *Nature Reviews Neuroscience*, *17*, 183–195.

Sharmin, S., Panda, P., Sarwar, S. S., Lee, C., Ponghiran, W., & Roy, K. (2019). A comprehensive analysis on adversarial robustness of spiking neural networks. In *Proceedings of the 2019 INNS/IEEE international joint conference on neural networks*. IEEE Press, arXiv:1905.02704.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

Srinivasa, N., & Cruz-Albert, J. (2012). Neuromorphic adaptive plastic scalable electronics: analog learning systems. *IEEE Pulse*, *3*(1), 51–56.

Stein, R. B., Gossen, E. R., & Jones, K. E. (2005). Neuronal variability: noise or part of the signal?. *Nature Reviews Neuroscience*, *6*, 389 EP –, review Article.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). Cambridge, MA, USA: MIT Press.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2818–2826).

Tuckwell, H. C. (1988). *Cambridge studies in mathematical biology: Vol. 2, Introduction to theoretical neurobiology*. Cambridge University Press.

Versace, M., & Grossberg, S. (2008). Spikes, synchrony, and attentive learning by laminar thalamocortical circuits. *Brain Research*, *1218*, 278–312.

Wang, J. X., Kurth-Nelson, Z., Kumaran, D., Tirumala, D., Soyer, H., Leibo, J. Z., Hassabis, D., & Botvinick, M. M. (2018). Prefrontal cortex as a meta-reinforcement learning system. *Nature Neuroscience*, *21*, 1–9.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., & Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In M. F. Balcan, & K. Q. Weinberger (Eds.), *Proceedings of the 33rd international conference on machine learning, ser. proceedings of machine learning research: Vol. 48* (pp. 1995–2003). New York, New York, USA: PMLR.

Witty, S., Lee, J. K., Tosch, E., Atrey, A., Littman, M., & Jensen, D. (2018). Measuring and characterizing generalization in deep reinforcement learning. arXiv preprint arXiv:1812.02868.

Wu, Y., Deng, L., Li, G., Zhu, J., & Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in Neuroscience*, *12*, 331.

Zenke, F., & Ganguli, S. (2018). Superspike: supervised learning in multilayer spiking neural networks. *Neural Computation*, *30*(6), 1514–1541.