Porting Deep Spiking Q-Networks to neuromorphic chip Loihi

Mahmoud Akl
mahmoud.akl@tum.de
Technical University Munich
Munich, Germany

Yulia Sandamirskaya
yulia.sandamirskaya@intel.com
Intel Labs
Munich, Germany

Florian Walter
florian.walter@tum.de
Technical University Munich
Munich, Germany

Alois Knoll
knoll@in.tum.de
Technical University Munich
Munich, Germany

ABSTRACT

Deep neural networks (DNNs) set the benchmark in many tasks in perception and control. Spiking versions of DNNs, implemented on neuromorphic hardware can enable orders of magnitude lower power consumption and low latency during network use. In this paper, we explore behavior and generalization capability of spiking, quantized spiking, and hardware implementation of deep Q-networks in two classical reinforcement learning tasks. We found that spiking neural networks have slightly decreased performance compared to non-spiking network, but we can avoid performance degradation from quantization and in-chip implementation. We conclude that since hardware implementation leads to lower power consumption and low latency, neuromorphic approach is a promising avenue for deep Q-learning. Furthermore, online learning, enabled in neuromorphic chips, can be used to compensate for the performance decrease in environments with parameter variations.

CCS CONCEPTS

• **Computing methodologies** → **Reinforcement learning**.

KEYWORDS

Spiking neural networks, reinforcement learning, neuromorphic hardware

ACM Reference Format:
Mahmoud Akl, Yulia Sandamirskaya, Florian Walter, and Alois Knoll. 2021. Porting Deep Spiking Q-Networks to neuromorphic chip Loihi. In *International Conference on Neuromorphic Systems, July 27–29, 2021, Virtual Conference*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Spiking neural networks (SNNs), sometimes referred to as the third generation of neural networks [14], mimic the behavior of biological neurons more closely than the more common analog neural networks (ANNs) that are prevalent in the state-of-the-art machine learning. In particular, the neuron model in SNNs has an explicit temporal dynamics and the activation levels are communicated in the network using asynchronous pulses called spikes. SNNs not only offer the potential to better understand computing in the brain, but also to benefit from its unique computational properties in AI applications. In particular, the communication through spikes enables asynchronous computation without a shared clock and can be implemented extremely efficiently in hardware [5, 6, 20].

Thus, SNNs hold the potential of building AI systems that are more brain-like in terms of online learning, low latency, fast inference, and energy efficiency. Until recently, the focus of SNN research has been on either converting trained ANNs to SNNs [24], or on training SNNs using biologically plausible local learning rules such as spike-timing dependent plasticity (STDP) [23]. In the past few years, new techniques to approximate gradients for SNNs were developed [11, 26]. These make gradient-based learning applicable to SNNs. SNN performance on benchmark datasets such as MNIST [4, 7, 10, 25, 29, 32, 34], CIFAR-10 [12], or ImageNet [21, 22, 28] is constantly improving, which inspired us to study their performance in the deep reinforcement learning setting.

Deep Reinforcement learning (RL) is particularly promising learning framework for robotics and autonomous systems [9]. For many robotic tasks, large datasets that are required for classical supervised learning and would capture the peculiarities of the required robot or scenario are lacking, but a simulation of the robot's physics is usually available and can be used to generate the required data using the RL paradigm. This approach has recently shown impressive results with convincing sim-to-real transfer of challenging behaviors (drone racing, quadruped control) learned in simulation with RL [8, 27].

Since work on using SNNs in RL is much less developed than in supervised-learning settings, we explore this combination here. In particular, we applied recent backpropagation-based learning algorithms for SNNs to train deep spiking Q-networks (DSQNs) with reinforcement learning on two classic control problems from the two OpenAI Gym environment [3]. We trained a DSQN with quantized parameters for deployment on the Intel's neuromorphic research chip Loihi [5]. Comparisons with a DSQN running on a standard processor and a baseline DQN in environments with random initialization seeds revealed that the network runs on Loihi without loss in performance in terms of accumulated over an episode reward. To assess the robustness of the models, we also evaluated them

ICONS '21, July 27–29, 2021, Virtual Conference

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

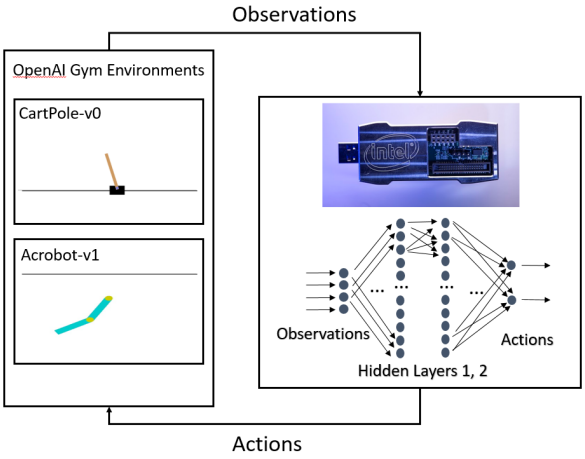


Figure 1: Schematics of the system used in our experiments. The OpenAI Gym environments run in closed-loop with the Intel neuromorphic research chip Loihi. The networks used in our experiments consist of two fully connected hidden layers. The detailed network architecture used for each experiment is listed in Table 2.

on randomized environments from the Sunblaze library [17], in which the physical parameters of the environment are drawn from different distributions.

Our study shows that the DSQN method can be used to train spiking networks that run on neuromorphic processors without loss in training and test performance. Performance of both DSQN and DQN degrades when the environmental parameters are perturbed during inference. Online learning on neuromorphic hardware can be used to cope with the problem of such environment change, e.g. in transition from simulation to real world, although we don't address this challenge in this paper.

2 RELATED WORK

Reinforcement Learning with SNNs has been previously explored through biological learning rules like reward-modulated STDP (R-STDP). An SNN was trained with R-STDP in [2] to make a vehicle keep its lane and in [33] for target reaching with a robotic arm. However, R-STDP only works with shallow networks and may not be as powerful as modern deep reinforcement learning methods in more complex tasks.

More recently, after ANN to SNN conversion techniques yielded more stable results in supervised learning tasks, the same techniques were tried out for DQNs. In [19], a DQN trained to play the Atari Breakout game was converted to an SNN and showed improved robustness against perturbations, e.g. when obscuring a part of the game screen. We haven't observed this increased robustness in our experiments.

In another work, a spiking version of the Deep Deterministic Policy Gradient (DDPG) algorithm was trained and deployed on the Loihi neuromorphic chip [31]. Here, only the actor network was an SNN, while the critic network was a regular DNN. In our implementation, we considered spiking networks both for the policy and target networks of the Q-learning algorithm.

Table 1: DQN algorithm hyperparameters

Hyperparameter	Value
replay memory size	10000
discount factor	0.999
learning rate	0.001
target network update frequency	100
initial exploration	1.0
final exploration	0.05
batch size	128

Table 2: SNN parameters

Hyperparameter	Cartpole-v0	Acrobot-v1
α	1	1
β	1	1
threshold	0.1	1.5
quantized threshold	64	128
simulation time	10	4
network architecture	[4, 64, 64, 2]	[6, 256, 256 ,3]

3 METHODS

In this section, we describe the Q-learning algorithm used to train the DQN, the CartPole and Acrobot problems, the details of training a DSQN, and the adjustment to training procedure required to enable the deployment on Loihi.

3.1 Q-Learning

Q-learning is a model-free off-policy reinforcement learning algorithm that estimates the optimal action-value function

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

that defines the expected discounted future reward when taking action a from state s. The Q-values are updated iteratively after each interaction with the environment by moving the current Q-value towards the experienced reward plus the (discounted) maximum Q-value over all actions in the next state s':

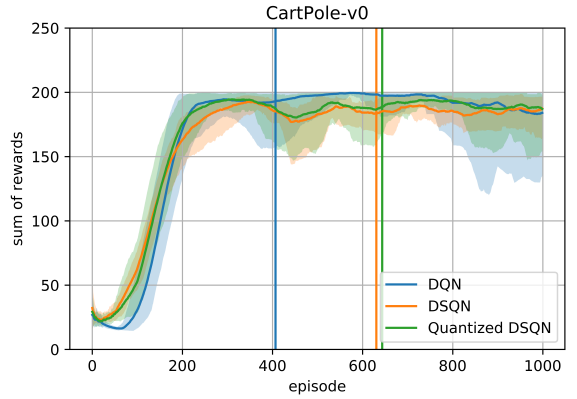
$$Q(s,a) := Q(s,a) + \alpha(-Q(s,a) + r + \gamma \max_{a'} Q(s',a')). \quad (2)$$

When using this approach to solve problems with continuous observation spaces, it becomes intractable to maintain an action value for each state. The Deep Q-Learning algorithm [15], instead, represents the optimal action-value function as a neural network.

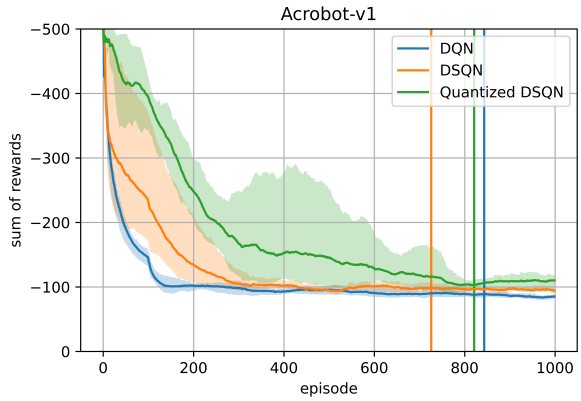
The Deep Q-Learning algorithm consists of two parts: a sampling and a learning part. During sampling, the agent interacts with the environment and collects experience tuples that consist of the current state, the action taken, the reward achieved, and the next state: $\{S, A, R, S'\}$. The experience tuples are stored in a finite replay memory buffer. In the learning step, experiences are randomly sampled from the memory buffer, and the following quadratic loss function is minimized:

$$L(s,a|\theta_i) = (r + \gamma \max_{a'} Q(s',a'|\theta_i^-) - Q(s,a|\theta_i))^2, \quad (3)$$

where θ_i are the parameters of the Q-network, and θ_i^- are the parameters of the target network. The target network parameters



(a)



(b)

Figure 2: Training: as sum of rewards over training episodes for the CartPole (a) and the Acrobot (b) environments. The solid lines are rewards (window size of 100 episodes) averaged over 5 runs with random initialization seeds. Shaded areas show the standard deviation. The vertical solid lines represent the episode at which the best trained model was achieved.

θ_i^- are set to the Q-network parameters every few episodes. This is also known as the target network update frequency and improves stability during training.

3.2 The CartPole and Acrobot Problems

3.2.1 CartPole. The CartPole problem is a classic problem in the reinforcement learning literature [1] and consists of an un-actuated rigid pole hinged to a cart. The cart can move left and right on a one-dimensional track and the pole is free to move only in the plane vertical to the cart and track. The task is to keep the pole balanced by applying a force of +1 or -1 to the cart. At each time step, the agent is given four observations from the environment: the cart's position, the cart's velocity, the pole's angle, and the pole's velocity. The agent receives a reward of +1 for every time step that the pole remains upright. In our experiments, an episode is terminated when the pole falls beyond 15 degrees from the vertical, or when the maximum time step 200 is reached.

3.2.2 Acrobot. The Acrobot problem is also a classic reinforcement learning problem [30]. It consists of a two-link robot, with an actuated joint between the two links. The task is to swing up the lower link to reach a certain height as fast as possible. At each time step, the agent is given six observations from the environment: $(cos(\theta_1), sin(\theta_1), cos(\theta_2), sin(\theta_2), \dot{\theta}_1, \dot{\theta}_2)$, where θ_1 and θ_2 are the joints' angles, and chooses to either apply a positive, negative, or no torque on the joint between the two links. The agent receives a reward of -1 for each time step. In our experiments, an episode is terminated when the lower link reaches the required height, or when the maximum time step 500 is reached.

3.3 Training a DSQN

To train a DSQN with backpropagation and surrogate gradients, we used the SpyTorch framework [16], which is based on the PyTorch deep learning library [18]. In SpyTorch, the synaptic currents (input to the neurons) are calculated according to a standard leaky integrate-and-fire model:

$$u_i(t) = \alpha u_i(t-1) + \sum_j w_{ij} s_j(t), \tag{4}$$

where $\alpha \in [0, 1]$ is the current decay factor, w_{ij} are synaptic weights, and $s_j(t)$ is a binary function, representing the emission of a spike from neuron j. The membrane potential dynamics is in turn calculated using equation:

$$v_i(t) = \beta v_i(t-1) + u_i(t-1), \tag{5}$$

where $\beta \in [0, 1]$ is the membrane potential decay factor. If the membrane potential exceeds the threshold, we set the potential back to zero. We chose the reset-to-zero mechanism since it is utilized by Loihi. The details of implementation of neuronal model on Loihi can be found in [13].

The system architecture used to solve both problems is shown in Figure 1, while the network shape and parameters are listed in Table 2. The observations from the environments are multiplied by the first weight matrix (input to hidden layer 1), and the resulting vector is injected as current in the first hidden layer. We set the threshold of the output layer to infinity, i.e. remove the spiking mechanism from the output neurons, and read out the membrane potentials of the output neurons as the Q-values (note, spike-rate or inter-spike interval could be used as the output as well). We reset the network state after every inference step: since temporal dynamics is not part of the original DQN network, we aim at a more clear comparison this way. Taking temporal dynamics into account could potentially improve performance of the SNN in tasks with temporal contingencies.

3.4 Model Quantization

To be able to deploy the trained SNN on Loihi, we need to take the Loihi model constraints into account. Since Loihi only accepts 8-bit quantized weights, we quantized the weights for the forward pass during training, i.e. during action selection, and kept the floating point weights for the backward pass of the training. Quantizing the weights was done according the to following formula:

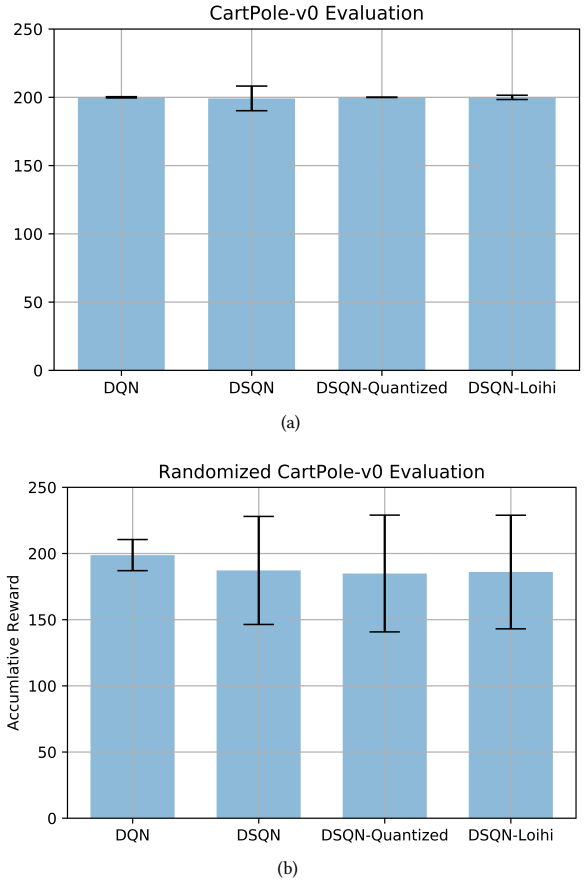


Figure 3: CartPole evaluation on normal (a) and random (b) environments. Average accumulated rewards over 500 runs with random initialization seeds.

$$w_{quant} = \lfloor \frac{w_{float}}{scale} \rfloor, \quad scale = \frac{max(w_{float}) - min(w_{float})}{q_{max} - q_{min}}, \quad (6)$$

where q_{max} and q_{min} are the maximum and minimum bits. Since we are using mixed sign weights on Loihi, we set $q_{min} = -256$ and $q_{max} = 254$.

Additionally, we multiply the quantized weights by 2^6 statically, since we inject the observations as current, and the Loihi compartment current is calculated according to:

$$u_i(t) = u_i(t-1) \cdot (2^{12} - \delta^u) \cdot 2^{-12} + 2^{6+wgtExp} \cdot \sum_j w_{ij} \cdot s_j(t), \quad (7)$$

where δ^u parametrizes the decay term of the input current $u_i(t)$, and $wgtExp$ is an additional weight exponent that is set to zero by default.

4 RESULTS AND EVALUATION

We trained a DQN and a DSQN with the same feed-forward network architecture that consists of two fully connected hidden layers (Table 2), and the same DQN hyperparameters (Table 1) for the same number of episodes to solve the CartPole-v0 and the Acrobot-v1 tasks. Furthermore, we trained a quantized DSQN model for both problems and deployed it to the Loihi neuromorphic chip. In each case (DQN, DSQN, and quantized DSQN), we trained five different models and measured the mean rewards during training and evaluation. Training results for both environments are shown in Figure 2[1].

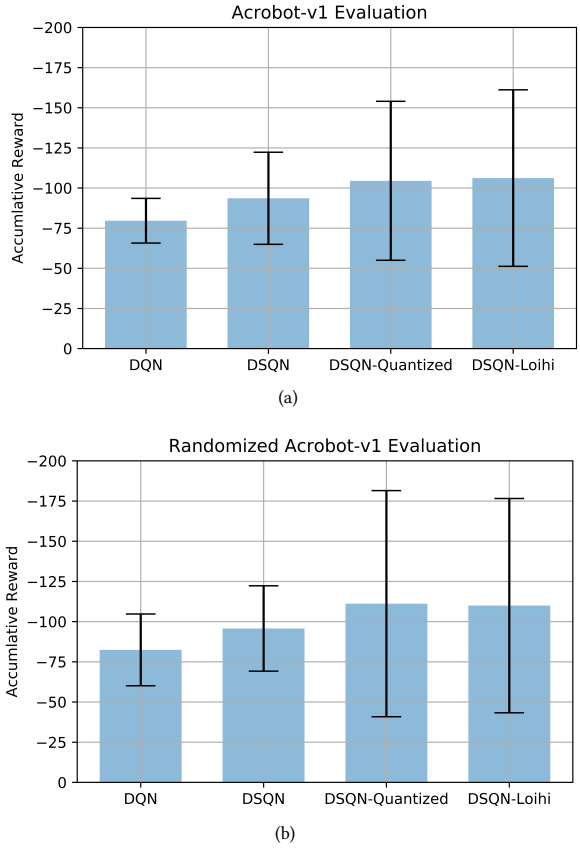
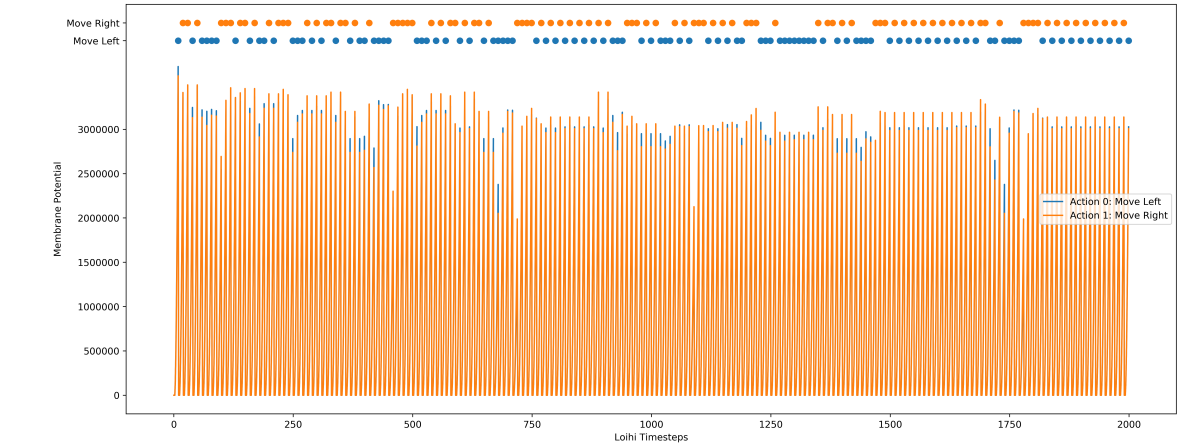


Figure 4: Acrobot evaluation on normal (a) and random (b) environments. Average accumulated rewards over 500 runs with random initialization seeds.

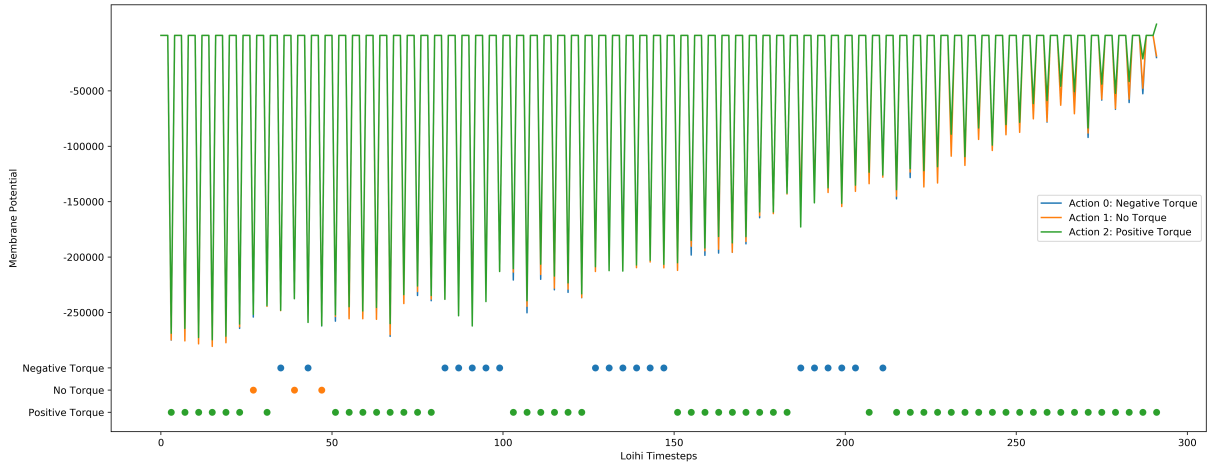
Table 3: Random Environment Details

Environment	Parameter	Default	Random
CartPole	Force	10	[5,15]
	Length	0.5	[0.25,0.75]
	Mass	0.1	[0.05,0.5]
Acrobot	Length	1	[0.75,1.25]
	Mass	1	[0.75,1.25]
	MOI	1	[0.75,1.25]

[1]All experiments were run on a NUC computer featuring Intel® Core™ i7-8559U CPU 2.70GHz × 8 with 32GB RAM running Ubuntu 18.04, python 3.6.9, and version 0.9.9 of the Intel NxSDK. All performance measurements are based on testing as of April 2021 and may not reflect all publicly available security updates.



(a)



(b)

Figure 5: Output neuronal activity for CartPole (a) and Acrobot (b) recorded for one episode on Loihi. Horizontal axis: the number of time steps of the episode multiplied by the simulation time of each trial, listed in Table 2. The dots indicate which action was selected at each time step, based on neuron with the higher membrane potential. In the example runs shown here, the CartPole episode lasted 200 time steps (reward = 200); the Acrobot episode lasted 72 time steps (reward = -72).

Moreover, we evaluated the trained models on 100 randomly initialized environments from the Sunblaze library to further test the generalization ability of the trained models. The randomized environments vary some physical parameters, e.g. the mass and length of the pole and the strength of the applied force in case of the CartPole environment and the length, mass and moment of inertia in case of the Acrobot environment. The values for these parameters are sampled from specific intervals provided in Table 3.

Figure 2 shows the results of the training phase for both environments. In the CartPole environment, we observed that there was no significant difference between the DQN, DSQN and the quantized DSQN (Figure 2(a)). The most notable difference is that the DQN requires fewer episodes to reach the best reward (indicated by the vertical lines in the figure). The evaluation results of the trained CartPole models are shown in Figure 3. In the non-altered environments, all four models act very similarly and achieve the best possible reward, in that case 200. In the randomized environments, however, we notice a slight degradation of performance and increased variability.

During training on the Acrobot task, we observed that the spiking models were more noisy and learned slower than the DQN. Here, quantization also contributes to higher variability and slower learning as visible in Figure 2(b). This can be attributed to the increased units per layer for the Acrobot environment (256 vs. 64), which increases accumulation of discretization errors. However, all models reach a similar reward by the end of the training episodes. The evaluation results of the trained Acrobot models are shown in

Figure 4. Here we observe that the DQN outperforms the spiking models.

A consistent result from both experiments is that there is no loss of performance between the quantized DSQN simulated on SpyTorch and the DSQN running on Loihi.

Figure 5 shows an exemplary run of an episode on Loihi for both environments, depicting activity in the output layer and the selected actions. In both examples, the neuromorphic RL agent could successfully solve the task.

5 DISCUSSION AND CONCLUSION

In this work, we successfully trained SNNs with the deep Q-learning algorithm using backpropagation and surrogate gradients to solve classic control problems from OpenAI Gym. With some additional considerations, we were able to deploy the trained weights to the Loihi neuromorphic chip and run the simulated environments in closed-loop with the hardware without accuracy loss. Measure energy consumption or latency was not practical in out RL setup, but we can expect that neuromorphic implementation can bring significant advantages here [6].

Furthermore, we tested the trained DSQN models for robustness against perturbations in the physical parameters of the environments. Unlike results from [19], we observed a slight degradation of performance when testing the DSQNs with perturbed environments (see Figures 3). However, backpropagation provides a good starting point that can be further fine-tuned with the online learning capabilities of SNNs. This can be a potential solution for better generalization abilities, which is a current problem in reinforcement learning.

Due to the additional parameters in SNNs, e.g. the membrane threshold, the simulation time and the voltage and current decay factors, training SNNs with backpropagation requires a more extensive hyperparameter tuning phase, as changing the values for those parameters can have a substantial impact on the results.

This work paves the way for exploring training deep reinforcement learning algorithms with SNNs and deploying them to neuromorphic hardware. With the rising popularity of deep reinforcement learning in fields like robotics, we believe that neuromorphic implementations will lead to an energy-efficient and potentially more robust alternative and will broaden the application domains of neuromorphic hardware.

ACKNOWLEDGEMENT

We thank Intel's Neuromorphic Research Community for fruitful discussions and Andreas Wild and Philipp Plank for their help with porting the networks on Loihi.

REFERENCES

[1] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. 1983. Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. *IEEE Transactions on Systems, Man and Cybernetics* SMC-13, 5 (1983), 834–846. https://doi.org/10.1109/TSMC.1983.6313077

[2] Zhenshan Bing, Claus Meschede, Kai Huang, Guang Chen, Florian Rohrbein, Mahmoud Akl, and Alois Knoll. 2018. End to End Learning of Spiking Neural Network Based on R-STDP for a Lane Keeping Vehicle. *Proceedings - IEEE International Conference on Robotics and Automation* (2018), 4725–4732. https://doi.org/10.1109/ICRA.2018.8460482

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. (2016), 1–4. arXiv:1606.01540 http://arxiv.org/abs/1606.01540

[4] Gregory K. Cohen, Garrick Orchard, Sio-Hoi Leng, Jonathan Tapson, Ryad B. Benosman, and André van Schaik. 2016. Skimming Digits: Neuromorphic Classification of Spike-Encoded Images. *Frontiers in Neuroscience* 10 (2016), 184. https://doi.org/10.3389/fnins.2016.00184

[5] Mike Davies, Narayan Srinivasa, Tsung Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang. 2018. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro* 38, 1 (2018), 82–99. https://doi.org/10.1109/MM.2018.112130359

[6] Mike Davies, Andreas Wild, Garrick Orchard, Yulia Sandamirskaya, Gabriel A. Fonseca Guerra, Prasad Joshi, Philipp Plank, and Sumedh R. Risbud. 2021. Advancing Neuromorphic Computing With Loihi: A Survey of Results and Outlook. *Proc. IEEE* 109, 5 (2021), 911–934. https://doi.org/10.1109/JPROC.2021.3067593

[7] Peter U Diehl and Matthew Cook. 2015. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience* 9 (2015), 99.

[8] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. 2019. Learning agile and dynamic motor skills for legged robots. *Science Robotics* 4, 26 (2019).

[9] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. 2021. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research* (2021), 0278364920987859.

[10] Laxmi R Iyer and Arindam Basu. 2017. Unsupervised learning of event-based image recordings using spike-timing-dependent plasticity. In *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1840–1846.

[11] Jacques Kaiser, Hesham Mostafa, and Emre Neftci. 2020. Synaptic plasticity dynamics for deep continuous local learning (DECOLLE). *Frontiers in Neuroscience* 14 (2020), 424.

[12] Chankyu Lee, Gopalakrishnan Srinivasan, Priyadarshini Panda, and Kaushik Roy. 2018. Deep spiking convolutional neural network trained with unsupervised spike-timing-dependent plasticity. *IEEE Transactions on Cognitive and Developmental Systems* 11, 3 (2018), 384–394.

[13] Chit-Kwan Lin, Andreas Wild, Gautham N Chinya, Yongqiang Cao, Mike Davies, Daniel M Lavery, and Hong Wang. 2018. Programming spiking neural networks on Intel's Loihi. *Computer* 51, 3 (2018), 52–61.

[14] Wolfgang Maass. 1997. Networks of spiking neurons: the third generation of neural network models. *Neural networks* 10, 9 (1997), 1659–1671. https://doi.org/10.1016/S0893-6080(97)00011-7

[15] 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. 2015. Human-level control through deep reinforcement learning. *Nature* 518 (2015). https://doi.org/10.1038/nature14236 arXiv:1604.03986

[16] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. 2019. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine* 36, 6 (2019), 51–63.

[17] Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. 2018. Assessing Generalization in Deep Reinforcement Learning. arXiv:arXiv:1810.12282

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703* (2019).

[19] Devdhar Patel, Hananel Hazan, Daniel J Saunders, Hava T Siegelmann, and Robert Kozma. 2019. Improved robustness of reinforcement learning policies upon conversion to spiking neuronal network platforms applied to Atari Breakout game. *Neural Networks* 120 (2019), 108–115.

[20] Bipin Rajendran, Abu Sebastian, Michael Schmuker, Narayan Srinivasa, and Evangelos Eleftheriou. 2019. Low-power neuromorphic hardware for signal processing applications: A review of architectural and system-level design approaches. *IEEE Signal Processing Magazine* 36, 6 (2019), 97–110.

[21] Nitin Rathi and Kaushik Roy. 2020. Diet-snn: Direct input encoding with leakage and threshold optimization in deep spiking neural networks. *arXiv preprint arXiv:2008.03658* (2020).

[22] Nitin Rathi, Gopalakrishnan Srinivasan, Priyadarshini Panda, and Kaushik Roy. 2020. Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. *arXiv preprint arXiv:2005.01807* (2020).

[23] Saeed Reza, Mohammad Ganjtabesh, Simon J Thorpe, and Timothée Masquelier. 2018. STDP-based spiking deep convolutional neural networks for object recognition. *Neural Networks* 99 (2018), 56–67. https://doi.org/10.1016/j.neunet.2017.12.005

[24] Bodo Rueckauer, Iulia Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih Chii Liu. 2017. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience* 11 (2017), 1–12. https://doi.org/10.3389/fnins.2017.00682

[25] Amar Shrestha, Khadeer Ahmed, Yanzhi Wang, and Qinru Qiu. 2017. Stable spike-timing dependent plasticity rule for multilayer unsupervised and supervised learning. In *2017 international joint conference on neural networks (IJCNN)*. IEEE, 1999–2006.

[26] Sumit Bam Shrestha and Garrick Orchard. 2018. SLAYER: spike layer error reassignment in time. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 1419–1428.

[27] Yunlong Song, Mats Steinweg, Elia Kaufmann, and Davide Scaramuzza. 2021. Autonomous Drone Racing with Deep Reinforcement Learning. *arXiv preprint arXiv:2103.08624* (2021).

[28] Christoph Stöckl and Wolfgang Maass. 2021. Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes. *Nature Machine Intelligence* (2021), 1–9.

[29] Evangelos Stromatias, Miguel Soto, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. 2017. An Event-Driven Classifier for Spiking Neural Networks Fed with Synthetic or Dynamic Vision Sensor Data. *Frontiers in Neuroscience* 11 (2017), 350. https://doi.org/10.3389/fnins.2017.00350

[30] Richard S Sutton. 1996. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems* (1996), 1038–1044.

[31] Guangzhi Tang, Neelesh Kumar, and Konstantinos P. Michmizos. 2020. Reinforcement co-Learning of Deep and Spiking Neural Networks for Energy-Efficient Mapless Navigation with Neuromorphic Hardware. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 6090–6097. https://doi.org/10.1109/IROS45743.2020.9340948

[32] Amirhossein Tavanaei and Anthony Maida. 2019. BP-STDP: Approximating backpropagation using spike timing dependent plasticity. *Neurocomputing* 330 (2019), 39–47.

[33] Juan Camilo Vasquez Tieck, Pascal Becker, Igor Peric, Jacques Kaiser, Mahmoud Akl, Daniel Reichard, Arne Roennau, and Rüdifer Dillmann. 2019. Learning target reaching motions with a robotic arm using dopamine modulated STDP. In *18th IEEE International Conference on Cognitive Informatics and Cognitive Computing*.

[34] Ruthvik Vaila, John Chiasson, and Vishal Saxena. 2019. Deep convolutional spiking neural networks for image classification. *arXiv preprint arXiv:1903.12272* (2019).